Designing Dexter-based hypermedia services for the World Wide Web

Kaj Grønbæk, Niels Olof Bouvin, and Lennert Sloth
Computer Science Department
Aarhus University,
Ny Munkegade, Bldg. 540, 8000 Århus C, Denmark
Phone: +45 8942 3188
E-mail: {kgronbak,bouvin,les}@daimi.aau.dk

In proceedings of Hypertext 97–The Eighth ACM Conference on Hypertext
Southampton, UK, April 6-11, 1997

ABSTRACT

This paper discusses how to augment the WWW with a Dexter-based hypermedia service that provides anchors, links and composites as objects stored external to the Web pages. The hypermedia objects are stored in an object-oriented database that is accessible on the Web via an ordinary URL. The Dexter-based hypermedia service is based on the Devise Hypermedia framework. Three client solutions are described and discussed, one that is platform independent based on Netscape Navigator 3.0, utilizing Java, Javascript, and LiveConnect, and two that are platform dependent, one utilizing Netscape plug-ins, and another using Microsoft Internet Explorer 3.0, utilizing mainly ActiveX. The server part is developed as a specialization of the Devise Hypermedia framework accessible through CGI scripts. Thus the system provides the full power of Dexter-based hypermedia to arbitrary Web pages on the Internet. This includes the ability for multiple users to create links *from* parts of HTML Web pages they do not own and support for creating links to parts of Web pages without writing HTML target tags. Support for providing links to/from parts of non-HTML data, such as Quicktime movies or VRML documents will also be possible in the future provided that appropriate open plug-in modules become available.

KEYWORDS: Open hypermedia service, link objects, World Wide Web, HTML, Dexter hypertext reference model, Java, JavaScript, ActiveX, OLE.

1. INTRODUCTION

The hypermedia pioneers Bush, Nelson and Engelbart [5, 8, 24] formulated a grand hypermedia vision that included support for global distributed hypermedia structures which meant to include all human writings, and support people in searching, navigating, reusing and augmenting the giant "Docuverse." However, for several decades the hypermedia implementations produced were local and non-distributed. NLS/Augment could be distributed on the early Arpanet, and a few systems like Intermedia and KMS could be distributed in local area networks, but many systems like NoteCards[18], HyperCard[9], and Guide were restricted to run on a single workstation. All these early systems were built around a special kind of "database" or proprietary file format which made distribution a very complex issue.

In the 90s, however, parts of the grand vision became a reality by means of the World Wide Web [3, 4], which was constructed around much simpler principles than the earlier systems: a tagged ASCII file format with embedded jump addresses, a uniform Internet addressing schema, and an enhanced file transfer protocol. The WWW has become a popular and efficient means of distributing information with simple hypermedia links world wide on the Internet.

The vision put forward by the pioneers and many of the early systems, however, included flexible support for people to freely create links between documents in the Docuverse. For example, the Intermedia system[20] was the first to use the term Web for a hypermedia structure and the first to provide two-way links as objects between so-called anchor objects (called 'blocks' in Intermedia) stored apart from the document contents. With this approach Intermedia let users create two-way links between documents they did not own, as well as inspect which documents were linked *to* a given document. This idea of links and anchors as separate objects became the central idea of the Dexter Hypertext Reference Model [16] and systems based on this model, e.g. Devise Hypermedia[11, 13].

As of today, the WWW provides little support for dynamic link creation and collaboration. But research on augmenting the WWW with link services that store link information separate from the document contents is underway. Hyper-G [1] and Microcosm's Distributed Link Service (DLS) [6]

are examples of systems that support non-embedded links to WWW pages on the Internet and link storage in hypermedia databases on arbitrary Internet servers.

These next generation WWW systems are, however, only in their infancy with respect to providing: linking inside pages, in particular non-text pages; linking in documents as they are being edited; collaboration support and distribution of hypermedia databases. This paper contributes to the design of next generation Web systems by discussing approaches to augmenting the WWW with support for Dexter-based hypermedia structures. These approaches are based on the work by Grønbæk and Trigg [14] on extending the Dexter model to handle both link objects and embedded addresses. Implementation experiments with the DHM/WWW system built on the Devise Hypermedia (DHM) framework [13] are discussed.

2. EMBEDDED ADDRESSES VERSUS LINK OBJECTS

A characteristic of the WWW is the use of embedded unidirectional links also known as jump addresses whereas the Dexter model advocates external links. We outline here the main differences between these two strategies, and explain why we find the external link model desirable. Jump addresses and external links are illustrated in Figure 1 and their features are schematically compared in Table 1.

The jump addresses found in the WWW are of the now familiar form `<a href="http://www.authors.jp/mishima.html#oeuvre">Yukio Mishima</a>` where `www.authors.jp` names the server where the file `mishima.html` containing the label `oeuvre` is found.

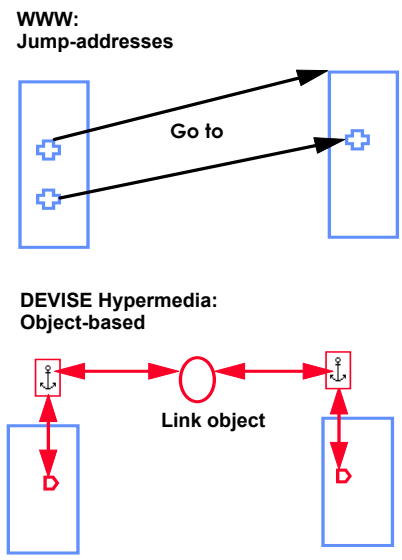

Figure 1: Jump addresses and external links

The biggest advantage of the embedded links in WWW is their simplicity: there is no need for a specialized link server, and the WWW only has to manipulate tagged ASCII files. This simplicity comes at a cost however, as only the owner of a document can create links from the document. At the same time links to specific parts of a document can only be made if there are already target tags at the desired point in the document. It is impossible to see which documents point to a document, and there can only be one set of links from a given document. If two users wish to have different links from the same document, they must maintain two copies of the document, identical apart from the different links. This requires extra maintenance, if the original document is later changed.

This situation is further complicated if the original document is not a simple ASCII file. Documents must first be converted into HTML before they can be properly used in a WWW context. This problem has been addressed by a variety of conversion programs, from simple RTF to HTML conversion to large scale WWW publishing systems such as Interleaf's Cyberleaf [2]. Powerful as some of these systems are, their existence is indicative of the problematic nature of HTML.

Links as implemented in the WWW are unidirectional and with only modest typing support; link typing as suggested by e.g. Nelson [24] or Trigg [26] are not possible.

External links are considerably more complicated, but offer advantages over embedded links. Because the original document is unaffected by link creation, users can easily create, maintain and share their own links for a given document, regardless of who owns the document. As links in systems based on the Dexter model [16] are first class objects in a database, links to and from a document can be traced from the document by querying the link database. Links are named and have types, so the user can differentiate between e.g. a quote link and reference link. Typed links have been found to be of considerable value in many hypermedia systems [23, 26]. Unlike the WWW, external links can have more than one target. Finally as linking is handled outside the documents, the documents can be of any format.

Anchors act as reference objects for links and are responsible for encapsulating location information for a piece of information inside a document. Locating via anchors is not tied to text, and is limited only by the methods applications provide for accessing parts of their data. Anchors can be located in segments of sound and video, areas of pictures, or rows in a relational database.

3. AUGMENTING THE WWW WITH DEXTER -BASED HYPERMEDIA

The limitations of the WWW with respect to dynamic link creation and sharing have been pointed out above. At the same time, we acknowledge the power of such a simple notion of linking. Instead of aiming at (naively) *replacing*

	Embedded address approach	Dexter-based approach
Storage of links	jump addresses inside content	link objects in separate database
Openness with respect to linking	closed : requires special content format, e.g. HTML, VRML	open: no requirement on content formats – applications' own formats can be linked
Media support	links are mostly supported from text based data	anchors may reference segments in any data-type, e.g. video
Maps of link structures	difficult (often impossible) to see who is referencing a specific node	link relations can be inspected and maps generated
Distribution	simple to distribute – only content has to be distributed	more complicated to distribute – also links and anchors
Collaboration	collaborative manipulation of link network is difficult	collaborative manipulation of links is easier – requires no write permission to content

Table 1: Comparison of the embedded address and Dexter-based approaches

the WWW with a DHM Framework based system[1], we advocate *augmenting* the WWW with DHM hypermedia services. This can add value to the WWW, e.g. for distributed work groups sharing a body of web documents. Users working with DHM/WWW augmented pages in a browser can:

- create links to and from parts of WWW documents without having write access to them;
- follow both ordinary WWW links and the DHM based links imposed on the pages;
- group WWW documents by mean of composites such as collections and guided tours;
- obtain CSCW support (e.g. lock exchange and awareness notifications) on pages that are shared by a group of users;

To provide DHM based hypermedia support for the WWW, only modest extensions to the framework classes are needed. The DHM Framework is sufficiently general to model the WWW and a WWW component with tailored anchors and LocSpecs (see [14] for a discussion on LocSpecs). The main challenge here is to integrate the architectures of the two system concepts.

This architectural integration involves some of the same steps described in [12] on tailorability. The DHM support needs to be integrated in the Application layer to provide a user interface for the extended functionality and in the Communication layer to provide HTTP/CGI based communication between the clients and the Hypermedia Service. Moreover the Runtime and Storage layer classes should be specialized to model Web documents. The architectural integration of a DHM hypermedia service with the WWW is depicted in Figure 2.

In Figure 2, the small boxes attached to the browsers represent a small extension to the users' favorite WWW browser. In particular, the browser's user interface is extended to support communication with the DHM service. The browser extension can be connected to one or more Hypermedia Database Servers that all impose link and composite structures to the Web pages being visited with the extended browser. The browsers may still communicate directly with Web servers without going through the extension module, however, to gain access to the external links available in the document, the extension module needs to be activated on the document.

At least two systems already provide a similar external hypermedia service. The Hyper-G system uses a special browser called Harmony, which handles both HTML rendering and the user interface/communication support for the extended service, i.e. it replaces the user's ordinary WWW browser [1]. Harmony supports the manipulation of links and collections, objects that are stored in an object-oriented database separate from the base WWW documents. The second system is Microcosm's DLS [6] which is based on a true augmentation of the user's WWW browser by means of a "GumShu". For each platform the GumShu attaches menus to the title bar of the WWW browser window. Via these menus, users connect to Microcosm link bases on the net and use those to follow and create links that again are stored apart from the base WWW documents.

Both Hyper-G and the Microcosm DLS represent new steps toward providing value-adding hypermedia services to WWW documents. Our development based on the DHM framework takes further steps in this direction. However, both systems have limitations which we believe it is

[1] Different aspects of the Devise Hypermedia Framework are described in [12-14]

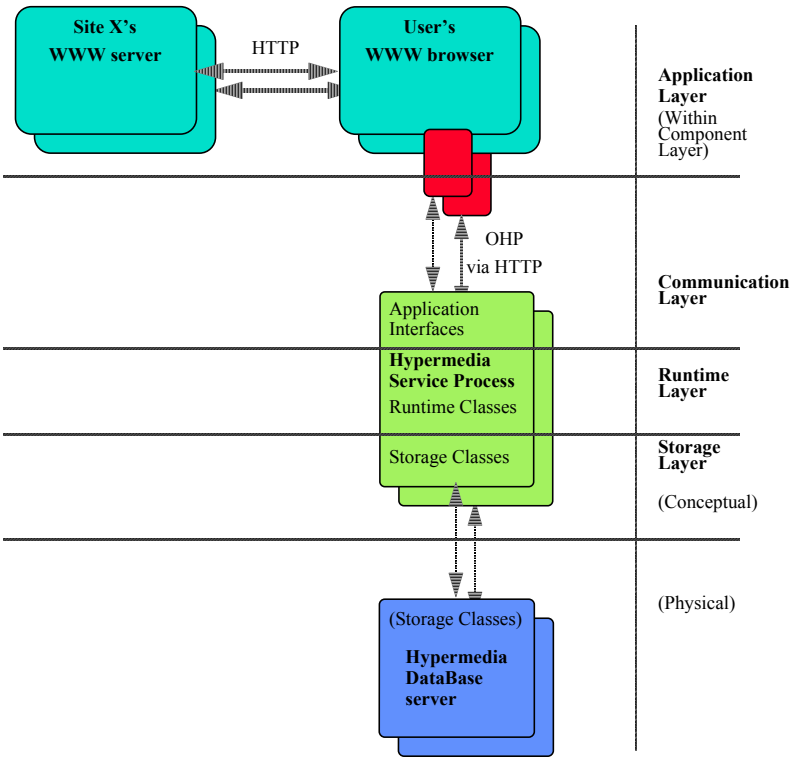

Figure 2: DHM architecture for augmenting the WWW with an open hypermedia service. Web browsers and documents are integrated as part of the Application layer of the architecture.

possible to overcome. Hyper-G requires a special browser (and currently also a special text document format, HTF) in order for the user to get the full benefits of the system, including the ability to create links and collections. Microcosm has a slightly different limitation: building a GumShu requires low level patching of the browser in the window system of the given platform.

The latest batch of WWW browsers from Netscape and Microsoft[2] have new functionality, such as Java applets and scripting, that can be utilized to integrate the link database and the WWW within the browser. In the next section, we outline two strategies for making the functionality of the DHM Framework available through WWW browsers.

3.1 Implementing DHM/WWW: challenges and choices

When we decided to augment the Web with an external link database we were faced with two major client-side challenges:

1. how to present the new links as natural as possible, and
2. how to maintain control over the document (and thus link) presentation.

We decided early on that the only acceptable solution to the first problem was to make the links resemble ordinary links in the WWW document. This required access to the document before it reached the browser, so that we could insert our own links. In addition, we needed to maintain control over the displayed document, so that we could continue to insert links from the DHM link server. Our two solutions to these challenges derive from the different approaches chosen by the Navigator and the Explorer.

Navigator offers in its present form the powerful combination of (Java) applets,[3] Javascript, and plug-ins. Applets are small programs running in a restricted environment, and are, at least in theory, platform neutral. [4] The Java language is supported by a host of powerful and increasingly stable class libraries, making development easy. Javascript programs are interpreted in the HTML

[2] As of this writing, Netscape Navigator 3.0 and Microsoft Internet Explorer 3.0 (henceforth Navigator and Explorer).

[3] As applets are compiled to run on a virtual machine, there is no need for applets be written in Java, though most still are.

[4] In practice, they can be both platform and (especially) browser dependent.

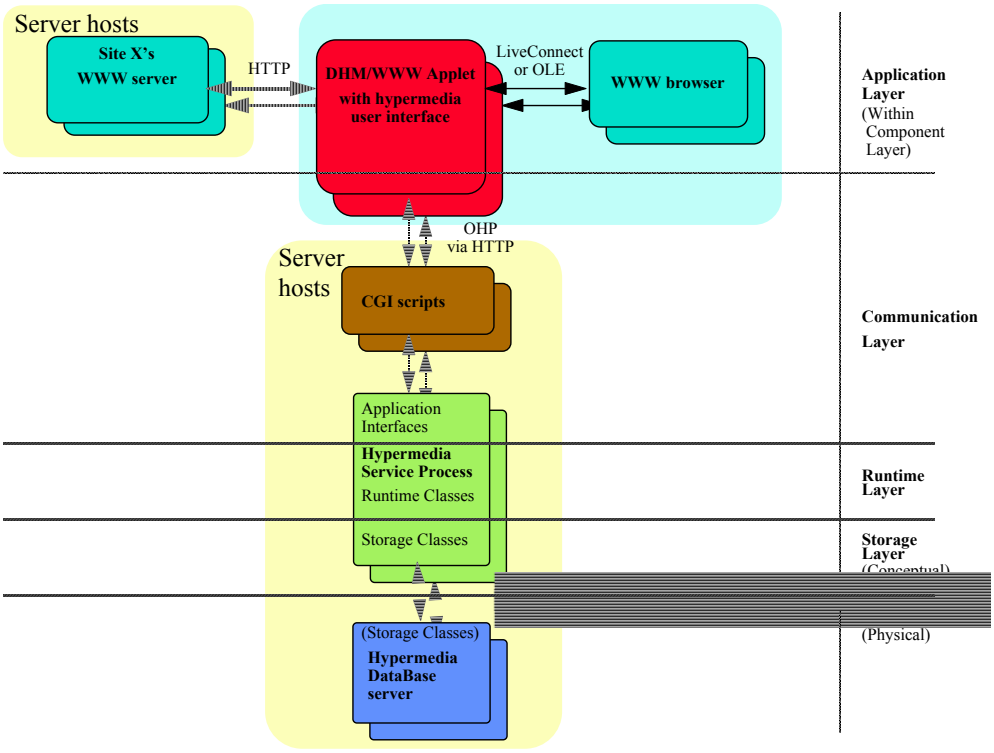

Figure 3: DHM based Hypermedia service to the Web by means of a Java or ActiveX applet and CGI scripts. Note that all the DHM modules from the Communication level and below run on the server host; only the modules in the Application Layer run on the client workstation.

document, and as such can be used for HTML document creation. Javascript has suffered from continuous development, but is now fairly stable. Plug-ins, on the other hand, are platform-dependent programs (so-called native programs), designed primarily as contents handlers (of video, sound, etc.). Netscape implements the SDI (Software Development Interface) API [25], as defined by Spyglass, Inc., for remote control of the browser.

While announcements have been made for Unix versions, the Explorer is as of yet bound to the Windows and Macintosh platform. This is a far cry from the 12 platforms supported by the Navigator, but the Explorer has the advantage of a tighter integration with its environment on Windows. In addition to applets and scripting (in Explorer known as VBScript and JScript), Explorer supports ActiveX, the latest incarnation of Microsoft's OLE technology. Because Explorer conforms to the ActiveX architecture, it can easily be integrated within other Windows programs, just as it easily integrates other OLE programs, such as Microsoft Word or Excel. In its ActiveX interface Explorer implements a rich set of properties, methods and events that can be used to control the browser from other applications.

3.2 DHM/WWW: a platform-independent solution

To provide a true platform-independent extension to the users' browser, we used the Java language[5] to write a general applet that handles browser integration and communication with the Hypermedia Service. This approach resulted in the architecture shown in Figure 3.

In the Web architecture it is a natural choice to use HTTP and CGI scripts to communicate between the browser and the Hypermedia Service Process (HSP). Thus the HSP runs on a server host rather than on the users' workstations as was proposed for environments with a common file system (see [11]). This implies an increased work load on the server hosts, but the HSP and the Hypermedia DataBase (HDB) are still separate processes that can be distributed across multiple hosts if performance becomes an issue.

3.2.1 Application layer

At the Application layer, the browser extension can be developed as a platform independent DHM/WWW Java applet. The applet provides a user interface for the Hypermedia Service functionality and supports the DHM OHP protocol [10] in its communication with the

[5] (http://java.sun.com/)

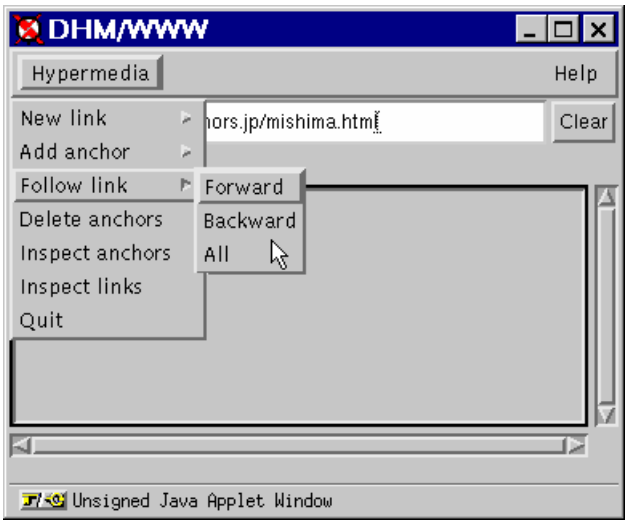

Figure 4: The DHM/WWW applet

Hypermedia Service. The applet also handles the presentation of links in the HTML documents.

The applet (Figure 4) is composed of several threaded components, linked together by a stream of data originating from the web server and the Hypermedia Service and ending up in the Navigator frame. Java threads are employed to maximize parallelism and performance. The data streaming is implemented so that new components can be added seamlessly as needed. [6]

The document presented to the user consists of two frames, one hidden containing the applet and supporting Javascript functions, and one visible containing the user's document.

The left side of Figure 5 shows the flow of the HTML document through the applet from the web server to the Netscape frame. On the right side is an example of the transformation of a piece of HTML code. In this case a line in the original document mentions the Japanese author Kenzaburo Oe.

The URL of the original document is sent to the DHM server, which returns an anchor encoded as a LocSpec. The LocSpec is stored by the applet, and a link containing a reference to the LocSpec is inserted in the document, as indicated in the LocSpec. This link is later encapsulated in Javascript, so that the Navigator always calls back to the applet when following links. Finally the line is handed over to a Javascript procedure (printFrame) that prints the line in a Netscape frame.

The URL of the desired document is either input directly in the applet window or provided indirectly by clicking on a link in the Navigator frame. The URL is checked, and if valid, turned over to the first part of the stream shown above. The task of actually retrieving the document is easily handled by standard Java library classes.

Communication with the DHM server is carried out using CGI scripts. The applet constructs a query containing an opcode and a LocSpec (consisting of the URL, position, offset, time stamp, etc.), as exemplified in Table 2, and uses the post method to send the request to the server. The server returns a stream consisting of LocSpecs encoding the locations of the anchors as shown in Table 3. The applet decodes this information and makes it available to the component inserting links in the HTML stream. An exception is raised if the time stamp of the anchors are earlier than the time stamp of the document, alerting the user to the possibility of corrupted links.

Based on the HTML stream and the anchor information, links are inserted into the stream. Currently, the algorithm uses simple position offsets, but we hope soon to have developed a more sophisticated technique involving the contexts of link anchors.

Downstream from link insertion is the insertion of Javascript code. In order to maintain control over the displayed documents, the applet must be used on all subsequent follow-links. To ensure this, Javascript code is inserted in every link, including the ordinary existing URLs, so that clicking on a link calls the applet with the appropriate parameters, rather than just causing the browser to retrieve a new document. If a link has more than one destination, the user is queried as to which one should be opened.

In contrast to Java applets, Javascript programs are allowed to write to a HTML document. In order to present the document to the user, calls to Javascript functions are generated and presented to the browser through LiveConnect.[7] Once the document has been generated, the user can click on any link, causing the applet to repeat the procedure outlined above. A user can however break out of this cycle by manually entering a URL or by selecting a bookmarked URL. As there is no Java equivalent of the BeforeNavigate event (see below), this cannot be helped.

[6] Currently, links are inserted in the document stream based on position and offset in the original HTML document. Because HTML documents are apt to change, this component will later be extended to recognize contexts of link anchors. This extension will not affect the other components.

[7] LiveConnect was introduced by Netscape in order to facilitate communication between JavaScript programs, Java applets and plug-ins. JavaScript objects and methods can be manipulated by Java applets and vice versa, and both can control appropriate plug-ins.

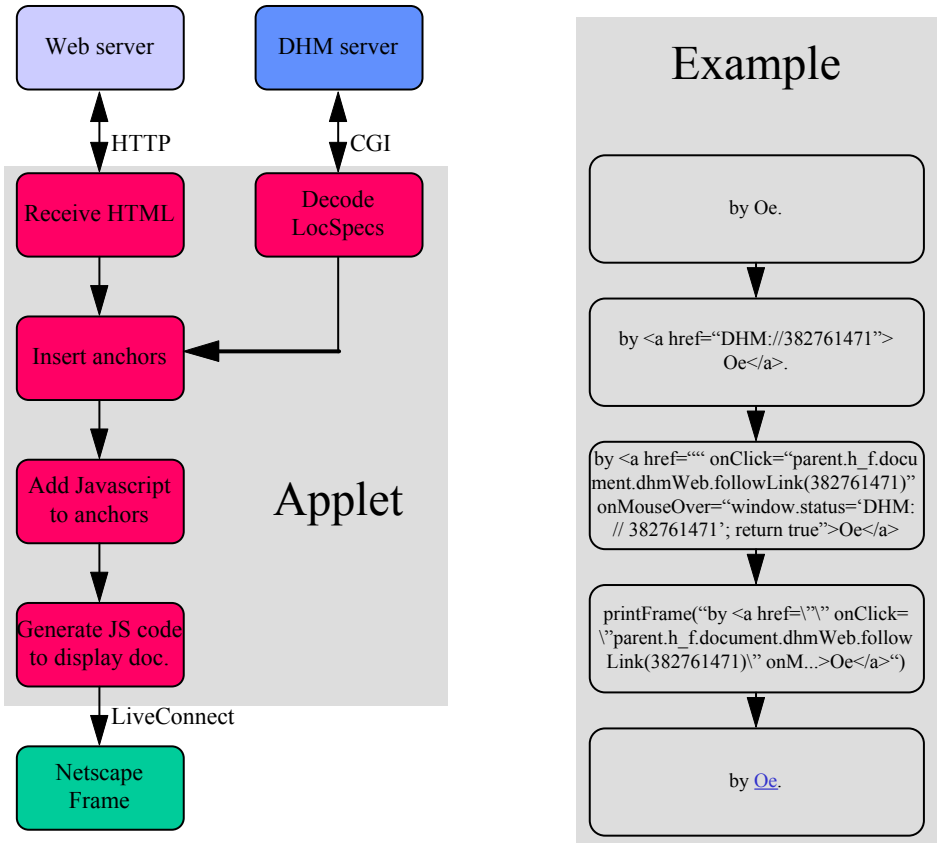

Figure 5: Structure of the DHM/WWW applet

3.2.2 Limitations

The DHM/WWW integration applet was designed to be platform independent, but unfortunately not browser independent; as LiveConnect, a necessary part of the current implementation, is pure Navigator technology. The promise of Java Beans [22] and the evolution of signed applets will hopefully provide the Java language with tools versatile enough to make vendor specific solutions obsolete. Furthermore, text selection in a browser window cannot be detected by an applet, making it impossible to support more elegant link creation.

The performance issue with regard to applets should not be forgotten – the response time of the DHM/WWW applet does not match normal Navigator use. As the 'Just In Time' compiler technology used by Navigator is still fairly new, we expect performance to improve as the compiler becomes more advanced. The current version of the applet uses CGI to communicate with the DHM server; future versions will probably employ direct socket communication to eliminate the overhead of CGI.

```
http://www.daimi.aau.dk/cgidhm/sendhm?QUERY
=webBrowser,0202,http://www.daimi.aau.dk/~b
ouvin/favoritebooks.html,1,("http://www.dai
mi.aau.dk/~bouvin/favoritebooks.html""Oe""A
nother good read is A Personal Matter by
Oe""374""2""853100323")
```

Table 2: Example of an invocation of the FollowLink operation using a CGI script

3.2.3 Communication layer

At the Communication layer, a CGI script is used to implement the hypermedia service part of the DHM OHP protocol. This CGI script is fairly simple; it decodes the query and starts a communication with the HSP, which returns a response to the CGI script to send back to the applet before the script is terminated. The CGI script and the DHM HSP use simple socket communication.

Table 2 illustrates what is sent from the DHM/WWW applet when invoking a follow link operation on an anchor in a document named http://www.daimi.aau.dk/~bouvin/-favoritebooks.html. The parameters involves a LocSpec, which consists in this case of the URL of the document, the selection and its context, the position and offset of the selection and the last modification date of the document. The date is sent, so that an exception can be raised and appropriate steps taken, if the document should be newer than the date in the LocSpec.

Finally, the sendhm CGI script returns the LocSpecs matching the anchor (see Table 3), allowing the DHM/WWW to retrieve the WWW document and present the destination to the user. If the follow link operation results in more than one destination document (as in this case), all links are returned to allow the user to select a destination. Again the modification date registered by the Hypermedia service is included in order to enable the applet to raise a warning if the document has changed since the anchors were updated.

3.2.4 Runtime and Storage layers

The Runtime and Storage layers require only minimal extensions including specializations of Component, Instantiation, and Presentation classses similar to that described in [12]. Moreover, the Anchors are designed in accordance with the unifying concepts, refSpecs and LocSpecs, proposed in [14]. In this case the LocSpecs contain redundant information in the form of positions for the text string to be located. This enables detection and possibly repair of the LocSpecs if the documents change after the link is established, as discussed in [7, 19].

3.3 DHM/WWW: platform-dependent solutions

We now describe a platform-dependent version of the above applet. Platform dependency is usually not desirable, but in this case, the advantages of developing to a specific platform may be worth the tradeoff. Apart from the differences in the Application layer, the platform dependent solution still assumes the rest of the model, as shown in Figure 3, from the Communication layer and down.

3.3.1 Using Navigator

We have investigated two approaches to this solution, one using plug-ins and one using the SDI (Software Development Interface) API [25], as defined by Spyglass, Inc..

The Plug-in approach

With the introduction of LiveConnect in Navigator 3.0, it is now feasible to use plug-ins for document retrieval and processing. Because plug-ins currently have far wider privileges than applets, they are better able to support the necessary network communication. (Plug-ins can connect to more than one server and, running natively, are ostensibly faster than applets.) Furthermore, plug-ins can stream data from and to Navigator; a plug-in may request a HTML document and (after processing it) send it back to the browser for rendering. Javascript patching is still needed to maintain control, but this integrates well with the plug-in since plug-ins can be controlled from Javascript. In this context the plug-in takes on the role of the DHM/WWW applet as shown in Figure 4.

The SDI (Software Development Interface) approach

```
(1,2,(“http://www.authors.jp/oe.html””Oe
””Kenzaburo Oe””70””2” ”850650404”),-
(“http://-
www.daimi.aau.dk/japan.html””Kenzaburo
Oe” ”Yukio Mishima, Kenzaburo Oe and
Musashi Miyamoto are my””230”
”12””852100133”))
```

Table 3: Example response generated by a FollowLink operation

Figure 6: DHM menu added to Navigator

Apart from the WWW augmentation we have also integrated Navigator into a local use context (with a local DHM server). Figure 6 shows how Navigator can be augmented with a DHM menu. The menu allows the user to create new DHM links to URLs, add URLs as anchors for existing DHM links and follow DHM links.

E.g. a user may create a link from say, Microsoft Word to a URL; following the link causes Navigator to load the URL. Likewise, a link can be followed from Navigator to another document. The user can use 'Show Connections' to see the available links from the WWW document to other applications' documents.

The implementation of the DHM extension consists of three parts:

1. attaching a menu to the menu bar in Navigator,
2. catching the events generated when menu items in the DHM menu are selected by the user, and
3. fetching and presenting URLs.

Attaching the DHM menu to the menu bar in Navigator is done using standard window and menu functions defined in the Win32 SDK. Catching the menu events from the DHM menu is implemented using hooks. Fetching and presenting URLs are accomplished through Netscape's version of the SDI (Software Development Interface) API [25]. This approach could potentially replace the DHM/WWW applet, but would then only work for Windows.

3.3.2 Using Explorer

As the OLE Automation interface exposed by Explorer is richer than the SDI API found in Navigator, more control is possible (see [21] for a description of the interface). There are two obvious approaches: a DHM/WWW Windows program can either embed the WebBrowser ActiveX control, or it can use the InternetExplorer OLE Automation object to control the Microsoft Internet Explorer application.

Essential to both approaches is that the DHM/WWW program can be notified, via the BeforeNavigate event, whenever the WebBrowser control/ InternetExplorer is about to navigate to a different URL[8], handle the retrieval of the WWW document, and hand the modified document to the WebBrowser control/InternetExplorer for rendering. The existence of the BeforeNavigate event eliminates the need to patch links with Javascript and ensures that control is *always* held by the DHM/WWW program. This total control is not possible with a Javascript solution, as the user is always free to enter an URL or to use bookmarks, operations which are not caught by the Javascript program.

3.3.3 Limitations

The main limitations of these two solutions are the platform dependency and the lack of support for text selection. The Navigator plug-in should be fairly easy to port to Macintosh and Unix; it remains to be seen how well Microsoft will be able to port ActiveX to other platforms.

4. TOWARDS SUPPORT FOR COOPERATIVE WORK ON WWW MATERIALS

One of the goals of introducing DHM-based hypermedia in a WWW context is to support better ways for workgroups to cooperate on shared materials on the WWW [11]. In this section we describe how the DHM extension to the Web described above can be extended to support different aspects of cooperation on Web materials. We assume in the following that the workgroup members are registered as users with the shared DHM/WWW server and that they use an HTML editor which is integrated with the DHM/WWW applet.

---

[8] Which may happen as a result of external automation, internal automation from a script, or the user clicking a link or typing in the address bar.

4.1 Multiple hypermedia structures for the same body of materials

The approach illustrated in Section 3 immediately provides support for imposing several different link and composite structures on the same body of Web documents. At the same time it allows anyone to make links *into* and *from* documents. This is one step towards support for cooperative working and it is reminiscent of Intermedia's multiple "webs" over a single corpus of documents [15]. Like Intermedia, we support "annotation access", the ability for non-document owners to create links. But as Halasz proposed in his "Seven issues paper" [17], full support for cooperative authoring on the Web requires both flexible transaction mechanisms and awareness notifications.

4.2 Locking and long term transactions

Assume that a group of people wants to establish a working environment in which they can cooperatively use and edit a set of Web pages residing on one or more servers. They could then use an DHM-based hypermedia service to coordinate their work on individual pages. A Java-based HTML editor could be tailored to request a write lock on the page to be edited from the hypermedia service before allowing the editor to change the document. If the write lock is already taken by another user, the request fails to assign a write lock, and the requesting user is told who possesses the write-lock. This way asynchronous cooperation based on turn taking and exchange of locks on WWW documents can be supported via an external hypermedia service.

4.3 Awareness notifications

Awareness notifications can be supported for Web documents via a DHM-based hypermedia service. The client DHM/WWW applet described above needs to be extended with mechanisms to subscribe to notifications on specific events. It also needs a thread that listens to notifications sent from the connected DHM server. This would allow a user who is unsuccessful in obtaining a write lock on a given page managed via the DHM server, to subscribe to a lock release for that particular page. Whenever the write lock is released, the DHM/WWW applet receives a notification that can be displayed to the user.

The cooperation support for Web documents is weaker than the support that can be provided for fully DHM-based hypermedia structures. For instance, awareness notifications based on the embedded HTML based links cannot be distinguished from changes to contents, since the links are not represented as objects in the Hypermedia Database. But the hypermedia service can provide full notification support for the link and composite structures created "on top" of the Web pages and stored in the hypermedia database.

5. THE VOLUME OF HYPERMEDIA STRUCTURES COMPARED TO CONTENTS

In an open hypermedia system, information contents are usually stored apart from hypermedia structures although some system architectures and databases allow mixed contents and structure storage [11]. In architectures where contents is kept completely separate from structure, it is usually the case that the size of hypermedia structuring data is much smaller than the contents. To illustrate with an extreme example, it is possible to create thousands of links in a large body of digitized video that still only occupy a fraction of the volume of the contents data. Thus providing a large body of video together with hypermedia structures having numerous links does not turn the hypermedia database into a bottleneck. The bottleneck continues to be the transfer and playback of the video. In less extreme cases hypermedia structures still only add minimal overhead to the resources required to transfer information over the net.

Implemented in an object-oriented language and using an object-oriented database, DHM Components typically require a few hundred bytes depending on the number of Anchors, RefSpecs and user-defined attributes they include.[9] For instance, establishing a link between anchors in two different Web documents implies the creation of three DHM Components (two atomic components and a link component), which in all requires less than half a Kbyte storage. Adding more links to the same documents adds only link components and anchors, not new atomic components. For example, five new links between anchors in two given documents may add as little as a single Kbyte storage in overhead. Compared to typical document sizes on the Web this is barely noticeable.

In short, the main challenge in providing DHM-based hypermedia for documents residing in a distributed environment is not storage and transport overhead. The more serious challenges are to keep documents in the distributed environment in synch with the hypermedia databases, and to provide ready access to the DHM information when a document with links is accessed.

6. CONCLUSION

In providing Dexter-based hypermedia support for linking and collaboration on the WWW, our initial ambition was to create a platform-independent DHM/WWW applet that would run on all Navigator supported platforms. We have encountered two main obstacles to this goal: network restrictions and inadequate communication between applets and browsers (both Netscape and Explorer).

An applet is not allowed to connect to other than the originating server. There are good reasons not to give an

[9] This is a pessimistic estimate of the overhead, as specific databases may be able to compress the data even more.

applet free access to the rest of the Internet, but the limitations seem unnecessarily harsh. A more subtle approach allows some granularity regarding which servers the applet may connect to. Alternatively, signed applets can be granted greater privileges, a concept similar to ActiveX.

We have found the integration between applets and browsers to be inadequate. There is no Java equivalent of Explorer's BeforeNavigate, and events such as text selection in a WWW document are not accessible to an applet in either browser. Effectively, this makes it impossible to implement elegant link creation in a platform independent way. Applets are not allowed to write directly to an HTML document – the existing solution in Navigator of going through Javascript is unsatisfactory and clumsy. Rather than having to encapsulate every link in Javascript code (a hack more than a satisfying solution), an event should be triggered by the browser, so that the actual download of data could be handled by a Java applet. Applets should also be allowed to register themselves as content handlers with the browser, thus eliminating the need for plug-ins.

Our alternative to the unsatisfactory platform independent solution has been to use the two browsers' proprietary platform dependent architectures: plug-ins and ActiveX. Despite the relative ease of porting plug-ins and Microsoft's claims that ActiveX will spread to other platforms, this still runs counter to the original vision of the World Wide Web as *one* standard for accessing information. Not only are users expected to use the "right" browser, they must also run it on the "right" machine. Even then the integration between browser and program is insufficient. This fragmentation of the WWW is highly undesirable, and it remains to be seen whether Java can mature fast enough to provide an adequate alternative to the browser/platform dependent approach.

Since the technology is still in its infancy, augmenting the WWW with services that provide hypermedia structures outside the document contents is a challenge, but assuming that the Web browsers become more open as suggested in this paper the approach holds great promise. Perhaps, as envisioned by the hypermedia pioneers, the World Wide Web will one day become an environment supporting dynamic link creation and collaboration world wide.

ACKNOWLEDGMENT

This work is supported by the Danish National Science Research Foundation (SNF) grant number 9502631. We thank a number of anonymous reviewers for their helpful comments. Finally we thank Randy Trigg for his comments and suggestions for language improvement.

REFERENCES

1. Andrews, K., F. Kappe, and H. Maurer, *Serving Information to the Web with Hyper-G.* Computer Networks and ISDN Systems, 1995. 27(6).
2. Baldazo, R. and S.J. Vaughan-Nichols, *Web Publishing Made Easier.* Byte, 1995. December: p. 170 pp.
3. Berners-Lee, T., *et al. World-Wide Web: An Information Infrastructure for High-Energy Physics*. in *Software Engineering, AI and Expert Systems for High Energy and Nuclear Physics*. 1992. La Londe-les-Maures, France.
4. Berners-Lee, T., *et al.*, *World-Wide Web: The Information Universe.* Electronic Networking: Research, Applications and Policy, 1992. 1(2).
5. Bush, V., *As We May Think.* The Atlantic Monthly, 1945. August.
6. Carr, L., *et al. The Distributed Link Service: A Tool for Publishers, Authors and Readers*. in *Fourth International World Wide Web Conference : ``The Web Revolution''*. 1995. Boston, Massachusetts, USA.
7. Davis, H.C., S. Knight, and W. Hall. *Light Hypermedia Link Services: A Study of Third Party Integration*. in *European Congerence on Hypermedia Technology (ECHT '94)*. 1994. Edinburgh, UK.: ACM.
8. Engelbart, D. *Authorship provisions in Augment*. in *Proc. IEEE Compcon Conference*. 1984. San Francisco, California: IEEE.
9. Goodman, D., *The Complete HyperCard Handbook*. 1987, New York: Bantam Books.
10. Grønbæk, K. *Object oriented design of the Distributed Hypermedia Toolkit (DHTK)*. EuroCODE report: CODE-AU-94-5, Computer Science Department, Aarhus University, Denmark, 1994.
11. Grønbæk, K., *et al.*, *Cooperative Hypermedia Systems: A Dexter-Based Architecture.* Communications of the ACM, 1994. 37(2): p. 64-75.
12. Grønbæk, K. and J. Malhotra. *Building Tailorable Hypermedia Systems: the embedded-interpreter approach*. in *ACM conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA '94)*. 1994. Portland, Oregon, US, 23-27 October, 1994: ACM.
13. Grønbæk, K. and R.H. Trigg, *Design issues for a Dexter-based hypermedia system.* Communications of the ACM, 1994. 37(2): p. 40-49.
14. Grønbæk, K. and R.H. Trigg. *Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects*. in *HYPERTEXT '96 – Seventh ACM Conference on Hypertext*. 1996. Washington DC, USA, March 16-20.
15. Haan, B.J., *et al.*, *IRIS Hypermedia Services.* Communications of the ACM, 1992. 35(1): p. 36-51.
16. Halasz, F. and M. Schwartz, *The Dexter Hypertext Reference Model.* Communications of the ACM, 1994. 37(2): p. 30-39.
17. Halasz, F.G., *Reflections on NoteCards: Seven issues for the next generation of hypermedia systems.* Communications of the ACM, 1988. 31(7): p. 836 - 852.
18. Halasz, F.G., T.P. Moran, and R.H. Trigg, *NoteCards in a Nutshell*, in *Proceedings of ACM CHI+GI'87*

Conference on Human Factors in Computing Systems and Graphics Interface. 1987. p. 45-52.
19. Hall, W., H. Davis, and G. Hutchings, *Rethinking Hypermedia: The Microcosm Approach*. 1996, Boston: Kluwer Academic Publishers.
20. Meyrowitz, N. *Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework*. in *OOPSLA '86 Conference*. 1986.
21. Microsoft. *The Web Browser Control and the Internet Explorer Object*. WWW: http://www.microsoft.com/intdev/sdk/docs/iexplore/, Microsoft,
22. Microsystems, S. *Java' Beans: A Component Architecture for Java*. WWW: http://splash.javasoft.com/beans/WhitePaper.html, Sun Microsystems,
23. Nanard, J. and M. Nanard, *Using Structured Types to Incorporate Knowledge in Hypertext*, in *Proceedings of ACM Hypertext'91*. 1991. p. 329-343.
24. Nelson, T.H., *Computer Lib/Dream Machines*. 1974: Mindful Press.
25. Spyglass. *"Software Development Interface" API*. WWW: http://www.spyglass.com:4040/newtechnology/integration/iapi.htm, Spyglass, Inc.,
26. Trigg, R.H. *A Network-Based Approach to Text Handling for the Online Scientific Community*. Ph.D. Thesis: TR-1346, University of Maryland, 1983.