Synchronization in WebSmart Multimedia Authoring System

By

Lan Li

B.Sc. (Physics) Peking University, China
M.Sc. (Physics) The University of Calgary, Canada

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

© Lan Li, 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 allows without my written permission.

Department of Computer Science

The University of British Columbia

2366 Main Mall

Vancouver, BC, Canada

V6T 1Z4

Date: 

[signature]

Apr. 29, 1998
Abstract

While the Web has become the universal choice for on-line information exchange, the Internet is preparing for the next generation of Web document: synchronized multimedia. Compared to a year before, video contents on Web have been tripled. Most of the contents are media-on-demand news and entertainment clips played in Web browsers by plug-ins. However, simple playback of videos does not fully utilize the superiorities of digital video over analog video, and active computer over passive TV. Digitized video has the potential of being synchronized with other desktop applications to create interactive presentations. To realize this possibility, a comprehensive framework is required for multimedia authoring, delivering and presentation.

We developed and herein present WebSmart, an authoring system for bringing synchronized multimedia into the Web. The system is comprised of four components: a visual Editor that creates interactive documents by point-and-click, a CaptionMaker that synchronizes closed caption with video/audio, a QuizDesigner that enables two-way interactions between video and surrounding objects, and an Applet that presents the documents in the end users’ Web browsers.
Synchronization is the core issue at both the authoring and presentation stages of multimedia documents. We developed simple but effective algorithms for different synchronization requirements: coarse-grained HTML flipping and finer-grained closed captioning. Flexibility and efficiency are the major concerns of the WebSmart synchronization models. This thesis will be focused on the synchronization techniques of the WebSmart system.
Table of Contents

Abstract

Table of Contents

List of Tables

List of Figures

Acknowledgment

Chapter 1
Introduction

1.1 Problem and Motivation

1.1.1 Synchronized Multimedia on Web

1.1.2 One Solution: browser plug-ins

1.1.3 Another Solution: making standards

1.1.4 WebSmart: taking in Java

1.2 Synchronization Framework

1.2.1 JMF: display the continuous media

1.2.2 LiveConnect: express the synchronization

1.3 Contributions

1.4 Thesis Subject and Outline

Chapter 2
System Overview

2.1 Multimedia Objects and Documents

2.2 Physical Components

iv
Chapter 7
Conclusion and Future Work

7.1 Remaining Challenges ........................................... 85
7.2 Future Work ......................................................... 85
  7.2.1 Streaming server ........................................... 85
  7.2.2 Supporting SMIL .............................................. 86
  7.2.3 Live Documentation and Live Broadcast ................. 86
7.3 Status and Availability ........................................... 87
7.4 Summary ........................................................... 87

References .............................................................. 90

Appendix A
Web Course Document Samples .................................. 93
  A.1 Spatial layout: app1.html ..................................... 93
  A.2 Hyperlinks: syllabus.html ................................... 94
  A.3 A section file: Section2.html ............................... 95
  A.4 A subsection file: Section2.2.html ....................... 97
  A.5 Quiz section file: section3.html ........................... 99
  A.6 A question slide: q1.html ................................... 101
  A.7 The last question slide: q3.html ............................ 103
  A.8 Quiz summary slide: quiz_summary.html ................. 105
List of Tables

Table 2.1 WebSmart Authoring Tools .............................................. 18
Table 4.1 Objects and Their Exported Files ................................. 56
Table 5.1 Reaction to Media Events ................................................ 75
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Synchronization Framework of WebSmart</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Physical Model of WebSmart</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Flipping HTML Scenario</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Closed Captioning Scenario</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>WebSmart Editor and Player</td>
<td>57</td>
</tr>
<tr>
<td>4.2</td>
<td>Point-and-click Slide Timing</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>Reserve a pause event in a question slide</td>
<td>62</td>
</tr>
<tr>
<td>4.4</td>
<td>WebSmart CaptionMaker</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>A Web course application</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>Presentation Control Scenario</td>
<td>70</td>
</tr>
<tr>
<td>5.3</td>
<td>Event Sorting and Dispatching</td>
<td>72</td>
</tr>
</tbody>
</table>
Acknowledgment

I would like to acknowledge the following people for their contributions to this thesis. My supervisor, Dr. Gerald Neufeld, is thoroughly involved in this thesis. Without his financial support, his guidance, and his patience to answer my various questions, this thesis would not exist. I appreciate the many constructive opinions and creative work from my partner, Yue Xu, who has made this teamwork project a valuable experience. I would also like to thank Dr. Norm Hutchinson, for reading my thesis and offering me suggestions. Finally, I must thank my parents and my husband Li An, for their support and unconditional love.

Lan Li

April, 1998
Chapter 1
Introduction

Today, computers, disks and networks are fast enough to process, store and transmit very large video/audio files as well as the traditional text and image files. The integration of a variety of media types defines multimedia. Multimedia content that is both interactive and continuous is called synchronized multimedia [1]. Synchronized multimedia environments are transforming human-computer interactions and allowing a new family of products that may drive the second information revolution. Applications of synchronized multimedia include distance learning, corporate training, product demonstration, interactive advertisement, and electronic presentation.

The creation of a synchronized multimedia document involves combining different media types/applications and maintaining their temporal and spatial relationship. An effective authoring environment is essential for this task. WebSmart provides such an environment. The system consists of two sub-systems:

- **Authoring system**, including WebSmart Editor and two supplemental tools: CaptionMaker and QuizDesigner:

  Help document designers synchronize and maintain multimedia content in a visually direct way.
• Presentation system, including a Java applet — WebSmart Applet:

Present the synchronized multimedia in the end-users’ Web browsers.

1.1 Problem and Motivation

Authoring Systems, as defined by Locatis, 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.” [2]. A multimedia authoring system must provide the following services:

• An authoring environment that relieves document authors from intensive coding

• A presentation engine that presents the synchronized media

There are a lot of commercial multimedia authoring systems on the market. But most of these systems have the following problems:

• Desktop-centric — the composed documents are stored on disks for local access or on CD-ROMs for distribution. Typical examples are presentations created with tools like Microsoft’s Powerpoint [3] or Macromedia’s Authorware [4]; and CD-ROM products like interactive encyclopedia, training courses, lexica, etc.

• Require installation of specific end-user softwares for viewing the multimedia content.
• Can only run on PC platforms such as Windows or Macintosh, few support UNIX.

The motivation behind WebSmart is to develop a new authoring system that is Web-oriented, platform-neutral, protocol-neutral, and content-neutral.

1.1.1 Synchronized Multimedia on Web

The problems of the current multimedia authoring systems can be summarized by one word: *portability*. Portability not only requires the software to be able to run on different platforms, but also ensures that the multimedia document, regardless of the machine it was authored on, can be viewed on a variety of delivery platforms.

To maximize the document portability, the Web is currently the best environment:

• First, almost every computer has Web browsers. Multimedia documents that can be viewed using Web browsers impose least obligations on the end-users.

• Second, the forthcoming high-speed Internet links make it possible that in the near future Web technology will support the transmission of the content that is distributed on CD-ROMs today.

• Finally, updating content on Web is much easier than producing new CD-ROMs, and distributing content via Web is cheaper than distributing CD-ROMs. As a result, more up-to-date information can be offered via the Web than on a CD-ROM.
However, the current Web technology is limited when it comes to synchronized multimedia. Current Web browsers cannot display continuous media like video by themselves, neither can the Web language, HTML [7], describe synchronization information like "15 seconds into video-clip 1, show slide_1 in the lower left Frame and tile_1 in the lower right Frame;... 37 seconds into video-clip 2, show slide_5...".

There are several approaches to enable synchronized multimedia presentation on the Web, such as depending on some other leading-edge technologies for synchronized presentations, or expanding the capacity of HTML and its browsers.

1.1.2 One Solution: browser plug-ins

The approach used by many products currently on the market is to use browser-compatible plug-ins. Examples are RealNeworks' RealPlayer [11], Macromedia's Shockwave [18], Apple's Quicktime plug-in [19], and the VXtreme plug-in [12].

The advantage of this approach is that it does not overlap with existing Web technologies. Content providers don't have to wait for the long process of format standardization for synchronized multimedia on the Web. They can use the existing authoring tools to create and publish their multimedia Web documents right away. However, it also has a number of drawbacks. First, because the content is in a format unknown to Web search engines and thus cannot be indexed, the potential audience may have difficulty in finding the content. Moreover, the documents composed by different authoring systems requires different plug-ins for presentation, a potential
audience may give up reading the content either because he/she is reluctant to install new plug-ins, or because the specific plug-in doesn’t support his/her platform — as mentioned before, most of the current authoring systems and plug-ins only work on Windows or Macintosh.

1.1.3 Another Solution: making standards

The second approach is to extend the HTML language to support the expression of synchronization and continuous media control. The key for synchronization is to make individual elements of an HTML page addressable. For example, each paragraph in a document could have an individual identifier. This allows such expressions as “10 seconds into the presentation, remove the second paragraph and show the 3rd paragraph...” To the control of audio and video files, [21] proposes to include audio and video in the HTML file using object elements.

Netscape’s Dynamic HTML [9], which is supported in Netacape 4.*, has made some progress on synchronization. Dynamic HTML uses a “layer” tag to address HTML elements, and introduces some syntax for positioning and overlapping layers. However, Dynamic HTML has no syntax to declaratively describe synchronization information. The synchronization is expressed by using Netscape’s scripting language JavaScript [8].

W3C, the organization founded in October 1994 to develop common protocols that enhance the interoperability and promote the evolution of the Web, established a working group on synchronized multimedia in March 1997. The working group has
focussed on the design of a declarative language for describing multimedia presentations referred to as SMIL (pronounced "smile"), which stands for Synchronized Multimedia Integration Language. A major contribution of SMIL is the introducing of two tags, "parallel" and "sequential" for synchronization presentation. By the time of this writing, the specification of SMIL is still a "work in process". The latest version working draft could be found on http://www.w3.org/TR/WD-smil.

The principal advantage of this standard-making approach is that when a standard like SMIL is accepted, it assures a large market and allows products from different vendors to communicate. The principal disadvantage of standards is that a standard tends to freeze the technology. By the time a standard is developed, subjected to review and compromise, new demands arise and more efficient techniques are possible. In these cases, people will go for the plug-in approach to promote new technologies.

1.1.4 WebSmart: taking in Java

WebSmart adopts the plug-in approach. However, unlike other products, WebSmart does not require end-users to install new plug-ins. Instead, it embeds a Java applet (WebSmart Applet) in the documents (Web pages). Applets are Java applications that travel across the Internet or intranet as part of a Web page and run inside of end-user's browser. This mobile-code approach not only relieves the end-users from installing new software, but also solves the portability problem of plug-ins since Java is platform neutral.
The authoring tools of WebSmart (Editor, CaptionMaker and QuizDesigner) are also entirely implemented in Java to guarantee the portability. Portability is not the only reason WebSmart chooses Java. Object-orientation, network-savvy, and multi-threaded support are also essential considerations. Moreover, the recently released Java Media Framework [6] augments Java with multimedia capabilities. All these make Java the natural selection for building a Web-oriented and platform neutral multimedia authoring system.

1.2 Synchronization Framework

To enable synchronized presentation on Web, WebSmart exploits the latest development in Java Media Framework (JMF) [6] and Netscape's LiveConnect [8]. Figure 1.1 shows the technology infrastructure of WebSmart. The whole software, including authoring tools and the presentation applet, is built upon Java. The newly released JMF provides the support for video & audio processing and manipulation. A document produced by the authoring tools is a set of HTML files which embed the WebSmart Applet and some JavaScript functions. The WebSmart Applet plays back the continuous media content (video/audio/closed-caption), and controls the synchronization of the whole document through LiveConnect communications between the Java applet and JavaScript.
1.2.1 JMF: display the continuous media

JMF (Java Media Framework)[6] is a collection of classes that enables the display and capture of multimedia data within Java applications and applets. JMF is being released in three stages: Player, Capture and Conference. By the time of this writing, only Player is released.

JMF player furnishes a platform-neutral, content-neutral and protocol neutral media display framework for WebSmart. Player provides a set of high-level interfaces which allow developers to write applications that display various media types independent of the network protocol, file format, or data format of the media. For example, JMF can present most of the standard media content such as MPEG-1, MPEG-2, QuickTime, AVI, WAV, AU, MIDI and real-time streaming video/audio; media data can come from
reliable sources such as a HTTP or FILE, or streaming sources such as video-on-demand (VOD) servers or Real-Time Transport Protocol (RTP).

Except for media display, JMF’s event model provides an event reporting protocol that enables the WebSmart system to respond to media-driven error conditions, such as out-of-data or resource unavailable conditions. This event model is an essential mechanism for WebSmart’s synchronization models.

1.2.2 LiveConnect: express the synchronization

In the Web documents generated by WebSmart, the multimedia synchronization information is expressed using JavaScript. JavaScript must communicate with the WebSmart applet to realize the synchronization. Netscape’s LiveConnect [8] provides this interactive mechanism.

LiveConnect enables two-way communications between JavaScript and Java applets in a page and between JavaScript and plug-ins loaded on a page. Through LiveConnect, JavaScript can reference an applet embedded in the HTML code, and invoke the methods of the applet object; while a Java applet can get a handle on the JavaScript window and execute JavaScript functions through that handle.

Java, JMF, JavaScript, HTML and LiveConnect provide the synchronization framework of the WebSmart system. Their integration defines a unique characteristic of WebSmart.
1.3 Contributions

The dynamic nature of the synchronization framework determines the flexibility of the synchronization model. Synchronization is the focus of this thesis. The major contributions include:

- Flexible synchronization model

WebSmart allows users to synchronize an unlimited number of media objects within a web page. It not only offers coarse-grained synchronization like flipping HTML, but also supports finer-grained closed captioning. Moreover, the interactions between media objects are mutual. For example, video playback controls the flipping of HTML slides, while the slides can also accept user inputs and issue commands to control the video.

- Closed Captioning

Closed caption is a new type of continuous media content introduced in the WebSmart system. Since no standard has been setup nor published, WebSmart defines its own specification for closed caption, and uses simple but effective algorithms to implement the closed captioning engines.

- Point-and-click synchronization setup

Synchronization setup in the WebSmart authoring environment can be done by point-and-click. New ideas of synchronization can be arranged by customizing the JavaScript functions in the output HTML files.
• Sequential and parallel presentation

WebSmart uses multiple threads to concurrently control the presentation and synchronization of the media objects. Media contents can be presented in sequence or in parallel.

1.4 Thesis Subject and Outline

The remainder of this thesis is organized as follows:

Chapter 2 gives an overview of the WebSmart system, briefly describes its physical components, functionality, and document structure.

Chapter 3 pictures the synchronization models of WebSmart, and defines a specification for closed caption.

Chapter 4 discloses the synchronization setup algorithms at the authoring stage, including scheduling flipping HTML pages and synchronizing closed captions.

Chapter 5 describes the synchronized presentation algorithms of WebSmart Applet.

Chapter 6 surveys the related works on bringing synchronized multimedia into Web, including the standards in progress and some commercial multimedia authoring systems.
Finally, Chapter 7 identifies a number of remaining challenges in our work, presents plans for future works, provides references to our implementation and applications, and concludes the thesis.
Chapter 2
System Overview

WebSmart uses HTML and JavaScript as its authoring languages, and relies on Netscape to browse the document. The document presentation task is handled by the WebSmart Applet, a Java applet that does the media playback and document synchronization control. The applet is embedded in the documents, and automatically migrates to the viewer’s local machine with the documents.

To satisfy the divergent requirements of document designers, WebSmart provides a flexible authoring environment that adapts to different levels of document design. The WebSmart Authoring tools (Editor, CaptionMaker and QuizDesigner) offer a visually direct way for page layout and objects synchronization setup, which completely frees authors from HTML and JavaScript programming. For authors who want more advanced control over the layout and synchronization methods, they can customize the HTML files and use their own JavaScript functions to collaborate the components.

In this chapter, we’ll give an overview of the WebSmart system and document structure. In next chapter, we focus on synchronization techniques. More detailed information on the system architecture can be found in [24].
2.1 Multimedia Objects and Documents

WebSmart is designed using object-oriented methodology. There is a natural association of object-oriented concepts and multimedia data types. WebSmart objects directly reflect real world objects or files. The WebSmart documents are the container of multimedia data. The documents are a collection of HTML files that link the distributed multimedia together using Uniform Resource Locators (URL), describe the temporal relation using JavaScript, and manage the spatial layout using HTML Frames. There is a one-to-one correspondence between the objects laid out on the WebSmart Editor and HTML Frames displayed in a Netscape browser. Each object is held in a separate Frame and referred to by a unique name.

There are three types of objects:

- **TOC (Table of Contents):**
  
The anchors in the TOC frame link to different sections, thus control the content of the presentation.

- **Media** (video/audio/closed-caption): continuous media content, presented by WebSmart Applet in the media frame

- **HTML:** all the other objects that are driven by the Media object

Of the three, a Media object, is the only required object. A TOC object is necessary if the authors want to divide the presentation into sections and subsections. An HTML object usually contains multiple HTML pages/files that can be flipped by the Media
object. The flipping HTML pages are called *slides*. Anything can be in these slides: text, images, as well as Java applets and JavaScript. While the slides are flipped by the media object, the JavaScript contained in them can also communicate to the WebSmart Applet and issue controls back to the media object, such as pause or continue a media playback, or to load a specific media clip according to the user’s action, etc. With the LiveConnect between WebSmart Applet and JavaScript, this two-way interaction can be easily arranged by the document designer.

The architecture of WebSmart document is closely related to the architecture of WebSmart objects. The top level HTML file named by the author stores the spatial information of the all objects laid out in the WebSmart Editor. It is generated by scanning through the objects’s position and size attributes. A syllabus.html file can be generated by the TOC object to embed hyperlinks to the sections. Each section/subsection object in the TOC object will generate a section file, e.g. section2.2.html, which contains the continuous media (video/audio/closed-caption files) of this section/subsection, and the synchronization information of the surrounding objects associated with the continuous media. A section*.html page will be loaded into the media frame when users click its title in the TOC frame, and its content will be presented by the WebSmart Applet.
2.2 Physical Components

The WebSmart system has two subsystems: the authoring system and the presentation system. The presentation system consists of a Java Applet: *WebSmart Applet*. The authoring system consists of a visual editor: *WebSmart Editor*, and two supplemental tools: *CaptionMaker* and *QuizDesigner*. Figure 2.1 illustrates the authoring and presentation stages of WebSmart.
Figure 2.1 Physical Model of WebSmart

* Video/Audio files can be prepared using any capturing device.

** HTML slides are the flipping HTML pages. They can be prepared using any HTML editor.
2.3 Authoring System

From Figure 2.1 we can see that the duties of the supplemental tools are to prepare media contents, i.e., question slides and closed caption files. The job of the Editor is to organize various media types into synchronized documents. Table 2.1 summarizes the authoring tools:

<table>
<thead>
<tr>
<th>Authoring Tool</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptionMaker</td>
<td>Synchronize closed caption texts with video/audio and encode the texts into caption files (.cap files).</td>
</tr>
<tr>
<td>QuizDesigner</td>
<td>Prepare question slides to be used in quiz sections</td>
</tr>
<tr>
<td>Editor</td>
<td>The ultimate tool that puts together every thing:</td>
</tr>
<tr>
<td></td>
<td>1. Set up sections, and distribute the continuous media (video/audio/closed caption) into sections.</td>
</tr>
<tr>
<td></td>
<td>2. Layout the objects in a WYSIWYG manner</td>
</tr>
<tr>
<td></td>
<td>3. Schedule the flipping HTML slides (including question slides)</td>
</tr>
<tr>
<td></td>
<td>4. Save the documentation status into an .ws file for maintenance purpose.</td>
</tr>
<tr>
<td></td>
<td>5. Export a set of HTML files for presentation in Web browsers.</td>
</tr>
</tbody>
</table>

2.3.1 Editor

An interactive multimedia web document is usually a collection of many HTML files. Multimedia objects are embedded in these files by Uniform Resource Locators (URLs). These HTML files are to be shown at different frames in a synchronized manner. Without an authoring environment to efficiently establish or reconstruct the
links between the files, it is a tedious procedure to put together everything and maintain the document.

The *WebSmart Editor* provides such an authoring environment. Using the WebSmart Editor, an author can easily add or delete objects, position the objects, establish links between the objects and specific points of media clips, and distribute media clips into sections/subsections. The work can be saved into an object file (a *.ws file) at any point, and be re-opened for modification at future time. The single .ws file that stores every piece of information gives the author the feeling of working on an integrated document instead of a set of distributed files. This makes the maintenance job much easier. Like HTML files which are platform neutral, the .ws file is also portable. It can be opened and modified in a WebSmart Editor at any platform no matter where it was originally authored on.

When the author wants to present the document in a Web browser, the “Export” function can be used to automatically generate all the HTML files.

### 2.3.2 CaptionMaker

*Closed Caption* is classified as a continuous media type in the WebSmart system. Like captions under pictures, captions of video/audio are texts that display the contents of the video/audio. In contrast to Open Captions, which have been integrated with the video and cannot be turned off, Closed Captions are signals that arrive in a separate stream from video/audio. They require a special decoder to make them visible, and can be turned on or off by the user.
Closed caption technology was first developed to enable deaf and hard of hearing people to watch television. But closed captions are not just for deaf and hard of hearing people. Research has shown that closed captions on TV programs can improve people's reading scores, and help people understand/learn foreign languages. On August 7th, 1997, the Federal Communications Commission of the United States (FCC) unanimously approved a new law which will mandate captioning on virtually all television programming in the United States [15]. The ruling took effect from January 1st, 1998.

Although captioning on TV has been enforced by law in the United States, few companies are working on Internet and digital (MPEG) movie captioning. SMPTE (Society of Motion Picture/Television Engineers) [16] has formed a committee to handle this issue. But so far, there is currently no standard for captioning in MPEG files, and no place in the file designated to hold the captions.

WebSmart defines its own caption format and implements original algorithms for closed captioning. WebSmart CaptionMaker is a visual tool that can synchronize caption texts with video/audio, and encode the texts into .cap files. The .cap files can then be attached to their video/audio files in the WebSmart Editor for adding closed captions into multimedia documents. Encoded captions are dispatched through a separate stream to the client. If there are captions in the document, the WebSmart Applet will decode and display the texts in a way synchronized with the video/audio.
Unlike the captions on TV which block part of the picture frame, WebSmart captions are played back in a separate text field that will not obstruct the viewers.

2.3.3 QuizDesigner

The interaction between WebSmart objects can be reciprocal: not only does the continuous media drive the surrounding objects, the surrounding objects can also influence the display of the continuous media. WebSmart QuizDesigner provides an engine for designing a conventional application of this two-way interaction — Quiz. QuizDesigner helps authors prepare input or multiple-choice questions associated with video/audio clips, and embed the questions into HTML slides. The question slides can then be attached to a quiz section in the WebSmart Editor as flipping HTML pages. At the presentation, while these question pages are being flipped by the media object, they also issue commands back to the media object, such as to pause the media playback until the client answers the question, and to display different clips depending on the score.

2.4 Presentation System

The WebSmart Applet is the presenter of synchronized multimedia documents. It is embedded into the document’s section files by the WebSmart Editor to handle continuous media playback and objects synchronization at the client site. It provides the following services:

- Playback streaming video/audio
Built upon Java Media Framework (JMF), the WebSmart Applet can present all the media formats that JMF supports, such as MPEG-1, MPEG-2, QuickTime, AVI, WAV and AU. And JMF’s HTTP streaming technology allows video playback before the whole file is downloaded.

- Display closed captions
  If there are closed captions attached to the video/audio clips, the applet will decode the captions and display the texts in a way synchronized with the video/audio. The captions will be displayed in a text field that won’t block the picture frame.

- Drive surrounding objects
  If there are HTML pages synchronized with the video/audio clips, the applet will flip these pages during the playback of the clips.

- Driven by surrounding objects
  The HTML slides can also issue control back to the applet. For example, the question slides generated by the WebSmart QuizDesigner contain commands to pause and continue the applet playback.

- Driven by viewers
  The applet provides a VCR style control panel to let viewers play, pause, stop, fast forward, and rewind the continuous media playback. Since other objects are driven by the applet, by controlling the applet, viewers control the presentation of the whole document.
• Switch between alternatives

If there are audio-only and image files provided as alternatives to video files, the applet will display an [Audio Only] button which toggles with a [Video/Audio] button to allow viewers switch between the alternatives depending on their bandwidth and network situations.

2.5 Summary

WebSmart provides an environment for users of different levels to build multimedia web documents. The system has the following characteristics:

• Easy to use environment

Despite the dynamic content of WebSmart documents, the authoring tools provide an easy-to-use interface, and a platform which adapts to different levels of document designers. For users with no programming interest, WebSmart does timing information setup by point-and-click, and page layout in a WYSIWYG (What you see is what you get) fashion. Users with HTML knowledge can customize the page layout without affecting the object synchronization. Users with JavaScript experience can impose advanced control over synchronization by writing JavaScript functions. The WebSmart Applet provides a set of simple interface for programmers to specify the control over video playback and JavaScript executions.
• **Extensibility**

The object-oriented nature of the WebSmart system and its underlying Java and JMF technologies ensures the extensibility of the system. For example, WebSmart can be extended to author and present SMIL or some other standard language by adding some parser/translator modules. Live capture and conference could also be integrated into WebSmart’s synchronization models after JMF releases the APIs. Another example is that JMF can be extended to support new media formats by simply defining a new source class and implementing a standard set of interface methods. As a result, incorporating a streaming video server to WebSmart is convenient.

• **Portability**

The 100% Java implementation guarantees the portability of WebSmart softwares. The authoring tools and presentation engine can run on a variety of platforms. HTML and JavaScript secure the portability of the documents. The documents can be viewed in Netscape browsers from a variety of platforms.
Chapter 3  
Synchronization Models

The task of multimedia synchronization involves two aspects:

- **Temporal synchronization**: creates a multimedia presentation by arranging the multimedia objects, such as video/audio, images and texts, according to a time-ordered relationship

- **Spatial management**: links various multimedia objects into a single entity, as for example a page including text and graphics

For spatial management, WebSmart adopts a *channel approach*. During the presentation stage, the presentation screen is partitioned into channels, each channel fits in an area on the screen. Media contents for different channels are presented in parallel, while the contents within the same channel are presented in sequence. At the authoring stage, each channel corresponds to a WebSmart object in the WebSmart Editor, and the contents of the channel, i.e., video/audio, closed caption, and flipping HTML pages, are the attributes of the object. At the current stage of implementation, HTML *Frames* are employed to present channels. The advantage of using frames is their popularity — almost every Web browser supports frames. The disadvantage is that the layout is static. The size and position of each channel are fixed at the authoring stage, which means channels can not be added, deleted or resized dynamically at presentation time.
The temporal synchronization models of WebSmart are simple yet flexible. The synchronization has two levels: the coarse-grained slides and the finer-grained closed captioning. For the coarse-grained slides, WebSmart uses a model called the **PED model** (point-based event-driven model). In this model, the synchronization is centered around a video/audio object — the *media object* in the WebSmart Editor. The time of the media object is used as the common clock for scheduling synchronization events of the surrounding HTML objects. Which means each slide of the HTML objects is linked to a time instant (*point*) in the video/audio stream. The synchronization presentation is controlled through an event-reporting mechanism. As for the finer-grained closed captioning, WebSmart uses a *continuous model*, in which the pace of the caption texts and the video/audio progress is monitored to almost “lip” synchronization.

As a Web-oriented system, WebSmart takes up HTML’s hyperlink feature to give users the choice to jump to a certain section of the presentation. This “jump” action adds one more degree of flexibility into the synchronization models.

In this chapter we’ll define the synchronization models and illustrate their scenarios. We’ll also walk through a document to see how the scenarios are arranged in the WebSmart files using HTML and JavaScript.
3.1 Slide Synchronization

Slides use coarse-grained synchronization. According to Drapeau's paper [17], coarse grained synchronization can be categorized into two models:

- Rising edge model

  Rising edge model only aligns the starting points of different media segments, but cares nothing on whether those segments will maintain synchronized as performance progresses. This is the model used by the MAEstro system[17].

- Constraint-based model

  Constraint-based model allows authors to specify synchronization relationships within media segments. Each object can be described in terms of other temporally related objects. This method imposed a finer degree of synchronization than the “Rising edge” model, yet it also has a fatal weakness: the incapability to recover the time order when a certain intermediate segment crashes before its dependent segments start. Most of the current multimedia systems adopt constraint based model.

Constraint-based model can be further divided into two categories: interval-based and point-based [14]. In interval-based models, each object is associated with a non-zero duration of time, while in point-based models, relations are based on time instants.
Using intervals, authors can express things like: "play video clip 1 at position x, show image 1 at position y and keep it on screen for 5 seconds... At the end of video clips 1 show image 2...". Interval-based relations are dynamic but quite complicated to arrange. The complexity lies in the fact that two temporal intervals can have 13 mutually exclusive relations [23], authors and the authoring system need careful calculations and validations to specify interval relations.

On the other hand, the point-based system is simple and intuitive. Point-based models can express things like "5 seconds into video clip 1, show image 1 at position y; 10 seconds into video clip 1, show image 2 at position y...". Few systems are pure interval-based or point-based, most use the combination of the two models. For example, SMIL uses both duration and begin/end time instants to specify the temporal relations among the media objects. The SRG model used in HPAS system [14] also adds end-points to the interval-based approach.

3.1.1 PED Model

WebSmart adopts the point-based approach to make the flipping of a slide straightforward. In the WebSmart system, the flipping HTML of a slide is scheduled by the time of the media objects. As a result, to schedule a flip in the WebSmart Editor, authors only need to enter a time with respect to a media clip for that slide; or even simpler, just click on that slide when the media clip is being displayed in the Editor's
player. (The details on synchronization setup will be discussed in Chapter 4: Synchronization Setup at Authoring Stage). At presentation time, the slide will be displayed on the screen at its scheduled media time, and stays there till the next slide of the same channel is displayed to replace it.

WebSmart combines this intuitive point-based model with the event reporting mechanism of JMF. This combined model, which we name PED (point-based event-driven), adds the following refinements to the point-based schema:

- **Incorporating user interactions into synchronization**
  All the user control actions, such as play, pause, stop, rewind, skip and fast-forward, are directly on the media object. By sending out corresponding event messages, such as StartEvent, StopEvent, MediaTimeSetEvent, and RateChangeEvent etc., the media object drives the slides along with the user interactions.

- **Sequentially playback media clips**
  The WebSmart system associates media clips into sections. Each section can have one or more media clips. If a section has more than one media clip, these clips are to be played in sequence. When a media playback reaches its end, an EndOfMediaEvent will be posted. This enables the system to start a new sequence of presentation by launching the next media clip and the slides linked on it, or to end the presentation if there is no next clip.
• Interval semantics

EndofMediaEvent also introduces some interval semantics into the point-based model. Authors can arrange interval scenarios like "at the start of video clip 1... at the end of video clip 2...."

• Recover synchronization from error conditions

The media object can also post error events, such as data starved, resource unavailable, and internal error, etc. If an error event message is received, the system can re-adjust the synchronization accordingly. As a result, the system is capable of detecting and recovering from network jitters and segment crashes.

3.1.2 Synchronization Scenario

Figure 3.1 demonstrates the synchronization scenario of flipping HTML. In the figure,

• vertical arrows (↓) shows that each flipping HTML page (slide 1 — slide 5) is associated with a time instant within a certain video/audio clip. For example, slide 2 is linked to the 20th second in clip 1, and slide 5 is linked to the 50th second in clip 2.

• horizontal arrows show the playback paths. Clip 1 and clip 2 are played sequentially. Play, Fastforward (→) can cross the boundary of the 2 clips, Rewind (←) and Skip (...) are restricted within one clip.
Figure 3.1 Flipping HTML Scenario

- dark dots on at the intersection of a playback path and a slide schedule means the slide will be activated (i.e. displayed in its channel) at that point. In WebSmart system, after a slide is displayed, it will stay on screen till the next slide in the same channel is activated to replace it.

WebSmart allows unlimited number of channels on the presentation screen. As a result, there could be more than one slide linked to the same point of a clip. When that point is reached, all the slides linked to it will be displayed concurrently at different
channels on the screen. (See Figure 5.1 on page 69 for an example of two concurrent slide channels on one page). For simplicity, Figure 3.1 illustrates the situation of only one slide linked to one point of a clip, and assumes all the slides are to be displayed in the same channel. The scenario presented in the figure can be summarized as follows:

- At the Play/FF path, a slide will be activated when the path reaches its reserved point in the video/audio stream. Which means when play or fastforward through the clips, a slide will be displayed at its scheduled time (with respect to the clip), and stays till the next slide of the same channel comes up.

- The Rewind path shows that no slides will be activated during the rewind process (the backward arrow). The last slide before the rewind (in this case, slide 3) will stay on screen till slide 2 is activated on the resumed play path.

- The Skip path shows that when a portion of the clip is skipped (denoted by dashed line on the path), the slides associated with that portion will also be skipped. When the play is resumed, the slide just before the resume point (slide 3) will be displayed till the resumed play path activates slide 4.

### 3.2 Specification of WebSmart Closed Caption

Closed caption is treated as a continuous media type in WebSmart. It is displayed within the WebSmart Applet just like the video and audio. Figure 5.1 on page 69 shows closed captions being displayed in the applet's caption field — a textfield just below the video. Since no standard has been set up for closed captioning on digital
video, WebSmart defines its own format for closed captions. In this sense, closed caption is a new media type introduced in the WebSmart system.

3.2.1 Terms and Parameters

Digital video, such as MPEG, is in the unit of Frame. The motion is created by refreshing the frames at a constant frame rate. For example, 30 frames per second is the rate for full motion. Similarly, WebSmart closed caption is in the unit of Line. Caption texts follow the audio by refreshing the lines at a constant Caption Pace. For example, 3 seconds is the default Caption Pace in the WebSmart system. The preferred size of video picture frame is determined by the video's resolution. Correspondingly, closed caption also has a Maximum Column parameter to restrict the length of a caption line. The following list gives some definitions of WebSmart closed caption:

- **Caption Line**: The smallest unit of closed caption. Caption texts are encoded and displayed line by line.

- **Maximum Column**: defines the maximum number of characters allowed in a caption line.

- **Caption pace**: is the time interval (in seconds) between two adjacent caption lines.

Maximum column ensures that each caption line can be fitted into the caption field, whose width is restricted by the size of presentation Applet. WebSmart system uses the resolution of the video to determine the size of the Applet. Common resolutions of
digital video include quarter screen (160 x 120 pixels), half screen (320 x 240 pixels), and full screen (640 x 480 pixels). During the authoring process, the Editor can launch a Player to playback a video/audio file (see Figure 4.1 on page 57). The player wraps the video in its preferred size, or a default size (320 x 240) when only audio is provided, and sends the size information back to the Editor for calculating the size of the Applet. WebSmart CaptionMaker (Figure 4.4 on page 63) also embeds a Player, and uses its size to set a Maximum Column for caption lines. WebSmart uses 12 point Times fonts to display caption texts, and calculates Maximum Column by the following equation:

\[
\text{Maximum Column} = \left(\frac{\text{video}_\text{width}}{6.25}\right) - 3
\]  

(EQ 1)

By this equation, a 320 x 240 video corresponds to a Maximum Column of 48, which means each caption line can have no more than 48 characters. This Maximum Column calculation has been tested on Solaris, IRIX and Windows95/NT platforms. The caption lines set to the length of Maximum Column have no problem fitting into the width of caption field on these platforms.

Unlike Maximum Column which is determined by the video resolution, Caption Pace is independent of video frame rates, but depends on the speaking speeds. And unlike Maximum Column which is automatically set by the CaptionMaker, Caption Pace could be manually adjusted by the user. In WebSmart system, Caption Pace is an integer value, which indicates the precision of caption synchronization with video/audio is in the order of 1 second. Although looser than the “lip sync” requirement
between video and audio, considering the fact that caption texts are displayed line by line not word by word or character by character, 1 second can be considered accurate enough in matching caption lines with speeches.

Caption Pace can also be interpreted as the time unit to divide a video clip into segments, each segment corresponds to one caption line. If a certain segment has no speech (no words), a blank line should be inserted to match that segment to ensure the one-on-one correspondence between video/audio segments and caption lines.

3.2.2 Encoding and Decoding

WebSmart uses simple yet effective algorithms to encode and decode closed caption. Since the caption data size is very small, compression and decompression are not necessary. The purpose of encoding is to conform caption texts to a format that is ready to be synchronized with video/audio. WebSmart closed caption encoding involves the following things:

1. Add a header line in the caption file

WebSmart caption files (.cap files) are plain text files. The first line is a header that records the following information successively: total rows, maximum column, and caption pace. For example, a head line r:20c:48p:3 indicates this caption file has 20 lines of caption texts, the maximum column is 48 characters, and the caption pace is 3 seconds.
2. Fill each caption line to maximum column

The lines in a .cap file (including the header line) are padded to identical length, which equals to the value of Maximum Column. White spaces are appended to short lines to fill them to the maximum length.

Since each caption line is associated to a unit of time (i.e a Caption Pace), by encoding the lines to identical length, each byte in the caption stream is aligned with a time interval within the video/audio stream. In other words, the temporal alignments between a caption stream and a video/audio stream is coded into each byte of caption texts. The decoder at the receiver site can easily calculate from the current media time the proper caption line to display.

The simple format WebSmart’s closed caption makes it easy to build an authoring tool for captioning. It also makes the decoding and synchronization efficient at the presentation time. In the following chapters, you’ll see such a captioning tool: WebSmart CaptionMaker, and the simple algorithms used by the WebSmart Applet for presenting closed captions.

3.3 Closed Caption Synchronization

3.3.1 Continuous Model

WebSmart applies a continuous model to closed caption synchronization. Continuous synchronization model begins multiple media segments at the same time and maintains continuous performance of the segments so they do not drift apart in real-
time. This model requires the finest tuning on those segments. It's usually applied to
the systems that must synchronize audio to the lip motion of the video, such as MPEG
and QuickTime. Closed captioning also adopts this model to continuously monitor the
pace of caption stream with video/audio.

3.3.2 Synchronization Scenario

Figure 3.2 illustrates the scenario of closed caption synchronization. Each of the
vertical dashed lines aligned on each caption pace denotes a caption line. Compare the
closed caption scenario with the slides showed in Figure 3.1 we see there are some similarities between these two synchronization models:

- Like a slide, a caption line will be activated when the play path reaches its point in the video/audio stream.

- The activated caption line will stay in the caption field till the next caption line (could be a blank line) is activated to replace it.

- No caption line will be activated along the rewind path (backward arrow).

- When a portion of the clip is skipped, the caption lines for that portion are also skipped.

The differences between these two scenarios include:

- Slides are discrete, a slide can be placed at any point within the media stream.

  Closed caption is continuous, caption lines are set up at a constant pace along the media stream. Every caption pace has a caption line associated with it. Even if there’s no words during that pace, a blank line is inserted to maintain the temporal alignment of caption byte stream with the video/audio stream.

- Slides can be activated at the FF path.

  Closed caption will only be displayed when the playback rate is normal. So no caption line will be activated along the fastforward path (no audio will be played at FF rate either).
• In the Flipping HTML scenario, the slide just before the rewind action will stay on the screen till the resumed play path activates the next slide.

In the closed caption scenario, when play is resumed after a rewind action, the caption line just before the resume point will be displayed, till the next line is activated to replace it. The reason for this difference is that continuous model actively enforces synchronization alignments, while PED model is more relaxed on alignment calculations. Detail algorithms will be discussed in Chapter 5: Synchronized presentation.

3.4 Hyperlinks

Hyperlink has been one of the primary forces driving the success of the Web. According to the HTML specification [7], a hyperlink is a connection from one Web resource to another. A link has two ends — called anchors — and a direction. The link starts at the “source” anchor and points to the “destination” anchor, which may be any Web resource (e.g., an image, a video clip, a sound bite, a program, an HTML document, an element within an HTML document, etc.).

The WebSmart hyperlinking is intuitive: the whole presentation can be divided into sections and subsections. A source anchor of a hyperlink is a section title, while the destination anchor is the HTML file that describes the temporal relationship of various media types for this section/subsection. By clicking on the source anchors, users can activate the destination anchor, thus jump to the presentation of this section.
3.5 Document Architecture

We have presented the synchronization model of WebSmart, this section will describe how scenarios are expressed in WebSmart documents.

3.5.1 Design Issues

There are two approaches to describe synchronization information: using declarative language or via scripts. The advantage of declarative language is that it is easy to understand and maintain. However, the current Web language HTML does not have declarative formats for describing synchronization information. At the time the WebSmart project began, there was no standard proposed for a multimedia language on Web. The first draft of SMIL was published in November 1997 when the implementation of WebSmart system was close to the end. At the design phase of WebSmart, we studied two possibilities:

1. Design a declarative language for multimedia synchronization; build presentation player and authoring tools for this language.

2. Use the widely accepted HTML and JavaScript as authoring languages to avoid overlapping with the current Web technology; depend on the LiveConnect between JavaScript and the presentation applet to realize synchronization.

We decided against the first scheme for the following considerations:

- People are reluctant to accept non-standard languages
• A declarative language tends to confine the functionality of the presentation player. If authors want to customize a document, e.g., to introduce some new configurations of synchronization, they may have to do some serious coding for the player.

For the above reasons, we decided that HTML and JavaScript are the best choices for our Web authoring system. The above are not the only reasons that we made this decision. Flexibility is a significant consideration. Using simple JavaScript functions, we put the webpage control outside the player. This made customizing much easier. If authors want to realize some new ideas on synchronizing the components on a webpage, they only need to provide their own JavaScript functions.

After we designed the document format, we started to implement a presentation engine to test the idea. By August 1997, we had finished the implementation of the WebSmart applet, and built a 20-minute interactive Web course on it. This proved the feasibility of the document architecture design. When it came to the authoring system design, a new problem arose. The problem lies in the distributed nature of a WebSmart document:

• The spatial layout is presented using Frames in the top level HTML file (for an example, see appl.html on page 93).

• The hyperlinks of Table of Contents is embedded in syllabus.html (see an example on page 94).
- The temporal relationship is described in the section and subsection files (for examples, see section2.html on page 95 and section2.2.html page 97).

Exporting a distributed set of files is not significant because we designed a one-to-one correspondence between the files and the objects in the editor. The problem is how to maintain these files. It is inefficient for the editor to scan through all the files to recollect information back for modification. An intermediate file must be created to serve this purpose. This intermediate file must store the states of all the objects in the editor, thus can be used to quickly reconstruct the objects. Again, there are two possibilities for the format of intermediate files:

1. Design a declarative language to describe the objects. Add a translator module and a parser module to the editor. When saving the states of objects into a intermediate file, the translator can translate the object fields into the declarative language and write them down to the file. When open the intermediate file, the parser can interpret the language and reconstruct the states of the objects.

2. Use the object serialization feature of Java 1.1, directly write the complete images of the objects (including the objects they refer to) into the intermediate file, and recreate the images from the object file.

The advantage of the first approach is, once authors understand the language, they can use any text editor to directly edit the intermediate file, and use the light weight parser to generate the set of HTML files. This gives the authors the freedom to do authoring on a platform without Windows/Xwindows. However this approach
demands much more work than the second approach. It not only requires designing a language, but also requires evolving the language with the editor architecture, and modifying the translator and parser modules accordingly. This could be a major distraction of the editor implementation and testing, especially at the early design stage. On the other hand, the second approach can free us from the language considerations and focus on the editor implementation. The intermediate file generated by Java object serialization can automatically reflect any changes made on the editor object properties. So we chose the second approach. The intermediate file is in the format of Java object file, called the .ws file. Users can save their work into a .ws file at any time, and reopen it for modification in the future. This .ws file only serves as a persistent storage of the author’s work. It can only be interpreted and modified in the WebSmart Editor. The final document that will be sent to the viewers is the set of HTML files exported from the editor.

The first approach can be used to extend the editor to support standard languages such as SMIL. We’ll discuss this issue in the Future Work.

3.5.2 Document Walk-through

Appendix A gives some sample files from a Web course application developed by WebSmart system. Figure 5.1 on page 69 is a screen shot of the course being presented in a Netscape browser, and Figure 4.1 on page 57 shows the application being authored in the Editor.
Here we'll walk through some files to see how the synchronization scenarios are documented. The basic meaning of HTML and JavaScript syntax is beyond the scope of this thesis. The walk-through will be focused on document functionality and LiveConnect interfaces. Readers can refer to the HTML specification [7] and JavaScript Guide [8] for syntax descriptions.

1. Spatial Layout: app1.html

The spatial information is contained in the top level of HTML file: app1.html (on page 93). This file describes the position and size information of the media objects using HTML Frames and Frame sets. Each frame corresponds to an object laid out in the Editor (compare the object layout in Figure 4.1 on page 57 and frame layout in Figure 5.1 on page 69). The name of the frame is the name of the corresponding object; the source URL of the frame, which links to the HTML file that will be displayed in this frame, are the HTML file generated by/for that object.

The height and width of a frame are expressed as percentages of the browser window. The frames are laid out either by rows or by columns. The position of a frame is related to the order it appears in the file. For more detail of the frame layout, refer W3C's HTML specification [7].

2. Hyperlinks: syllabus.html

Syllabus.html will be displayed in the TOC Frame, which corresponds to the TOC object in the Editor. Clicking on a section/subsection title (source anchor of a hyperlink) in TOC frame will load the corresponding section/subsection file
(destination anchor) into the media frame (target). Using these hyperlinks, viewers can directly jump to a certain section/subsection of a presentation.

TOC object is optional in the final document. If an author doesn’t want to display a Table of Contents for a document, he/she can put all the video/audio clips into one section, and hide the TOC object by setting its size to zero in the Editor before exporting HTML files.

3. Temporal Relationship: section files

The expression of the temporal information is the most important part of a WebSmart document. Temporal information of each section is written into the HTML file for that section (e.g. section2.html on page 95). Sections can be further divided into subsections. A subsection contains a subset of the media contents, thus a subset of synchronization events of its parent section. For example, Section2.2.html on page 97 shows the 2nd subsection of section2. In the following paragraphs, we’ll go through section2.html to see how the temporal relationship is specified.

The presentation engine of the section is the applet embedded in the applet tag:

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

Here, the MAYSCRIPT attribute enables the LiveConnect communication between the applet and the JavaScripts in this file. The media contents to be presented within the applet are given as parameters of the applet:
<PARAM name="VIDEO-FILE"
value="movies/sec6.mpg;movies/sec7.mpg">
<PARAM name="AUDIO-FILE"
value="movies/sec6.mpa;movies/sec7.mpa">
<PARAM name="IMAGE-FILE"
value="movies/dxu.gif;movies/llan.gif">
<PARAM name="CAPTION-FILE"
value="movies/sec6.cap;movies/sec7.cap">

Among the parameters, the VIDEO-FILE parameter is required. All the other
parameters are optional. The value of the VIDEO-FILE parameter is a media clip or
a list of clips whose time will be used as the common clock for the synchronization.
These clips could be video files or audio-only files. If there are more than one clip, the
clips are separated by semicolons and will be played back sequentially in the applet.
Their corresponding closed caption files are listed as the value of CAPTION-FILE
parameter, they are to be played under the picture frame of the video/audio. The audio
files and image files listed in the AUDIO-FILE and IMAGE-FILE parameters could
be used as alternatives of the video files listed in the VIDEO-FILE parameter for the
purpose of reducing bandwidth. The files in different parameters are aligned by the
semicolons. It is not necessary that each file in the VIDEO-FILE list has
corresponding closed-caption, audio, or image files. An arrangement like the
following is acceptable:

VIDEO-FILE: videol.mpg;audio2.mpa;video3.mpg
AUDIO-FILE: audiol.mpa;;
IMAGE-FILE: imagel.gif;image2.gif;
CAPTION-FILE: caption1.cap;;caption3.cap

From the alignments, we can see that the section starts with a video clip:
videol.mpg, and its closed caption caption1.cap. There is an audio clip:
audio1.mpa, and an image file: image1.gif aligned with video1.mpg, so they can be used as alternatives to video1.mpg at the presentation (an [Audio Only] button will be added on the applet during the playback of video1.mpg to allow the viewers switch to the image + audio option). After the video1.mpg is finished, audio2.mpa will be displayed with image2.gif. There’s no caption file aligned with audio2.mpa, so no caption will be displayed during audio2.mpa. Finally, video3.mpg will be displayed with caption3.cap. Since no alternative audio and image files provided for this video clip, viewers won’t get the alternative switch at the presentation.

In order for the applet to drive the flipping HTML pages with the video/audio, a JavaScript function must be defined for each of the HTML frames to control the content in that frame. In this Web course application, there are two frames that display flipping HTML: the “title” frame that flips section/subsection titles, and the “Notes” frame that flips the notes of this section. So there are two JavaScript functions defined:

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

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

Then a synchronization statement like "At the start at sec6.mpg, show sec_title2_1.html in the title frame; 15 seconds into
sec6.mpg, show slide7_l.html in the Note Frame" can be expressed by adding addEvent() function calls to the WebSmart Applet:

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

    APP.addEvent("changetitle", "sec_title2_l.html", "movies/sec6.mpg", "0");
    APP.addEvent("changeNotes", "slide7_l.html", "movies/sec6.mpg", "15");
    
    APP.endEvent();
}
```

Where loadEvents() is the "OnLoad" function of the section file. Which means loadEvents() is automatically executed when section2.html is loaded into the media frame. The first two statements of loadEvents() are preparation works: get a reference of the applet, and call a clearAllEvents() method of the applet to clear the old entries in the event tables. The final statement calls endEvent() to indicate the events are ready for presentation. Between clearAllEvents() and endEvent() are a sequence of addEvent() functions to fill the event tables with new events for this section. An ‘addEvent("changeNotes", "slide7_l.html", "movies/sec6.mpg", "15")’ inserts an entry into an event table that will make the applet to execute JavaScript function changeNotes(slide7_l.html) at the 15th second of sec6.mpg. The execution of changeNotes(slide7_l.html) will load slide7_l.html into the "Notes" frame. Time is given as a string in the last argument of addEvent() function. The
unit of time is second, and value should be an integer. There is a special time string that has interval semantics: “end”. Combined with “0” — the start point of a media clip, some interval scenarios can be expressed:

```
APP.addEvent("showSlide", "welcome.html",
    "movie.mpg", "0");
APP.addEvent("showSlide", "good-bye.html",
    "movie.mpg", "end");
```

Translated to English, the above script means: “at the start of the movie, show welcome slide; at the end of the movie, show good-bye slide”.

Here we see, the flipping is done by JavaScript functions outside the applet. The applet itself doesn’t have build-in functions for controlling the Web page. Instead, it controls the synchronization by executing functions defined outside. This design makes the presentation engine very flexible, and gives experienced authors the freedom to realize new ideas by simply providing their own JavaScript functions. For example, if authors want to use DynamicHTML Layers to introduce animations into the presentation, they can do it by:

1. Change the layout in the appl.html from the static Frame layout to the more dynamic Layer Layout.

2. Customize the changtitle() and changeNotes() functions in the section files, add animation control in these JavaScript functions.
4. Two-way interaction

The design of WebSmart document also enables two-way interactions to be easily arranged between video/audio and the slides. That is, the slides driven by the applet can contain JavaScript functions that issue commands back to the applet. The applet provides the following methods that could be invoked from JavaScript to pause or continue the video/audio display:

- `appletPause()`: pause the video/audio display
- `appletContinue()`: continue the video/audio display

A common application of two-way interaction is the "quiz". The WebSmart system provides tools to help authors prepare question slides and set up quiz sections without JavaScript and HTML programming. A quiz section contains synchronization scenario like "15 seconds into video-clip 3, show the slide of question 1; 20 seconds into video-clip 3, pause the video till the viewer answers question 1 in the slide;...". Author can also arrange a summary of the quiz, which shows the percentage of correct answer in a summary slide, and load a specific video tape according to the score. The scenario is like "When the final question is answered, show the slide that summarizes the score; if the viewer scores over 50%, display video clip 5 in the media Frame; else display video clip 6".

WebSmart Applet provides the following methods for writing a quiz section:
quizReset(): reset the numbers for total questions and correct answers
one_more_question(): increase the total question number by 1
one_more_correct_answer():
    increase the correct answer number by 1
getQuestionsTaken(): get the number of total questions
getCorrectAnswers(): get the number of correct answers

Appendix A page 99 to page 105 samples some files from a quiz section. Interested reader may use these files to figure out the details of the two-way interaction setup in a typical WebSmart quiz. Authors can also use the applet interface to design other two-way interactions.

In the following two chapters, we introduce the details of the synchronization setup and presentation algorithms.
Chapter 4
Synchronization Setup at Authoring Stage

The major tasks of multimedia authoring system is to layout and synchronize multimedia components (video, audio, closed caption, image, text etc.). The tasks can be classified into three categories:

- **Predefined Documentation:**
  By “predefined”, we mean that the sources of the multimedia components existed in whole before the author combines them into a document using authoring tools.

- **Live Documentation:**
  Here “Live” is only on the documentation site but not on the viewer site (no live broadcast). By “Live” we mean the documentation is taking place at the same time the multimedia components are being captured and synchronized. When the live capturing is finished, the document is done. Viewers can view the document by accessing the finished file(s).

- **Live Broadcast & Documentation:**
  Here “Live” can be on both the documentation site and the viewer site. The multimedia components and their synchronizations are captured from a Live Broadcast and documented into a file. The capturing can be done by the site doing the live broadcasting, or a site receiving the broadcasting.
The current implementation of WebSmart Editor only supports *Predefined documentation*. Authors must have the media components, such as video/audio, closed captions, HTML slides, and question slides, prepared before synchronizing them in the Editor. WebSmart provides a CaptionMaker for preparing closed caption files, and a QuizDesigner for preparing question slides. HTML slides, which can include images and texts, can be prepared by using any of the existed HTML editors, e.g., the editor in a Netscape communicator. As to video/audio, authors can use the capturing devices on their operating systems to record and encode the content into any standard format. After JMF releases capturing semantics, WebSmart can incorporate capturing into the Editor, and implement engines for Live documentation and Live Broadcast. This possibility will be discussed in the chapter 7: Conclusion and Future Work.

The synchronization setup in WebSmart Editor includes the following processes:

- **Schedule slides (including the question slides)**
  
The slide sources (files) are treated as attributes of WebSmart HTML objects. They are scheduled on the time lines of video/audio clips. The schedule of each slide is treated as an *event* in the object containing that slide.

- **Prepare question slides**
  
The difference between question slides and ordinary HTML slides is that question slides contain JavaScript functions to communicate with the applet. A question slide usually needs to reserve a pause event to wait for the viewer
answering the question. The last question slide may arrange a grade summarize and display different video/audio clips according to the final grade. These pause and summarize events can be set up using the \textit{QuizDesigner}.

- Prepare closed caption files

WebSmart defines its own specification of closed caption. \textit{CaptionMaker} is the tool for synchronizing caption texts with video/audio and encode the texts into the format defined by WebSmart.

In this chapter, we will disclose the algorithms for scheduling flipping events, and making closed caption files.

\textbf{4.1 Laying out the Objects}

WebSmart Editor has a WYSIWYG type of graphic user interface (see Figure 4.1). The objects are laid out in two rows or two columns depends on the author’s choice. In row layout, objects are added from left to right; and from the top row to the bottom row (which means when the row at the top is full, new objects will be added to the row at the bottom). In column layout, objects are added from top to bottom, and from the left column to the right column. The size (width and height) of an object is specified in percentages of the presentation screen. The size of the rows and columns are also adjustable. Authors can change the layout at any time, for example, change the size of an object, switch the layout style between row and column, and swap the position of the objects etc.
When a WebSmart Editor starts a new file, two default objects are placed in the editor window: a TOC object (Table of Contents) and a Media object (Video/audio Panel). Authors can add as many HTML objects as they want. Media object is where the WebSmart applet will appear at the presentation time, so it should not be deleted or hidden. TOC objects can be hidden if the author doesn’t want a table of contents for the presentation (i.e., the presentation has only one section). All the HTML objects can be deleted or hidden.

A WebSmart document that can be viewed in a Web browser is a set of distributed HTML files. The design of the Editor makes extensive use of the physical metaphor paradigm to represent the real-world files. Each HTML file in the document architecture is reflected by an object in the Editor’s object architecture, and the writing of this file is done by its corresponding object (see Table 4.1).

The interactions between the real-world media objects are captured through the communications between the corresponding Editor objects. For example, the point-and-click timing of HTML slides are realized through the communication between the Media object and the HTML object, and the events are sorted into sections at the exporting time by the communications among the TOC object and the HTML objects.
## Table 4.1 Objects and Their Exported Files

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS</td>
<td>Top level object that contains the handles of all the objects on the screen</td>
<td>Top level HTML file, named by the author (e.g. appl.html)</td>
<td>Frame layout of the objects</td>
</tr>
<tr>
<td>TOC</td>
<td>Contains an dynamic array of section objects</td>
<td>syllabus.html</td>
<td>Hyperlinks to presentation sections</td>
</tr>
<tr>
<td>Section</td>
<td>Contains media contents and an event table for the flipping HTML scheduling</td>
<td>section*.html</td>
<td>Media contents and events(^1) for a presentation section/subsection</td>
</tr>
<tr>
<td>HTML</td>
<td>Contains the source URLs of the slides and their timing events. Reports the events to section objects at exporting time.</td>
<td>No direct output file, but contributes to the section files</td>
<td></td>
</tr>
<tr>
<td>Media</td>
<td>Center of synchronization. Contains a media player that plays the video/audio content, and reports current media time and clip name to HTML objects on requests.</td>
<td>No direct output file, but contributes to the section files</td>
<td></td>
</tr>
</tbody>
</table>

Note: 1. Event — The schedule of each slide is treated as an *event*

### 4.2 Flipping HTML Scheduling

Each object on the Editor screen corresponds to a frame in the presentation browser.

Each slide that will be shown in this frame should be specified as a source attribute of
Figure 4.1 WebSmart Editor and Player
the object. When entering the source URLs for the slides, authors can also enter in the start times and reference video/audio clips, or leave the time and clip fields empty at this moment. After the sources of the slides are listed, the timing of each slides can be easily scheduled or updated via point-and-click during the playback of a video/audio clip. Figure 4.1 shows the slide sources are listed in the HTML objects *(title and Notes)*.

### 4.2.1 Slide Timing

Setting up flipping HTML synchronization by point-and-click indicates the slides are linked to a clip in real time. On user’s request, the Video/audio panel can pop up a media player to playback video/audio clip(s) (see Figure 4.1). During the playback, authors can select a slide source in an HTML object and click the [Timing] button on that object to link the slide to the current media time of the current clip. Slides in different objects can be scheduled to the same time instance of the same clip. When a slide is linked to a clip, its parent HTML object is also associated with the clip and will be responsible for reporting the event to the section objects that contain the clip.

Figure 4.2 illustrates the algorithm for point-and-click slide timing. A TOC object contains a dynamic array of section objects. An HTML object contains three dynamic arrays that store the source URLs of the slides, start times for the slides, and the reference video/audio clips respectively. Each media object contains a media player. When the user clicks [Play Clip] *(denoted by Action: Play Clip in Figure 4.2)*, the player will obtain a clip file name from user input and playback that clip. When the user clicks
Figure 4.2 Point-and-click Slide Timing

[Play Section] (Action: Play Section), the player will fetch the video/audio list from the current section in the TOC object, and send the pre-fetched player size of each video clip back to the section object for calculating the applet size. When the user clicks [Timing] on an HTML object (Action: Timing), the HTML object will request the current media time and clip name from the player, and fill the information into the
Start-Times and Clip-Names array at the position aligned with the selected slide source. This way, the slide is linked to the current clip at the current media time, and the event is recorded in the parent HTML object.

Since slides are linked to video/audio clips not sections, it is not necessary to set up sections before synchronizing slides. In another word, distributing clips into sections and linking slides to clips are two independent tasks. Only at the exporting time, the slide timing events of all HTML objects will be sorted into the section objects and written to section files.

4.2.2 Exporting the Synchronization Information

When authors select “Export” from the File menu, the synchronization setup in the editor will be output to a set of HTML files. This exporting process involves three level relay:

1. **Top level: exporting the spatial information to the top level HTML file**

   Exporting spatial information is done by scanning through the size and position attributes of the objects laid out on the screen. This task is handled in the top level object: WS, which contains the references of everything in the editor.

   After the WS object finishes exporting spatial information, it communicates with all the HTML objects to collect entries for a *Vid2Obj* hashtable. *Vid2Obj* uses video/audio clip names as keys. An entry of the table is a dynamic array that stores all the
HTML objects associated with the key. When the hashtable is ready, it is passed to the TOC object to be used in the next levels of exporting.

2. **TOC level: exporting hyperlinks to syllabus.html**

Exporting hyperlinks is done by scanning through the title attributes of the sections listed in the TOC object. It also involves communicating with the Media object to obtain the name of Media frame. The name will be used as the target of the hyperlinks (see syllabus.html on page 94).

TOC object then passes the $Vid2Obj$ hashtable to all the section objects in it. The section objects will use this hashtable to sort out the temporal events for each section.

3. **Section level: exporting temporal events to section files**

Exporting temporal events into section files is done by the section objects contained in the TOC object. Each section object has a list of the video/audio clips assigned to this section. From the $Vid2Obj$ hashtable, a section object can query out the HTML objects associated with its clips, and then communicates with the HTML objects to collect the entries for the section’s $EventTable$. A section’s $EventTable$ is a dynamic array that stores the sources and start times of the slides linked to the clips of this section. These timing events are then written down to the file of this section.

4.2.3 **Pause Event Reservation**

Question slides may reserve pause events. Pause events are not written in section files, but in the question slide file itself (e.g., q1.html on page 101). Question slides are
generated by QuizDesigner. When setting up a question, the time and clip name for a pause event can be manually specified (see Figure 4.3). At the export time, the event is directly written into the file for this question.

![Figure 4.3 Reserve a pause event in a question slide](image)

Figure 4.3 Reserve a pause event in a question slide
4.3 Making Closed Caption Files

Each video/audio file can have a closed caption file (.cap file). WebSmart CaptionMaker is the authoring tool for synchronizing closed captions with video/audio and encoding the texts into the formats defined in Chapter 3.

There are two types of captioning: on-line (real-time) captioning and off-line (pre-recorded) captioning. On-line captioning, such as the captions on TV news broadcast, is performed by stenocaptioners, who are court reporters with special training. They use a special keyboard, called a “steno keyboard” or “shorthand machine”, to write what they hear as they hear it. Unlike a traditional “QWERTY” keyboard, a steno keyboard allows more than one key to be pressed at a time. The basic concept behind machine shorthand is phonetic, where combinations of keys represent sounds. The
audio is translated into text and commands and formatted into captions by a computer system, and sent to a caption encoder. Off-line captioning, such as the captions on the movie rental tapes, is usually performed by captioning companies using expensive equipments. Off-line captioning requires editing softwares for creating the caption texts, and encoding softwares and hardwares for placing the captions on tape with some form of timecoding for synchronization.

WebSmart CaptionMaker is designed for off-line closed captioning on digital video/audio. It provides an easy to use interface for editing caption texts, and efficient algorithms for encoding the captions. Figure 4.4 is a screen dump of the graphic user interface. The single line text field under the picture frame is called *caption_field*. It is used for displaying and editing a caption line for the current clip segment. The multiple line text area besides the picture frame is a *file-viewer* that displays the whole caption file with the current caption line highlighted. Captions in the *file-viewer* will be saved into a default file name, which is the same name of the clip file but with a .cap suffix, e.g., video_1.cap for video_1.mpg. Authors can choose *Save as* from the *File* menu to change the caption file name.

### 4.3.1 Editing caption texts

Not everybody can write down speeches as fast as professional captioners. To make caption editing an easier job for ordinary people, CaptionMaker offers two modes for playing back the video/audio: *Step Play* and *Continuous Play*. It also allows editing at *Stop* mode.
• *Step Play* mode: pacing caption lines

In *Step Play* mode, the playback will automatically pause at each caption pace. At each pause, authors can edit the caption line for the current caption pace. *Ctrl_B* (or chose *Step Backward* from *Edit* menu) can be used to replay the current segment if necessary. When the line is ready, *Ctrl_E* (or chose *Enter Caption* from the *Edit* menu) should be used to enter the line into the *file_viewer*. The playback will automatically continue to the next caption pace. Even if there's no dialog in the current pace, a blank line should be entered into the *file_viewer*.

• *Continuous Play* mode: previewing synchronization

*Continuous Play* mode can be used to preview the synchronization of captions with video/audio. During the playback, the caption line associated with the current caption pace will be highlighted in the *file_viewer* and loaded into the *caption_field*. Authors can pause the playback at any point to modify and re-enter the current caption line in the *caption_field* to the *file_viewer*.

• *Stop* mode: proofreading caption file

The whole caption file is presented in the *file_viewer* for convenient proofreading. If authors want to modify a certain line, they don't need to start the playback. They can just drag the position slider along till the problem line is highlighted and loaded into the *caption_field* for editing.
4.3.2 Encoding captions

The simple format of WebSmart closed caption determines the effectiveness of caption encoding. CaptionMaker performs encoding in real-time, which means encoding is achieved at the time of editing. The file viewer displays the encoded caption file, including the header line. At the time a video/audio clip is loaded into the CaptionMaker, the Maximum Column of caption lines is calculated out from the video resolution, or from the default resolution 320 x 240 when only audio is provided (see (EQ 1) on page 34). Maximum Column will be used to encode the caption lines to the maximum length. The value of Maximum Column, with the value of total rows (0 at this moment), and the default caption pace (3) are entered into the header line in the file viewer. If the clip already has a .cap file, CaptionMaker will load that caption file into the file-viewer and retrieve values of Maximum Column, total rows, and caption pace from the header line instead of calculating or using defaults. At every attempt of entering a caption line, the length of the line will be checked. If the line is longer than Maximum Column, CaptionMaker will ask the author to cut the line shorter; else, white spaces will be appended to the end of the line to encode it to maximum length. The encoded line will be entered into the file viewer. If this is a new caption line, the value of total rows will also be updated in the header line.

Authors can also adjust the caption pace. Whenever a caption pace is changed, the headline will be updated with the new value. Since caption pace is a constant value for the whole caption file, all the caption texts must be re-adjusted after a pace change.
The functions in the Edit menu, such as “Copy”, “Cut”, and “Paste” can be used to make the re-adjustment easier.

Although the .cap files generated by CaptionMaker are plain text files, every byte in those files conceals timecoding. If authors use ordinary text editors to modify the .cap files, the synchronization alignment between the caption stream and the video/audio stream could be easily destroyed. Only CaptionMaker should be used to edit caption files.
Chapter 5
Synchronized Presentation

The WebSmart Applet is used to present multimedia documents. A WebSmart document can have arbitrary number of objects that concurrently synchronized with the media object. To control the parallel presentation, the applet devotes a separate thread (called an EventThread) to each of the objects. To ensure the temporally related objects won’t drift apart during the presentation, the applet sets apart a ClockThread to enforce the time alignments of the EventThreads. Closed caption is also monitored by the ClockThread. Figure 5.1 shows a web course document is being presented by the applet within a Netscape browser. This chapter will discuss the algorithms used in the applet for temporal scheduling.

5.1 Flipping HTML Control

Each schedule of a flipping HTML slide is called an event. At the authoring stage, the events in each HTML object are sorted into the EventTables of section objects and exported as event statements (i.e. the addEvent() calls) in section files. Each HTML object also has a JavaScript function defined in the section files to handle the flipping within the object. For example, to control the flipping in Obj_1, a JavaScript function “changeObj_1()” will be defined.
Figure 5.1 A Web course application
At presentation time, the WebSmart applet devotes a separate thread (called an EventThread) for each of the JavaScript functions to ensure the parallel control over the objects and guarantee the timely executions of the specific JavaScript function. In other words, each HTML frame is assigned an EventThread to control its slide scheduling and flipping. The EventThread is named after the JavaScript function, e.g., the EventThread of function “changeObj_1” is also named “changeObj_1”. Figure 5.2 illustrates the correspondence and the control scenario between editor object, browser frame, JavaScript function in a section file, and EventThread in the applet.

Figure 5.2 Presentation Control Scenario

WebSmart allows unlimited number of flipping HTML objects on a page, as a result, there could be an unlimited number of EventThreads in the applet. Each EventThread controls the temporal presentation in a distinct frame, thus maintains its unique event table and has its unique actions. However, they also share many common
behaviors, such as they all need to adjust their clocks with the media time of the media object, to respond to media events posted by the media object, and to react to user interactions. In this sense, they can be managed as an integrity. WebSmart applet uses an EventGroup object to supervise the EventThreads. EventGroup has a thread table that stores the handles of all the EventThreads for the current presentation. The major tasks of EventGroup are event dispatching, clock alignment, and reaction to media events.

5.1.1 Event dispatching

Event dispatching is the first assignment of an EventGroup. It is performed at the load of a section file. A section file usually contains events of many HTML objects, these events must be dispatched to the EventThreads before the presentation can be started. On the load of a section file, its “loadEvents()” function will be invoked. The first requirement (clearAllEvents()) is to clear the event tables of all the EventThreads in the EventGroup. Then applet is then asked to execute the “addEvent()” statements of this section. When the applet executes an addEvent() call such as “addEvent("changeObj_1", "slide_1.html", "video_1.mpg", "5")”, it will first query the EventGroup object for the thread named “changeObj_1”. (If that thread does not exist yet, EventGroup will create it and add it into the thread table). The applet will then inform the “changeObj_1” thread to insert the event into its event table. In English, this event can be described as: “execute JavaScript function changeObj_1(slide_1.html) at 5 seconds into
video_1.mpg”. Finally, the “endEvent()” call prepares the applet to start the presentation and asks the EventGroup to broadcast the name of the current media clip to all its EventThreads.

Figure 5.3 depicts the event sorting and dispatching processes. We can observe the symmetry on the event organization and presentation: at the authoring stage, the events distributed in the parallel objects are documented into a presentation section; when the section is to be presented, the events are distributed into parallel threads and executed in parallel.

**Figure 5.3 Event Sorting and Dispatching**
5.1.2 Clock alignment

To reduce the work load of thread synchronization, the applet uses a *sleep-and-wake* approach to let each EventThread pace at its own clock by sleeping the thread for 1 second. Every iteration the thread wakes up from the sleep, it forwards its clock by 1 second, and checks its event table to see if there’s a slide scheduled at this second of the current clip (the name of the current clip is broadcasted to all the EventThreads by the EventGroup object when the clip is loaded). If there is an event, the thread will execute the JavaScript function to load the slide to the HTML object it is associated with.

This *sleep-and-wake* approach is resource saving. But the threads will drift apart over time. To enforce the time alignments of the EventThreads with the media time, the applet uses a ClockThread outside the EventGroup. ClockThread is also a *sleep-and-wake* thread, but it does not maintain its own clock. Instead, every time it wakes up from a 1 second sleep, it queries the media player for the current media time. At a pace of every 10 seconds of media time, the ClockThread requests the EventGroup to broadcast the current media time to all its EventThreads. When an EventThread receives the media time, it will set its clock to the media time. If the media time is bigger than the time on its own clock, the EventThread will check for the skipped events in between and execute the events immediately. On the other hand, if the media time is smaller than its clock, it is not necessary for the EventThread to do a backward check to replay the old events.
This way, EventThreads adjust their clocks with the media time in a coarse grained manner (every 10 seconds) for the coarse-grained flipping HTML synchronization. While the ClockThread is continuously adjust itself with the media time at every second. This grid is fine enough for closed caption presentation. So the ClockThread is also used to handle the closed caption monitoring.

5.1.3 Reaction to Media events

Flipping HTML presentation needs to react to user interactions as well as media events posted by the media object. User interactions, such as play, stop, fastforward, rewind and skip etc., are also posted as media events, such as StartEvent, StopEvent, RateChangeEvent, MediaTimeSetEvent etc. So reacting to user interactions are actually realized by reacting to media events. Table 5.1 lists the major media events, the causes of the events, and the reaction of the EventGroup.
Table 5.1 Reaction to Media Events

<table>
<thead>
<tr>
<th>Media Event</th>
<th>Causes</th>
<th>EventGroup Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartEvent</td>
<td>User interaction: play</td>
<td>Broadcast the current media time to all the EventThreads, and start/resume the pro­­­­­­­­cesses (clock forwarding, event checking etc.) in all the EventThreads</td>
</tr>
<tr>
<td>EndOfMediaEvent</td>
<td>A clip plays to its end</td>
<td>Suspend the processes in all the EventThreads. If a new clip is loaded, broadcast the clip name to all the EventThreads</td>
</tr>
<tr>
<td>StopEvent</td>
<td>user interaction: stop; EndOfMedia; DataStarved</td>
<td>Suspend the processes in all the EventThreads</td>
</tr>
<tr>
<td>Media-TimeSetEvent</td>
<td>User interaction: rewind (media time is set backward)</td>
<td>Broadcast the media time to all the EventThreads. EventThreads will set back their clocks.</td>
</tr>
<tr>
<td></td>
<td>User interaction: skip (media time is set forward)</td>
<td>Broadcast the media time to all the EventThreads. EventThreads will adjust their clocks and catch up with the skipped events when the play is resumed.</td>
</tr>
<tr>
<td>RateChangeEvent</td>
<td>User interaction: fastforward</td>
<td>Broadcast the rate to all the EventThreads. The EventThreads will process the events at the new rate.</td>
</tr>
<tr>
<td>DataStarvedEvent</td>
<td>Network problems; insufficient bandwidth</td>
<td>Suspend the processes in all the EventThread</td>
</tr>
</tbody>
</table>

5.2 Closed caption monitoring

The simple format of WebSmart closed caption not only determines the effectiveness of encoding, but also determines the effectiveness of decoding and presentation. Captions are played back directly from the network in a streamed manner. The applet fetches caption lines from a data stream. No buffer is reserved to store the used lines, a line will be immediately discarded after it is processed or displayed. The first line
received is the header line, from which the Maximum Column, total rows, and caption pace can be retrieved. The synchronized display of caption lines is monitored by the ClockThread which continuously queries the media time at every second. When the video/audio is being playing back at its normal rate, the playback of the caption lines is straightforward: the ClockThread always pre-fetches a line, when time comes to display that line, the line will be immediately loaded into the caption field, and the next line will be pre-fetched.

User interactions may enforce re-alignments of captions with the video/audio. Whenever the play is resumed from a skip, rewind, or fastforward action, the ClockThread will immediately calculate the current line from the current media time (current line is line number of the caption line that should be displayed at the current media time):

\[
\text{current line} = (\text{int}) \left( \frac{\text{current mediaTime}}{\text{captionPace}} \right) \quad (EQ \ 2)
\]

The current line is then used to calculate the number of bytes that must be skipped from the beginning of the caption data stream:

\[
\text{skipBytes} = (\text{current line} + 1) \times \text{maximum_column} \quad (EQ \ 3)
\]

(skipBytes calculation uses current line +1 to skip the header line)

The ClockThread will then jump to the correct position in the data stream to fetch and display the proper caption line.
Chapter 6
Related Work

Synchronized multimedia on Web is a fast growing area of research. During the past few years, many multimedia languages and file formats have been defined, and many authoring systems and protocols have be developed. Organizations and companies have made numerous attempts to promote their products as standards. Although none of the languages, formats or protocols has been officially accepted as standards or been widely adopted into a variety of products, each of them has its unique benefits. Their co-existence and competition enrich the world and propel the advancement of today’s Web technology. The work discussed in this chapter covers only a small portion of the their contributions.

6.1 Synchronization Languages and Multimedia File Formats

6.1.1 HyTime: a complicated standard

HyTime [22], The Hypermedia/Time-based Structuring Language, is an international standard (ISO/IEC 10744:1992) for encoding the structure of hypermedia documents. It is an extension of Standard Generalized Markup Language (SGML [26]), which encodes the structure of text documents.

SGML is a metalanguage for defining markup languages. A key contribution of SGML is separating the modeling of media content from processing specifications.
Media objects contained in an SGML document are named and described (using attributes and sub-elements) in terms of what they are — from a definition perspective — not in terms of how they are to be displayed or otherwise processed. As a result, markup languages defined by the formalisms of SGML are independent of data processing and presentation platforms thus suitable for electronic information exchange.

HyTime inherits from SGML the ability to define multiple document models, open and integrated documents, and document structure independently of document presentation. HyTime extends the ability to the following aspects of hypermedia document structure: addressing of document objects, relationships between document objects and numeric measurement of document objects. HyTime is an abstract enabling technology: it can describe virtually any kind of connected or temporal information. It covers many different aspects on linking, as well as a full repertoire of features for multimedia purposes, including virtual time, scheduling, synchronization and so on.

HyTime is powerful and flexible, but it is very difficult to read and understand, and proved to be too abstract so that most developers have ignored it. While less robust but very simple language like SMIL is more likely to be accepted.

6.1.2 SMIL: a simple declarative language

SMIL [10], the Synchronized Multimedia Integration Language, was proposed in 1997 by the World Wide Web Consortium (W3C), the same organization for evolving and
standardizing HTML. The intention of SMIL is to provide a simple yet comprehensive and flexible declarative language that will enable Web developers to synchronize text, graphics, audio and video on Web.

SMIL is an HTML-like markup language using easy to understand tags, and can be written with a simple text editor. HTML language features like URLs and hyperlinks are seamlessly integrated into SMIL. SMIL references media elements using URLs, and allows associating links to spatial or temporal subparts of a document. More importantly, the SMIL language provides a simple syntax for authors to schedule the playing of time-based media within their documents. For example, authors can indicate when an object should start and when it should stop, where it should appear on the presentation screen, and whether it should start a new context or just replace the current presentation. Objects can be played in sequence or in parallel. The synchronization among the objects could be “hard”, where all objects are mapped to a common clock, or “soft”, where each parallel object has its own clock that runs independently of other objects. SMIL also provides switches for choosing among alternatives, such as audio tracks of different languages, or media contents of different bitrates.

As a markup language, SMIL does not attempt to define media formats in itself, nor enforce specific synchronization or streaming mechanisms. Any media object elements (actually, media file URLs) can be included in SMIL under the tags of “ref”, “audio”, “video”, “text” and “img” using URLs. How and whether the objects will be
delivered, synchronized and presented will depend on the implementations of servers and presentation engines.

The first draft of SMIL was published in March 1997, and the specification is still in progress. HPAS [25] made the first implementation of a player that can present part of the SMIL features. Although not as robust as HyTime, SMIL has the advantage of simplicity and concreteness. Moreover, the working group of SMIL has included representatives from Microsoft and Netscape as authors. It also invites participants from relevant industries like CD-ROM, interactive television, and Web multimedia. So it is much more likely to be widely accepted and be implemented in products. As a non-proprietary synchronized multimedia integration format, SMIL could free authors from relying on the proprietary ASF format from Microsoft.

6.1.3 ASF: Microsoft’s Active Streaming Format

ASF (Active Streaming Format) is the underlying file format of Microsoft’s proprietary “all-in-one” Web multimedia solution: NetShow [20]. Microsoft has submitted ASF for standards consideration with the International Standards Organization and the Internet Engineering Task Force.

ASF is a format for synchronized streaming multimedia. Unlike SMIL which uses URLs to address the external media objects, ASF includes all the media elements in one file: text, audio, video, metafiles describing the content, etc. It organizes the storage of multimedia data packets into the intended sequence. The sequenced data packets will be inserted one at a time into the data field of network packets at
transmission. The data within ASF packets are prioritized for tuning the performance according to bandwidth. Essentially, ASF format acts as a container for the data, while SMIL is only a description of the object relationship.

The major disadvantage of ASF is that the text portion of the content can not be indexed by search engines thus is hard to find. While in SMIL, the text components are entirely searchable. SMIL also has the advantage of incorporating reusable material. The external media files referenced by a SMIL document could be stored on different servers, thus can be used by many other documents. However, this advantage could also be a disadvantage. The distributed files impose extra complexity on delivery and synchronization presentation. Which approach can dominate the Web is still an open issue.

6.2 Authoring, Delivering and Presentation

6.2.1 Authoring tools and presentation engines

Many commercial authoring products provide the ability to produce and publish synchronized multimedia on Web. The most popular ones include RealNetworks’ RealSystem [11], Microsoft’s NetShow [20], and Macromedia’s Authorware Interactive Studio [4].

All these systems provide some kind of authoring tools. RealSystem uses RealPublisher for RealVideo/RealAudio encoding and publishing, NetShow depends on ASF Editor for creating .asf files, and Authorware Interactive Studio offers
Authorware for setting up hyperlinks and custom interactions among media contents (text, video, audio, sound, graphics), and Director for creating Direct Movies which can import 2D and 3D graphics, text, animation, sound and digital video.

Most of the systems also define some simple languages for describing synchronization. RealPublisher only offers functions for encoding and embedding video/audio content in a HTML page. To express synchronization information (such as start time, endtime, URL etc.), users must prepare a plain text file that conforms the syntax of RealNetworks' Input Events File, and plug this file into HTML files. Director also requires authors to write scripts in Lingo language for adding interactions into director movies.

Currently, all the systems require plug-in installations for viewing their multimedia documents, and most of the authoring tools as well as their presentation plug-ins only work on Windows and/or Macintosh. However, the vendors are making efforts to port their products, at least the presentation engines, to as many platforms as possible, so the documents can reach as many readers as possible. RealNetworks has recently teamed up with SUN Microsystems for porting RealSystem to SUN Solaris platforms, as well as incorporating JMF (Java Media Framework) to support cross-platform playback of RealVideo and RealAudio without having to download plug-ins. Macromedia's Director also adds a "Save as Java..." solution to allow developers output their contents for delivery on Java-enabled browsers without requiring a plug-in. The contents are to be displayed by a Java applet player.
6.2.2 Streaming servers and protocols

There are two approaches to deliver the multimedia content over networks: server or server-less. Server approach is to establish a server specialized in delivering synchronized multimedia content. server-less means the delivering is handled by general purpose servers and protocols, such as Web servers and the HTTP protocol.

Streaming is the first consideration of delivering huge files. Most of the current commercial products furnish streaming servers to allow users to playback multimedia content (such as video/audio) immediately without waiting for the complete file to download. The most popular streaming server is RealNetworks’ Realserver. Realserver not only handles streaming. Combined with the multi-template encoding of RealAudio/RealVideo, Realserver provides bandwidth negotiation that allows viewers to receive the content at their stated bandwidth.

Synchronization maintenance is another important factor in delivering synchronized content. Extremely precise synchronization, required for lip-sync, is only currently possible at the level of the bits, or the files themselves. General purpose protocols like HTTP are not designed for synchronized delivery. For example, MPEG Audio Layer 3 (used in Microsoft's NetShow) suffers significant time drift problems, especially over longer files. To lessen such problems, RealNetworks, Netscape Communications, and Columbia University have developed the Real-Time Streaming Protocol (RTSP) [27], which is nearing standards approval.
RTSP is built on top of existing Internet standard protocols, including *UDP* (User Datagram Protocol) *TCP/IP* (Transport Control Protocol/Internet Protocol), *RTP* (Real-Time Transport Protocol) and *IP Multicast*. Unlike HTTP, RTSP knows when specific files are supposed to arrive to the client and can deliver them according to schedule. It delivers functionalities similar to that of a VCR: Play, fast-forward, pause, stop, and record. RTSP has random access to files for time-based seeking. It is compatible with timestamp formats such as SMPTE [16] timecodes familiar to filmmakers, and is designed to control multicast delivery of data streams. RTSP can deliver data streams like Microsoft’s Active Streaming Format (ASF).
Chapter 7
Conclusion and Future Work

7.1 Remaining Challenges

WebSmart offers an authoring environment, a presentation engine, and a flexible document architecture. What is missing from the framework is a delivering mechanism. At current stage, the documents are sent across networks by ordinary Web servers through HTTP protocol. Although the presentation engine supports HTTP streaming which allows playback while downloading data, the performance is often disturbed by data starvation over poor bandwidths or unstable network situations. Thus augmenting the framework with a streaming server that is specialized in timely delivery and bandwidth negotiation should take the highest priority in the future work.

7.2 Future Work

7.2.1 Streaming server

In the future release of JMF, streaming protocols like RSTP [27] may be incorporated if RSTP becomes a standard. If so, streaming is naturally supported in WebSmart. This is the advantage of the protocol-neutral JMF architecture. If we choose to develop our own streaming server and protocol, we need to add a new data source class for JMF. This class can implement some methods for supporting push or pull data streaming as well as time seeking according to the standard JMF interfaces.
7.2.2 Supporting SMIL

SMIL is a more comprehensive model that covers all the features of WebSmart documents. Supporting the overlapped part could be easy. Just add a translator module and a parser module to the editor, and provide a “Save as SMIL” option in addition to “Save as WS”. The translator can write the object states into the SMIL language, and the parser can read SMIL files and interpret the statements into object images.

There are some features of SMIL that WebSmart currently does not support. For example, WebSmart synchronizes all the objects around one media object, while SMIL allows multiple reference frames. To support features like this needs redesign of the synchronization scenario and significant change in the user interface.

7.2.3 Live Documentation and Live Broadcast

After JMF releases capturing, live semantics can be introduced into the synchronization models, and Live documentation and broadcast features can added into the system. Live documentation has little impact on the synchronization scenario or user interface. The only differences could be a record button on the player, and the content in the player is read from a capturing device or live broadcast protocol like RTP instead of from local files or HTTP protocol. Live Broadcast requires incorporating push technology and IP multicast into the system thus needs more works on the underlying modules. However, the user interface could remain the same.
7.3 Status and Availability

The implementations of WebSmart authoring tools (Editor, CaptionMaker and QuizeDesigner) and the presentation applet are publicly available. On-line documents and links to softwares and demo applications can be found in WebSmart’s home page:

http://rainforest.cs.ubc.ca/mmas/WebSmart/

7.4 Summary

For the past year, we’ve been working on the WebSmart project to develop a new synchronized multimedia authoring system. Leveraging the latest developments in Java and state-of-art Web technologies, WebSmart explores a platform-neutral, content-neutral and protocol-neutral solution for bringing synchronized multimedia to Web. It also defines a specification for closed caption and implements engines to bring this new media format to life. Extensibility, e.g., adding new modules and incorporating new technologies, is ensured by the object oriented design of the whole system. 100% Java implementation guarantees the software portability, while employing HTML and JavaScript as authoring languages secures the document portability.

The authoring tools offers easy-to-use interfaces that totally free the authors from writing HTML and JavaScripts. The Java applet presentation engine relieves viewers from installing plug-ins. WebSmart documents can be easily customized to support new configurations of synchronization by some HTML and JavaScript programming.
The synchronization model supports coarse-grained HTML flipping as well as finer-grained closed captioning. The PED (point-based event-driven) model applied to the HTML flipping combines the intuitiveness of point-based model with the dynamic of JMF's event reporting mechanism. Point-based semantics make the synchronization setup straightforward, while the event reporting mechanism incorporates user interactions and enables sequential presentation and error condition recovery. Compared to the PED model, the continuous model applied to the closed captioning refines the grid of synchronization and actively enforces the alignments between closed captions and video/audio streams to almost lip-sync. The simple format of WebSmart closed caption, which is based on identical length of caption lines, determines the effectiveness of caption encoding, decoding and synchronized presentation.

To control the parallel presentation of arbitrary number of media objects, WebSmart delivers synchronization events to different EventThreads devoted to distinct presentation channels. Each EventThread runs its own clock by sleep-and-wake. A special ClockThread which continuously adjusts itself with the time of the media object is set apart to monitor the clock alignments of the EventThreads and the playback of closed caption.

This system is well suited for developing interactive applications like Web course, on-line commercial, product demonstration and electronic presentation etc. Future enhancements include adding effective delivering mechanisms, supporting standard
synchronization languages, and enriching the synchronization models with live capturing and broadcast semantics.
References

http://www.w3journal.com/6/s2.hoschka.html

[2] Craig Locatis, et al., Authoring Systems,  

[3] Powerpoint, Microsoft  
http://www.microsoft.com/products/prodref/127_newf.htm

[4] AuthorWare, MacroMedia Inc.  
http://www.macromedia.com/software/authorware/

http://java.sun.com/docs/overviews/java/java-overview-1.html


http://www.w3.org/TR/WD-html40/cover.html

[8] JavaScript Guide (including LiveConnect), Netscape  

[9] Dynamic HTML, Netscape  
[10] SMIL, Synchronized Multimedia Integration Language
   \textit{W3C Working Draft} 2-February-98
   http://www.w3.org/TR/WD-smil


[12] VXtreme
   http://www.microsoft.com/netshow/vxtreme/

[13] Jin Yu, Yuanyuan Xiang, Hypermedia Presentation and Authoring System,
   \textit{Sixth International World Wide Web Conference}, 1996
   http://www6.nttlabs.com/HyperNews/get/PAPER91.html

    in the browser/Java environment. Technical Note 1997-027,
    \textit{Digital Equipment Corporation Systems Research Center},
    Palo Alto, CA, October 1997

[15] On-line version of FCC Report and Order,

[16] SMPTE (Society of Motion Picture/Television Engineers),
    http://www.smpte.org/

[17] George D. Drapeau,
    Synchronization in the MAEstro Multimedia Authoring Environment,
    http://www.mae.com/ProductInfo/
[18] Shockwave, Macromedia
    http://www.macromedia.com/shockwave

[19] Quicktime plugin, Apple
    http://www.apple.quicktime.com

[20] Netshow, Microsoft
    http://www.microsoft.com/netshow

    http://www.w3.org/TR/WD-DOM/

    http://www.hytime.org

[23] J.F. Allen, Maintaining Knowledge about Temporal Intervals

[24] Yue Xu, System Architecture of WebSmart,
    M.Sc. thesis of The University of British Columbia, 1998

[25] Hypermedia Presentation and Authoring System (HPAS), DEC
    http://www.research.digital.com/SRC/HPAS/

    http://www.isgmlug.org/

    http://www.cs.columbia.edu/~hgs/rtsp/draft/draft-ietf-mmusic-rtsp-09.txt
Appendix A

Web Course Document Samples

The files in this appendix are sampled from the Web course application (the screen dump of this Web application can be found on page 69). All the files here are generated by WebSmart’s authoring tools.

A.1 Spatial layout: app1.html

```html
<HTML>
<HEAD>
<TITLE>Multimedia Document Example</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
<FRAMESET COLS="50%,50%">
  <FRAME SRC="syllabus.html" NAME="syllabus">
  <FRAME SRC="section1.html" NAME="media_app">
</FRAMESET>
<FRAMESET COLS="30%,70%">
  <FRAME SRC="sec_title0.html" NAME="title">
  <FRAME SRC="slide0.html" NAME="Notes">
</FRAMESET>
</FRAMESET>
</HTML>
```
A.2 Hyperlinks: syllabus.html

<html>
<head>
<title>Syllabus</title>
</head>
<body text="#000000" bgcolor="#FFFFFF" link="#990000"
 vlink="#333399" alink="#333399">
<h2> Table of Contents </h2>
<p><font size="+1">
1 <a href="section1.html" target="media_app">Course Overview</a></font></p>
<p><font size="+1">
2 <a href="section2.html" target="media_app">Basic Media Player</a>
2.1 <a href="section2.1.html" target="media_app">The Basic Player</a>
2.2 <a href="section2.2.html" target="media_app">Detecting Media Events</a></font></p>
<p><font size="+1">
3 <a href="section3.html" target="media_app">Quiz</a></font></p>
</body>
</html>
A.3 A section file: Section2.html

<HTML>
<HEAD>
<TITLE>Basic Media Player</TITLE>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">
<H2>2 Basic Media Player</H2>
<applet name="clock" code=Display.class width=320 height=350 MAYSCRIPT>

<br>
<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>
<br>
</applet>

<PARAM name="VIDEO-FILE" value="movies/sec6.mpg;movies/sec7.mpg">
<PARAM name="AUDIO-FILE" value="movies/sec6.mpa;movies/sec7.mpa">
<PARAM name="IMAGE-FILE" value="movies/dxu.gif;movies/lIan.gif">
<PARAM name="CAPTION-FILE" value="movies/sec6.cap;movies/sec7.cap">
</applet>

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

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

function loadEvents(){
  var APP = document.applets["clock"];
  APP.clearAllEvents();
}
APP.addEvent("changetitle", "sec_title2_1.html",
"movies/sec6.mpg", "0");
APP.addEvent("changeNotes", "slide7_1.html",
"movies/sec6.mpg", "15");
APP.addEvent("changeNotes", "slide7_2.html",
"movies/sec6.mpg", "35");
APP.addEvent("changetitle", "sec_title2_2.html",
"movies/sec7.mpg", "0");
APP.addEvent("changeNotes", "slide8_1.html",
"movies/sec7.mpg", "8");
APP.addEvent("changeNotes", "slide8_2.html",
"movies/sec7.mpg", "32");

APP.endEvent();
}
A.4 A subsection file: Section2.2.html

<HTML>
<HEAD>
<TITLE>Detecting Media Events</TITLE>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">
<H2> 2.2 Detecting Media Events </H2>
<applet name = "clock" code=Display.class width=320 height=340 MAYSCRIPT>

</applet>

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

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

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

    APP.addEvent("changetitle", "sec_title2_2.html", "movies/sec7.mpg", "0");
    APP.addEvent("changeNotes", "slide8_1.html", "movies/sec7.mpa");
</SCRIPT>

<PARAM name="VIDEO-FILE" value="movies/sec7.mpg">
<PARAM name="AUDIO-FILE" value="movies/sec7.mpa">
<PARAM name="IMAGE-FILE" value="movies/lIan.gif">
<PARAM name="CAPTION-FILE" value="movies/sec7.cap">
</applet>

<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>
</blockquote>
"movies/sec7.mpg", "8");
APP.addEvent("changeNotes", "slide8_2.html",
"movies/sec7.mpg", "32");

APP.endEvent();
}
</SCRIPT>
</BODY>
</HTML>
A.5 Quiz section file: section3.html

<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 MAYSRIPT>

   <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>
   </blockquote>

   <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">
   </applet>

   <SCRIPT>
   function changetitle(n) {
      parent.frames["title"].location=n;
   }
   
   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", "ql.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>
A.6 A Question slide: q1.html

<HTML>
<HEAD>
<SCRIPT>
function loadEvents()
{
    var
APP=parent.frames["media_app"].document.applets["clock"];
    APP.addEvent("pauseApp","movies/secll_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_question();
    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>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">

What does JMF stand for ?

<P>&nbsp;&nbsp;&nbsp;</P>

<P>&lt;FORM NAME="quiz"&gt;&lt;INPUT TYPE="text" NAME="answer" VALUE="" SIZE=50 &gt;&lt;/FORM&gt;</P>

<P>&lt;INPUT TYPE="button" VALUE="Done" NAME="submit" OnClick=check_answer()&gt;&lt;HR&gt;&lt;/P&gt;

&lt;DIR&gt;
&lt;DIR&gt;
<P>Your score is <INPUT TYPE="text" NAME="result" SIZE=10 readonly></P>
</DIR>
</BODY>
</HTML>
A.7 The last question slide: q3.html

<HTML>
<HEAD>
<SCRIPT>

function loadEvents()
{
    var
    APP=parent.frames["media_app"].document.applets["clock"];    APP.addEvent("pauseApp","movies/secl1_l.mpg","43");
}

function wrong_choice()
{
    var
    APP=parent.frames["media_app"].document.applets["clock"];    APP.one_more_question();
    document.quiz.result.value="wrong!";
    location="quiz_summary.html";
}

function correct_choice()
{
    var
    APP=parent.frames["media_app"].document.applets["clock"];    APP.one_more_question();    APP.one_more_correct_answer();
    document.quiz.result.value="correct!";
    location="quiz_summary.html";
}

</SCRIPT>
</HEAD>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">

Which of the following interface is used to present media stream ?

Which of the following interface is used to present media stream ?

<form NAME="quiz">
    <input TYPE="checkbox" NAME="choice" DEFAULTCHECKED="0" onClick="wrong_choice()">
</form>
</BODY>
<FORM>

1. A. Clock
2. B. Controller
3. C. Duration
4. D. Player

Your score is <INPUT TYPE="text" NAME="result" SIZE=10 readonly>

</FORM>

</BODY>
</HTML>
A.8 Quiz summary slide: quiz_summary.html

<HTML>
<HEAD>
<SCRIPT>

function reportGrade()
{
    var APP = parent.frames["media_app"].document.applets["clock"];
    var questions_taken=APP.getQuestionsTaken();
    var correct_answers=APP.getCorrectAnswers();
    var percentage=correct_answers*100/questions_taken;
    if (percentage>50)
        parent.frames["media_app"].location="congratulations.html";
    else
        parent.frames["media_app"].location="sorry.html";
    document.grade.total.value=questions_taken;
    document.grade.success.value=correct_answers;
    document.grade.final_grade.value=percentage;
}

</SCRIPT>
</HEAD>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="reportGrade()">
<H2 ALIGN=CENTER>&nbsp;&nbsp;&nbsp;</H2>
<H2 ALIGN=CENTER><FONT COLOR="#FF0000">Quiz Summary</FONT></H2>
<P><FORM NAME="grade"></P>
<P>You have answered <INPUT TYPE="text" NAME="total" SIZE=5 readonly>questions</P>
<P>You succeeded in <INPUT TYPE="text" NAME="success" SIZE=5 readonly>questions</P>
<P>Your grade is <INPUT TYPE="text" NAME="final_grade" SIZE=6 readonly>%</P>
</FORM></P>
</BODY>
</HTML>