Integrating temporal media and open hypermedia on the World Wide Web

Niels Olof Bouvin [a,*], René Schade [b]

[a] Department of Computer Science, University of Aarhus, Aabogade 34A, DK8200 Aarhus N, Denmark
[b] Tele Danmark Internet, Olof Palmes Allé 36, DK8200 Aarhus N, Denmark

Abstract

The World Wide Web has since its beginning provided linking to and from text documents encoded in HTML. The Web has evolved and most Web browsers now support a rich set of media types either by default or by the use of specialised content handlers, known as plug-ins. The limitations of the Web linking model are well known and they also extend into the realm of the other media types currently supported by Web browsers. This paper introduces the Mimicry system that allows authors and readers to link to and from temporal media (video and audio) on the Web. The system is integrated with the Arakne Environment, an open hypermedia integration aimed at Web augmentation. The links created are stored externally, allowing for links to and from resources not owned by the (link) author. Based on the experiences a critique is raised of the limited APIs supported by plug-ins. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Temporal media; Open hypermedia; Plug-ins; Web augmentation

1. Introduction

The World Wide Web has since its beginning steadily embraced more and more types of media. Today the average Web user will be exposed to pictures, video clips, sound recordings, music, and will interact with programs or 3D worlds residing on Web pages. These types of media are either handled by the Web browser itself or handled by specialised programs, 'viewers' or 'plug-ins'. Most media types are however supported/handled in the sense that it is possible to link to the entire media clip or to include it on a Web page, but not link from the media clip itself or to a segment of the media clip. Pictures are the exception, using image maps, to provide starting points for navigation. However all media types (HTML and otherwise) share the limitations of in-line unidirectional links[1]; links cannot originate from documents not owned by the hopeful link creator, and the destinations of a link into a HTML document are limited to the named regions in the target document. These deficiencies are being addressed by integrating open hypermedia systems and the Web, allowing link structures to be stored externally of documents. This approach also allows for links to and from media types less amenable to modification than HTML, provided that suitable plug-ins or viewers are used.

This paper describes an open hypermedia integration to provide linking facilities to and from

* Corresponding author.

[1] With the possible exception of image maps, which may have links defined externally.

temporal media (such as video and audio clips). A search for an appropriate plug-in to provide the necessary functionality left the authors empty-handed. This resulted in the implementation of the Mimicry player which substitutes for a plug-in. Through the Mimicry controller, the system interacts with the Arakne Environment [4], allowing users to create multiheaded bi-directional links to and from temporal media clips, embedded or otherwise, as well as to and from HTML documents.

The results achieved by the Mimicry system (and the relative ease of implementing the ideas behind it) raise the question, why plug-in developers do not yet provide the functionality to support such a system. It would substantially ease the work of Web page designers, as media clips or parts thereof could be reused, as well as supporting new (on the Web) technologies such as linking to and from temporal media.

The paper begins by introducing related work in the field of open hypermedia and on the Web. The merits of emerging standards such as SMIL, HTML+TIME, and XLink/XPointer are discussed. The Arakne Environment wherein the Mimicry system runs is introduced and described. The Mimicry system (player and controller) is described in detail and an example of Mimicry usage is given. Based on the experiences with the Mimicry system, the current state of plug-ins is discussed in the context of hypermedia systems. Finally directions for future work are discussed and a conclusion is reached.

2. Related work

This section will introduce the notion of externally stored link structures, open hypermedia systems, Web augmentation, and the work done with integration of temporal media and hypermedia systems.

2.1. Separating document and structure

The linking model found in the World Wide Web is based on in-line unidirectional links. While simple and scalable, this linking model is in some contexts inadequate in comparison with the linking model found in most modern hypermedia systems. In-line links are hard to maintain[2]; it is impossible to determine which links point to a page[3]; there can be only one set of links in a document; a link may point to only one destination rather than many; and this destination is limited to either a whole document or a named region therein. The problems with this approach can be illustrated by the following example. Consider a situation, where a company decides on a Web based Intranet solution to allow easy access to their technical documentation. The Web, given the URL naming scheme, is very well suited for document distribution. The technical documents are crucial to several independent groups in the company and they all wish to put links into the technical document. Current Web technology yields two scenarios: either give each group access to copies modifiable by the respective group, or incorporate all links into one central document. The former makes updating the technical documentation difficult, and the latter will clutter the technical document with links interesting to one group, but irrelevant to the rest. Both cases leave the technical document open to undesirable modification. A safer and more maintainable solution would be to have one copy of the documentation available and then have each group use their own set of links into the documentation. This can not be done with the existing Web linking model.

A way to achieve this kind of flexibility is through the use of anchor-based hypermedia, which separates the anchor (or endpoint) and the link from the document. An anchor specifies a span in a document (up to the entire document), an area in a picture, a segment of a video clip and so forth. Links specify relationships between anchors, as seen in Fig. 1. Anchors and links are stored outside the documents. A hypermedia application used by a user inserts links into documents *as they are retrieved* (e.g. not at the server, but at the client). The user can through the hypermedia application decide which sets of links to use, and the company described above would thus be able to maintain several sets of links to the same technical documentation.

[2] In the sense that links may point to documents that do no longer exist or have been moved.

[3] Save a brute force approach using search engines, which is computationally expensive and not necessarily accurate.

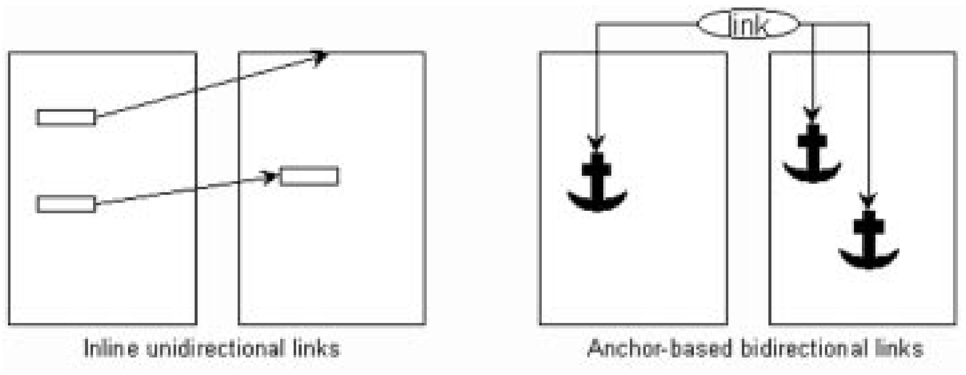

Fig. 1. In-line and anchor-based linking.

There are pros and cons of this approach. As links and anchors are now separate they can be maintained separately and can be checked and updated independently of the documents they link. Links can be bi-directional, multiheaded and link into documents without modifying them. However, the insertion of links into the document on the fly carries some additional overhead, and does generally not scale as well as the simpler in-line link model.

A great benefit of the anchor-based linking approach is that of opaque anchors, that is, the general system is not concerned with how an anchor addresses a selection in a media type. The general model need not be modified if a new anchor type is introduced to support a new media type, as long as the new anchor type adheres to the general anchor specification. The anchoring code (such as the ability to display an anchor into the media type) must of course still be written, but storage, link following, etc. is unaffected. This allows for complex anchoring constructs, and allows the developers to support new applications and media types without sacrificing existing work.

Anchors are created to match their media type. They must carry enough information to be able to identify the selection that the user had in mind when the anchor was created. In the case of text anchors, this information often consists of a selection and a context around the selection, so that it may be uniquely identified. An image anchor could (depending on its use) be as simple as two co-ordinate pairs to identify a rectangular selection, or be complex enough to identify arbitrary shapes as destinations. In the context of temporal media, it is natural to use time as unit in the system of co-ordinates. Frames are another possibility but the frame rate of a media clip may vary accordingly to the bandwidth of the user's connection. Furthermore some temporal media types, such as audio, may not have frames at all.

2.2. Open hypermedia and the Web

Open hypermedia systems are characterised by a focus on integration with third-party software. Historically, hypermedia systems have often been closed and monolithic, requiring other programs to comply to the standards of the hypermedia system in order to provide hypermedia functionality. This is problematic as it closes the door on existing non-compliant applications, and expects developers to change their programs — an unlikely proposition at best. Recognising that most people are not willing to throw away their applications in order to utilise a hypermedia system has led the open hypermedia community to integrate existing applications into their hypermedia systems instead. The level of which this is possible varies according to the applications, ranging from the simple (show document) to the advanced (a full integration). Whitehead handles the implications of integrating third-party applications with open hypermedia integration in [21].

A natural consequence of the open hypermedia approach is the interest of integrating the Web into open hypermedia systems. Several groups in the field have created Web integration tools, and a non-exhaustive list includes DLS [5,6], DHM/WWW [8], Webvise [9], Navette [3], Chimera [1], and HyperWave [18].

A common approach to open hypermedia Web integration is to modify Web pages while en route to the Web browser. This modification usually consists of the addition of links or other kinds of structure. These links are stored on a structure server and are inserted into the pages using CGI-scripts, proxies, or programs controlling the Web browser. The interface presented to the user varies, ranging from no interface at all to full authoring applications allowing the users to modify and extend upon the existing collections of links and anchors. Most, however, allow the user to create links to and from whatever pages the user may desire, thus alleviating one of the major limitations of the existing Web architecture mentioned above. For a fuller discussion on the techniques of open hypermedia Web integration and Web augmentation in general, see [4].

The main target of these integrations has been Web pages rather than other kinds of Web-distributed media, as HTML is easily analysed and modified by use of proxies or other means. Other kinds of media that could be interesting include graphics and temporal media, streaming or otherwise. These data types are less readily modified, come in great variation, and requires viewers or plug-ins that may be difficult to integrate with an open hypermedia system.

2.3. Hypermedia and temporal media

Several hypermedia systems have been extended or devised specifically to handle temporal media. The most influential hypermedia model to incorporate the notion of temporal data is HyTime [7], which allows for multidimensional anchors (including the temporal dimension). HyTime is a general standard aimed at interchange and does as such not specify the interaction between hypermedia application and multimedia applications. While many systems have integrated temporal media one way or another, some systems go further, such as AEDI [2], which tries to ease work with large amounts of temporal data with structured indexing. Some systems try to facilitate automatic tracking and location of anchors in media clips; two well-known examples are Himotoki [10] and MAVIS [15]. While the possibility of selecting an object in one frame, and have the system automatically track the object in the rest of the video clip certainly is alluring, it is also quite computationally expensive, and probably unlikely in a Web setting.

The ambitions of the authors are far more modest. We are merely interested in being able to create links to and from segments of temporal media. Future version may include anchors that cover an area of a video clip for some duration, but the area is not expected to move.

2.4. Temporal media on the Web

The use of temporal media on the Web has steadily increased. It is today commonplace for news sites to bring either video clips or to provide access to whole TV shows online. Likewise the research community (notably the linguistic) has taken to making animations or sound recordings available. This opens for a hitherto unseen availability of valuable research material (such as historic film clips or language recordings), and therefore also an increasing demand to be able to interact with these media clips in new and innovative ways.

An example of an organisation beginning to put large amounts of temporal data on the Web is the Danish national library, Statsbiblioteket. In order to facilitate research, the library has made an increasing amount of historically interesting Danish sound files available on the Web [20].

2.5. Emerging standards

Some of the new emerging W3C standards are of special interest to the subject of this paper. This section will briefly describe these and discuss the implications for temporal media on the Web. It should be noted, however, that these are still evolving standards, and can be expected to undergo some transformation before being finalised.

SMIL [19] (Synchronised Multimedia Integration Language) is a recent Recommendation from W3C. It is an effort to support the layout and synchronisation of multimedia clips, e.g. synchronising a video clip with animations or a slide show with audio narrative and HTML documents. The standard is presentation oriented, and as such requires an authoring tool to modify.

HTML+TIME [12] (Timed Interactive Multimedia Extensions for HTML) is a proposed standard inspired by SMIL, and is concerned with the implementation of SMIL concepts in HTML. It thus adds a concept of timing to HTML and allows any HTML element to appear for a defined duration. Of special interest in this context is the timed hyperlink, and the integration of media players, which can be addressed and timed. Not only can the players be set to begin playing at a predetermined time, it is also possible to address inside the media clip and thus play a segment of a media clip, using the `clip-begin` and `clip-end` attributes. As HTML elements (in particular links) can be synchronised with other elements' timing events, it would be simple to create links that would synch with segments of a media clip, e.g. links appearing during a segment of a video clip and disappearing after the segment had been played.

As such HTML+TIME has certain requirements of content handlers that fit nicely with the requirements of links to and from temporal media. Specifically media players should be able to synch with events, as well allow the showing of segments. This is however a very new (proposed) standard and so far no Web browsers or media players the authors are aware of meet the HTML+TIME requirements.

A general problem with SMIL and HTML+TIME from the standpoint of the authors is that these standards address authoring of presentations and thus is in principle (though admittedly far more advanced) no different than the existing HTML authoring tools. The basic problem of links being created exclusively by the author of the document remains. These standards also address a somewhat different problem than linking to and from temporal media, as a key point of SMIL and HTML+TIME is synchronisation and presentation.

Both standards are, as noted above, still in development and may very well change in the future.

XML defines a way to describe structured data and documents. XPointer and XLink (XML Linking Language) are the accompanying resource location and linking standards. As of this writing, neither is in 'Recommended' state from W3C, but this is expected to happen sometime in 1999.

XPointer [17] is designed to specify locations in XML and other well-structured documents. The syntax is based on the Text Encoding Initiative 'extended pointer'. This is a rather compact and tree-oriented notation, which might take the following form: `id(foo).child(3,SEC).child(4,LIST)`. This expression would start at the entity identified as 'foo', continue at this entity's third child of the type 'SEC', and end with that child's fourth child of the type 'LIST'. This format is not XML, which might come as a surprise, but it has the advantage that it can be used in URLs.

XLink [16] is the language that ties XPointers together in links. XLink links are not limited to XPointers — while endpoints in XML documents typically will be described with XPointers, XLink links can have any Web resource as a destination. Links may be in-line (as in HTML) or out-of-line, that is residing outside the linked documents. XLink supports bi-directional multiheaded links. Out-of-line links may be stored in simple text files or handled by link bases. XLink does not offer any protocol for such link servers.

XLink and XPointer are, from a Web augmentation standpoint, mainly interesting if XML becomes widespread on the Web. While the linking constructs are quite powerful in XML documents, the standards are not aimed at improving the state of the art with respect to HTML, that is not well-formed, nor does it address the intricacies of other media types, such as video or audio.

3. The Arakne Environment

The Arakne Environment, shown in Fig. 2, is a runtime environment aimed at supporting Web augmentation tools. The environment is primarily but not exclusively aimed at providing advanced hypermedia functionality to the Web. The environment is based on the Arakne framework [4], which is a general Web augmentation model, and was designed to be open and extensible. It currently supports a navigational hypermedia tool, Navette, and a guided tour tool, Ariadne [14].

The framework may support any number of Web augmentation tools. These tools (known as 'navlets') are dependent on four core components of the Arakne framework: the Operations, the Hyperstructure Store, the Browser, and the Proxy. The navlet is the domain-specific part of a Web augmentation tool. It provides a user interface as well as special logic to handle the specific domain. This may in-

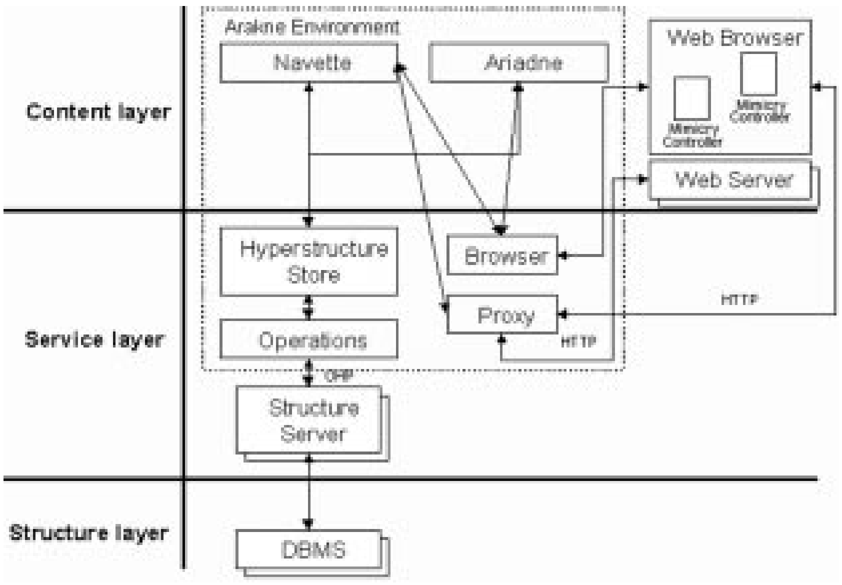

Fig. 2. The Arakne Environment.

clude deciding which links to display in a Web page based on information retrieved from the Hyperstructure Store component, or interfacing to the Proxy component for analysis of documents or to modify documents. Depending on the situation the computation and analysis may be carried out by the navlet or by another component.

The Operations component models the communication with the structure server layer. This component will thus typically support the same services as the structure server(s). This is where on the wire issues, such as network communication, marshalling, and multiplexing, are handled.

The Hyperstructure Store is the interface between the navlets and the Operations. The Hyperstructure Store provides convenience functions for the navlets, as well as caching the results of the queries retrieved with Operations. The Hyperstructure Store will also alert navlets to changes in the structures they subscribe to.

The Proxy component models the modification and analysis of Web content. Depending on their domain, navlets may require the Proxy to modify Web pages, and these requests for modifications are collected by the Proxy and used to modify the Web page. Other navlets may require access to the content of a Web page, which is also handled by the Proxy.

The Browser component models the user's Web browser. Through the Browser navlets can retrieve and modify the state of the Web browser such as which in URL is currently displayed, the structure of the current frame set, whether a selection has been made in a frame, and if so, what and where. Communication with plug-ins and applets running in the Web browser is also handled through the Browser component.

The situation depicted in Fig. 2 is a situation of two navlets running in the Arakne Environment: Navette is a link creation tool, and thus needs access to the Proxy in order to insert links into Web pages. Ariadne is a guided tour tool and does not modify Web pages, and is thus not connected to the Proxy. Both, however, need to be able to tell and set the state of the currently displayed documents and to interact with the Web browser in other ways, as well as retrieving data from the structure server through the Hyperstructure Store.

By providing the components described above and by having an open architecture, the Arakne Environment aims to provide developers with an environment that allows for easy implementation of Web augmentation tools. The Arakne Environment is written in Java (but is not an applet), and is currently integrated with the Microsoft Internet Explorer. The current version uses the DHMProxy [9] to insert links and other structures into Web pages. The relative ease of development has made the environment well suited for experiments such as the Mimicry system described herein.

4. The Mimicry player

The Coconut project has developed the Mimicry player to support linking to and from temporal media in the Arakne Environment. Realising that if we wanted to integrate temporal media into our hypermedia system, we would have to do it ourselves, as no existing plug-in seemed to offer an adequate API, we started looking for the easiest way to support temporal media.

The Java Media Framework [13] developed by JavaSoft, Intel, and Silicon Graphics is aimed at supporting temporal media in Java. The framework supports a wide range of video and audio formats [4], and more CODECs can be added. The Mimicry player is a Java Bean encapsulating the Java Media Framework player. It is basically interfaceless, but implements a rich API and event interface that can be utilised by other components.

The Mimicry controller is, referring back to Fig. 2, an applet that communicates with the Navette tool through the Browser component; it thus acts as an intermediary between the Arakne Environment and the Mimicry player. The Mimicry controller provides the interface to the Mimicry player as well as to Navette. The Mimicry controller acts as the interface of the Mimicry player to the user. The controller has a control panel for the browsing and creation of anchors, which can be displayed by right clicking on the player window.

4.1. Web page modification by the DHMProxy

Multimedia files are normally presented in a Web browser using the appropriate plug-ins, according to the MIME type of the file. This is done either by using a direct link to the file or by embedding it into a Web page using <EMBED> or <OBJECT> tags. The Mimicry player is designed to mimic plug-ins, and will appear (to the user) as an ordinary plug-in on a Web page.

Rather than depending on Web page designers to adopt the Mimicry player as the standard viewer applet, the system utilises the DHMProxy [9] to modify Web pages, so that the Mimicry player is used instead of plug-ins.

The DHMProxy is aware of the formats supported by the Mimicry player and changes the Web pages accordingly. If a Web page embeds temporal media using <EMBED> or <OBJECT> tags, these tags are replaced with a corresponding <APPLET> tag with the same layout dimensions. If a Web page has a direct link to a temporal media file, the DHMProxy returns a Web page containing the <APPLET> tag in the body of the page. Using this approach it is possible to translate plug-in invocations into applet invocations, without changing the layout of the Web pages.

The DHMProxy also takes care of inserting anchors from the chosen link collections on the structure server into the Mimicry controllers. All the anchors residing in a media clip are passed on to the applet in parameter tags consisting of name, id, and time span. When the Web page is loaded in the Web browser and the Mimicry controller applet is launched, it is thus aware of the anchors in the media clip. This modification or decoration is basically similar to the ordinary text decoration (that is, insertion of links) done by the DHMProxy.

4.2. Mimicry in action

In order to present Mimicry in action, we have started the Arakne Environment, which launches the Web browser [5], as presented in Fig. 3. The Web browser has been configured to use the DHMProxy. As we want to create links into Web pages and media files, we have started Navette in the Arakne Environment.

The situation in Fig. 3 is as follows. Another user has earlier created a link with the name ‘Link 10’ in the link collection ‘Hyperspace1’, which contains three anchors. The first anchor ‘Harrison Ford’ orig-

[4] Supported formats include AIFF, WAV, AVI, MIDI, MPEG-1, and QuickTime.

[5] The current version is integrated with the Microsoft Internet Explorer.

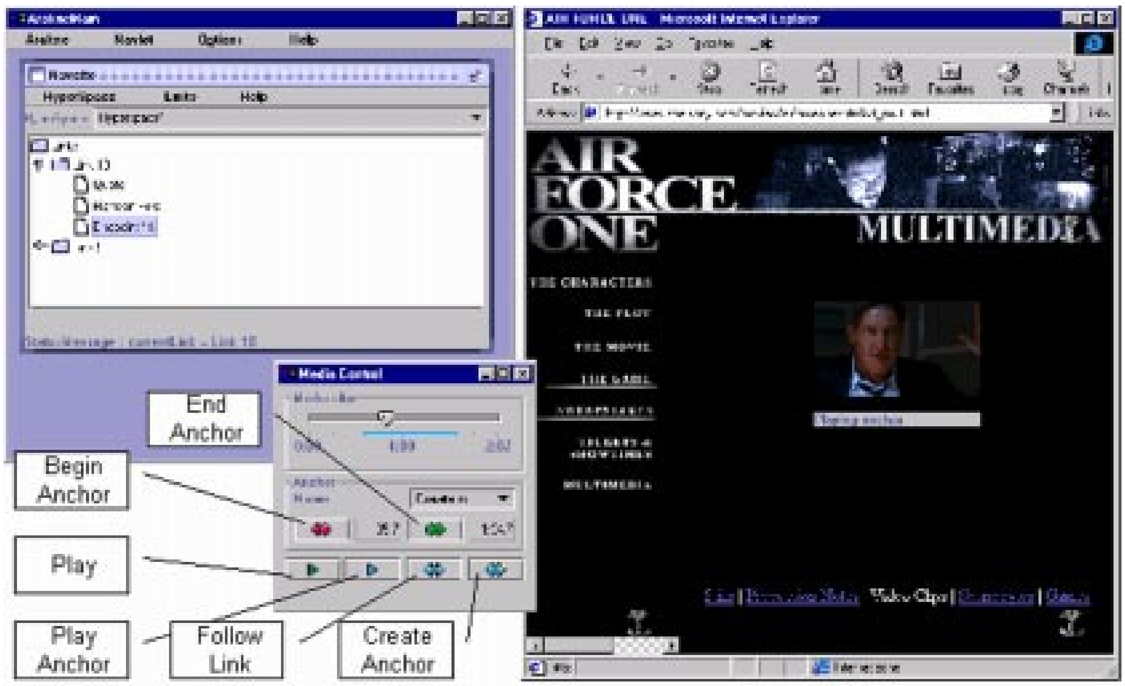

Fig. 3. Playing the anchor endpoint 'Endpoint 14'.

inates in a Web page containing a description of the actor Harrison Ford. The second anchor 'Quote' originates in a Web page containing famous movie quotes, referring a specific quote. The third anchor 'Endpoint 14' originates in a movie clip embedded in a Web page, referring to a 59 s long segment of the 2:02 min long clip, that features the quote.

Browsing the Web using the Arakne Environment, we encounter the Web page containing the movie quotes. Since we have 'Hyperspace1' opened, the DHMProxy has decorated the page with the 'Quote' anchor. The anchor is presented, so that we can follow 'Link 10' either to the media segment 'Endpoint 14' or to the Web page containing the anchor 'Harrison Ford'. We decide to follow the link to the media anchor and the Web page containing the video clip is loaded, as shown in Fig. 3.

The DHMProxy decorates the Web page by substituting the plug-in tag with an applet tag and anchors from the structure server. When the Mimicry controller applet is launched it retrieves its anchors from the parameter tags, alerts the Arakne Environment to its existence and asks the Mimicry player to start downloading the media file. In Navette 'Link 10' is opened and the 'Endpoint 14' anchor is in focus.

Clicking on the Mimicry control applet, the 59 s segment 'Endpoint 14' will be played, starting at 25.7 s and ending at 1:34.7. Interested in the movie clip we right-click on the applet and a popup menu appears. In the popup menu it is possible to select other anchors in the clip or to open the control panel. We choose 'open control panel', and the control panel, on top of the Web browser and the Arakne Environment in Fig. 3, is presented. The control panel consists of a slider, indicating the length and current position of the media clip, buttons for creating/editing anchors and a dropdown menu containing all the anchors in the media clip. The current anchor is drawn on the slider.

Dragging the slider to the start position and pressing the 'Play' button causes the Mimicry player to start playing from the beginning. While watching the media clip, we notice the actor Gary Oldman, and decide to find more information about him. In the dropdown menu containing all the anchors in the media clip, we find another media anchor named 'Endpoint 11'. Selecting 'Endpoint 11' and playing

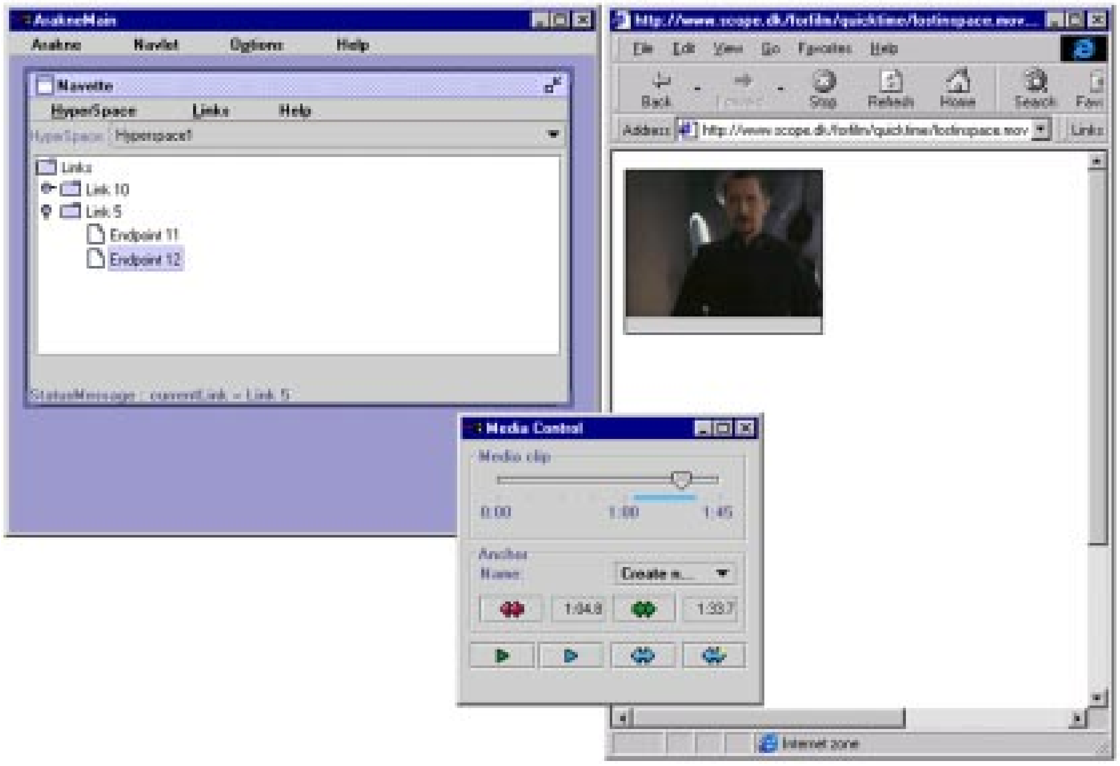

Fig. 4. Playing the anchor ‘Endpoint 12’.

it, we realise that the anchor is covering the part of the clip depicting Gary Oldman. Wondering if the link is about the actor, we click the ‘Follow Link’ button on the control panel. The browser loads another URL, which is a direct link to another movie clip, resulting in the situation shown in Fig. 4.

Another Mimicry controller is launched and Navette has changed its focus to ‘Endpoint 12’ in ‘Link 5’. Clicking the Mimicry controller, the segment ranging from 1:04.8 to 1:33.7 of the movie file is played. The ‘Endpoint 12’ segment does indeed refer to Gary Oldman appearing in another movie trailer.

Watching the movie trailer from the beginning, we find yet another actor, Matt LeBlanc. Since we have had the pleasure of following links created by others, we would like to add an additional link to the current link collection. We know where to find further information about Matt LeBlanc and decide to create this relation. In Navette we deselect the current link. In the control panel, we create a new media anchor using the ‘Begin Anchor’ and ‘End Anchor’ buttons, editing a few times, reviewing the new anchor several times with ‘Play Anchor’ and finally end up with exactly the portion of the clip concerning Matt LeBlanc. We press ‘Create Anchor’ in the control panel, and as no links are selected in Navette, a new link ‘Link 25’ is created, initially containing the anchor ‘Endpoint 15’. The Arakne Environment stores the new link and anchor on the structure server. We browse the Web to a Web page containing more information about the actor as shown in Fig. 5.

When the Web page is retrieved, we highlight the text ‘Matt LeBlanc’ in the Web browser, right click and select ‘Add Anchor’ in the popup menu[6]. This information is sent to Navette, which reacts by creating the endpoint ‘Matt LeBlanc’. Pressing the refresh button in the Web browser forces the

[6] The essential commands for link creation with Navette are available through the use of right-click menus on highlighted text in the Microsoft Internet Explorer.

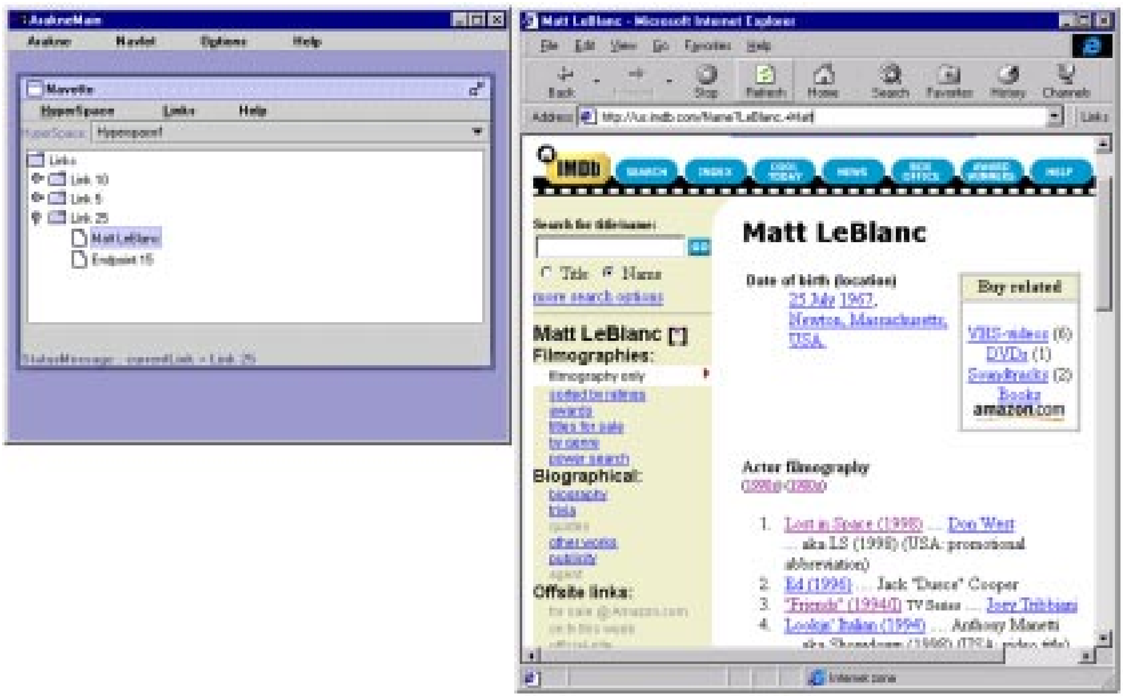

Fig. 5. Creating a HTML anchor using Navette.

DHMProxy to decorate the Web page with the newly created anchor. In the Web browser, the new anchor is presented as a link in '[*]' next to the Matt LeBlanc text. To test the link, we click it, and the media clip from Fig. 4 is loaded, ready to play 'Endpoint 15'. If we open the control panel and press the 'Follow Link' button the Web page from Fig. 4 is loaded. We have now created a link in the link collection 'Hyperspace1'. Next time another user is browsing the Web using the Arakne Environment and the 'Hyperspace1' link collection, he or she will be able to see and follow these links.

We have now described how to follow a link to a media clip using Mimicry, how to follow links between two media clips, and how to create anchors and links. In Fig. 3 the Mimicry is used as embedded on an HTML page, and in Fig. 4 Mimicry is used stand-alone as a direct link to a media clip.

5. Discussion

The following will discuss the implication of the results in the context of plug-in developers.

5.1. The problem with current plug-ins

Plug-ins are used to handle media types not supported natively by the Web browser. Plug-ins can be controlled at runtime through LiveConnect [11] using JavaScript. In order to support linking in and out of temporal media, we need to continuously be able to get and set the state of the plug-in. The degree of openness to this kind of control varies tremendously. The most extreme example of an open plug-in handling temporal media that the authors have been able to locate is the Beatnik plug-in published by Headspace. Beatnik is a plug-in targeted at sound and music files, and has a very rich API. The APIs supported by some popular plug-ins are shown in Table 1. Beatnik is sound only, and is as such only used as a comparison to the other media players, as well as Microsoft Media Player which cannot be controlled through JavaScript.

Apart from Beatnik, no plug-in allows for the playing of a designated but not in the media predefined segment. This ability is fundamental to link following in temporal media, and it is thus not possible to create a hypermedia system supporting links

Table 1
Supported APIs of various plug-ins

Plug-in/content handler	Methods [a]	Callback methods [a]
Real RealPlayer	SetSource DoPlayPause DoNextItem [b] DoPrevItem [b]	onClipOpened onGoToURL
Apple QuickTime	[c]	[c]
HeadSpace Beatnik	getPlayLength getPosition setPosition setStartTime setEndTime	onLoad onReady onStop
Microsoft Media Player [d]	GetCurrentPosition SetCurrentPosition GetSelectionStart SetSelectionStart GetSelectionEnd SetSelectionEnd	

[a] This is not a comprehensive list, but merely methods relevant to linking.
[b] These methods require that the items are defined by the author of the media clip.
[c] QuickTime does only support arguments to the `<embed>` tag.
[d] The methods described are through the COM-interface, not JavaScript.

to and from media clips with these plug-ins. The rich authoring environments of some media tools, especially QuickTime[7], make the lack of support for modifications at run-time more grating. The author should certainly have a very rich authoring tool at his or her disposal, but that does not eliminate the need to be able to dynamically alter the playback of a media clip, as it is shown on a Web page.

Integration with plug-ins requires some degree of openness on part of the plug-in API. In order to support linking (and other kinds of integration) it must at least be possible to designate a segment of a recording and to allow that segment to be played. Specifically we would suggest that the methods outlined in Table 2 could be the basic API of any plug-ins handling temporal media.

A plug-in would thus be able to play a designated clip or segment of a media clip and would report its progress through the media clip. This API would allow a developer to support the kinds of interactions supported by the Mimicry player through the use of JavaScript and LiveConnect [11]. This API is simple and should be easy to implement for plug-in developers. The Beatnik plug-in clearly demonstrates that this is indeed achievable.

Another solution investigated by the Coconut project is to make a specific integration with a media player, in this case the Microsoft Media Player that through its COM-interface supports a rich API. External applications would register events from the Media Player and through the COM-interface control it. Being a COM-component the Media Player can be integrated into a user interface supporting anchor and link creation. This approach does have some limitations. It is platform-dependent, and unless a proxy is used to modify the Web pages as described below, the supporting software is forced to change content handler while the Web page containing the media clip is being displayed. This is possible, but it is neither elegant nor seamless.

6. Future development

Unless a media player emerges that satisfies the needs of temporal media linking, the authors will continue to work on the Mimicry player. Some current development holds promise with regards to future versions. Future versions of the Java Media Framework will support more media formats and sport additional improvements. An interesting development would be Java Media Framework support for streaming, which is currently offered by Real Net-

Table 2
Basic API requirements for temporal media plug-ins

Methods	Callback methods
getSourceURL setBeginClip getBeginClip setEndClip getEndClip getDuration getCurrentTime playPause playClip	onLoad callbackWhen

[7] However, QuickTime offers another approach as mentioned in Section 6.

works, but this version is so far limited to Netscape Communicator.

As media players and Web browsers supporting SMIL and HTML+TIME become available, we will investigate the possibilities of linking into the new structures supported by these standards, as well as using the new generation of plug-ins and viewers supporting these formats.

Apple has created QuickTime for Java, featuring an API similar to the Java Media Framework, being able to playback formats supported by the QuickTime plug-in 2.0 in an applet. This may be an area worthy of future investigation.

The current Mimicry player is a Java applet, and a clear future direction would be to convert the player into a plug-in. This would probably eliminate much of the need for the intervention by DHMProxy, as the Mimicry plug-in would automatically launch on its registered MIME types.

7. Conclusion

We have described a system allowing users to dynamically create links to and from temporal media on the Web regardless of the users' ownership of the Web pages or media clips involved. The system has not been created to compete with existing media players, but merely to allow for experimentation and to highlight the lack of support for dynamic uses of temporal media currently found in most plug-ins. As such it has been successful.

The future of temporal media on the Web is a bright one. Bandwidth will continue to rise and emerging standards such as SMIL and HTML+TIME will support the use of temporal media in new ways. Whether these standards will converge to support only presentation, or open for more dynamic uses remains to be seen.

Acknowledgements

The authors are members of the **Coconut project**[8], a joint research project consisting of Department of Computer Science, Aarhus University and Tele Danmark Internet. The Coconut project is supported by the **Danish National Centre for IT-Research**[9]. The authors wish to thank Niels Husted and René Thomsen for adding to the code, Peter Ørbæk for creating the DHMProxy, and the anonymous reviewers for good suggestions.

[8] http://www.cit.dk/coconut/

[9] http://www.cit.dk/

References

[1] K.M. Anderson, Integrating open hypermedia systems with the World Wide Web, in: Proc. ACM Hypertext '97 Conference, Southampton, 1997, pp. 157–166.

[2] G. Auffret, J. Carrive, O. Chevet, T. Dechilly, R. Ronfard and B. Bachimont, Audiovisual-based hypermedia authoring: using structured representations for the efficient manipulation, of AV documents, in: Proc. ACM Hypertext '99 Conference, Darmstadt, 1999, pp. 169–178.

[3] N.O. Bouvin, Designing Open Hypermedia Applets: Experiences and Prospects, in: Proc. ACM Hypertext '98 Conference, Pittsburgh, 1998, pp. 281–282.

[4] N.O. Bouvin, Unifying strategies of Web augmentation, in: Proc. ACM Hypertext '99 Conference, Darmstadt, 1999, pp. 91–100.

[5] L.A. Carr, D. De Roure, W. Hall and G. Hill, The distributed link service: a tool for publishers, authors and readers, in: Proc. 4th Int. World Wide Web 95 Conference, Boston, 1995.

[6] L.A. Carr, W. Hall and S. Hitchcock, Link services or link agents? in: Proc. ACM Hypertext '98 Conference, Pittsburgh, 1998, pp. 113–122.

[7] S.J. DeRose and D.G. Durand, Making HyTime Work, Kluwer, Rotterdam, 1994.

[8] K. Grønbæk, N.O. Bouvin and L. Sloth, Designing Dexter-based hypermedia services for the World Wide Web, in: Proc. ACM Hypertext '97 Conference, Southampton, 1997, pp. 146–156.

[9] K. Grønbæk, L. Sloth and P. Ørbæk, Webvise: Browser and Proxy support for open hypermedia structuring mechanisms on the WWW, in: Proc. 8th Int. Conference on World Wide Web, Toronto, 1999.

[10] K. Hirata, Y. Hara, H. Takano and S. Kawasaki, Content-oriented integration in hypermedia systems, in: Proc. ACM Hypertext '96 Conference, Washington, DC, 1996, pp. 11–21.

[11] R. Hoque, Java, JavaScript and Plug-In interaction using Client-Side LiveConnect, http://developer.netscape.com/docs/technote/javascript/liveconnect/liveconnect_rh.html

[12] HTML+TIME, http://www.w3.org/TR/NOTE-HTMLplusTIME

[13] Java Media Framework, http://www.javasoft.com/products/java-media/jmf/index.html

[14] J. Jühne, A.T. Jensen and K. Grønbæk, Ariadne: a Java-based guided tour system for the World Wide Web, in: Proc. 7th Int. World Wide Web ’98 Conference, Brisbane, 1998.
[15] P.H. Lewis, H.C. Davis, S.R. Griffiths, W. Hall and R.J. Wilkins, Media-based navigation with generic links, in: Proc. ACM Hypertext ’96 Conference, Washington, DC, 1996, pp. 215–223.
[16] E. Maler and S.J. DeRose (Eds.), XML Linking Language (XLink) design principles, 1998, http://www.w3.org/TR/NOTE-xlink-principles
[17] E. Maler and S.J. DeRose (Eds.), XML Pointer Language (XPointer), 1998, http://www.w3.org/TR/WD-xptr
[18] H. Maurer (Ed.), Hyper-G Now, HyperWave: The Next Generation Web Solution, Addison-Wesley, Harlow, 1996.
[19] SMIL, http://www.w3.org/AudioVideo/
[20] Statsbiblioteket, Dansk Lydhistorie, http://www.sb.aau.dk/dlh/
[21] E.J. Whitehead Jr., An architectural model for application integration in open hypermedia environments, in: Proc. ACM Hypertext ’97 Conference, Southampton, 1997, pp. 1–12.

Niels Olof Bouvin is a Ph.D. student in Computer Science at the University of Aarhus, Denmark. His research interests include open hypermedia systems, Web augmentation, structural computing, and collaboration on the Web. He is currently involved in the Coconut project, a co-project between the Department of Computer Science and Tele Danmark Internet. Niels Olof Bouvin received his master’s degree in 1996 from Department of Computer Science, University of Aarhus, Denmark.

René Schade is a system developer at Tele Danmark Internet, Denmark. He is currently working at the Coconut project, a co-project with the Department of Computer Science, University of Aarhus, Denmark. He received his master’s degree in 1997 from the Department of Computer Science, University of Aarhus. His research interests are: World Wide Web; Hypermedia and Multimedia and Dynamic Programming Environments.