Unifying Strategies for Web Augmentation

Niels Olof Bouvin

Aarhus University,
Department of Computer Science,
Aabogade 34A, DK8200 Aarhus N, Denmark

bouvin@daimi.aau.dk

ABSTRACT

Since the beginning of the WWW, tools have been developed to augment the functionality of the Web. This paper provides an investigation of hypermedia tools and systems integrating the World Wide Web with focus on functionality and the techniques used to achieve this functionality. Similarities are found and based on this, a new framework, the Arakne framework, for developing and thinking about Web augmentation is presented. The Arakne framework is flexible and supports most kinds of Web augmentation. Finally an implementation of the Arakne framework is described and discussed.

KEYWORDS

Web Integration, Open Hypermedia Systems, Open Hypermedia Protocol, Collaboration on the Web, Unifying interfaces, Common Reference Architecture for open hypermedia systems, Java

INTRODUCTION

The World Wide Web has in an amazing short time span become the hitherto largest hypertext and is pervasive in everyday life as few things before. This success has in part been attributed to the simple architecture behind the Web: A stateless file transfer protocol (HTTP), an universal Internet naming scheme (URL) and an easily understood document format (HTML). These standards have (largely) been adhered to, and this has enabled the creation of a large amount of software, be it Web servers or browsers to work together to the benefit of all.

The success of this simple and hugely scaleable architecture has come, from the standpoint of the hypermedia research community, at some costs, as the Web itself is lacking in the ways of more advanced (but ironically often far older) hypermedia systems. Web links are unidirectional jump links, embedded in the HTML documents, severely diminishing the flexibility of use. There is yet no widespread support for collaborative authoring, though an initiative such as WebDAV [13]) holds great promise.

An important development in the hypermedia community in the nineties has been the focus on and development of open hypermedia systems integrating third-party applications. Significant work has been done in systems such as MicroCosm [9][20], HyperDisco [44][45], HOSS [32], DHM [14][15], and Chimera [1]. These systems have all addressed the problem of augmenting third-party applications, and the lessons learned are important guidelines for future work of Web augmentation. Based on these experiences,, taxonomies and studies have been developed, by Whitehead [43], Grønbæk & Wiil [17], and Wiil & Østerbye [46][48], to help researchers and developers discuss and reason about the open hypermedia field and how to utilise hypermedia in third-party applications. Given this experience of integration it should come as no surprise that the open hypermedia community was quick to develop open hypermedia Web integrations, e.g. DLS [7], DHM/WWW [16], and Chimera [2]. This article will investigate how they and others achieved their goals of Web augmentation.

Adhering to standards has a large part of the success of the Web and the very size of the Web has enormous inertia, so an attempt to replace the Web with something perhaps more advanced in certain aspects is, if not doomed, then up against tremendous odds. Clearly this is not the way to improve the Web. An approach that retains the benefits of the Web as well as adding new desired functionality is the development of systems that operate within Web standards, be it HTTP, HTML, browsers, or servers. This is in spirit with Meyrowitz' call for hypermedia integration in third-party applications [27], and the amount and quality of work done using this approach would suggest that it is at the very least possible.

AN OVERVIEW OF WEB AUGMENTATION STRATEGIES

As the scope of this article is to study the techniques used in Web augmentation tools, an overview of augmentation approaches is in order. This overview will work at two levels of abstraction: first a broad grouping based on the functionality of the tools and later a more specific characterisation of the individual tool. This characterisation will be based on the chosen approach to common problems, such as storage, Web browser integration, level of support for collaboration, and so forth. The former level of abstraction will let us discuss and compare related tools, while the latter will allow us to recognise reoccurring themes in the approaches taken.

A tool shall be considered a Web hypermedia augmentation tool, if it through integration with a Web browser, a HTTP proxy or a Web server adds content or controls not contained within the Web pages themselves to the effect of allowing structure to be added to the Web page directly or indirectly, or to navigate such structure. The purpose of such a tool is help users organise, associate, or structure information found on the Web. This activity may be done by a single user or in collaboration with others.

Following this definition, the purpose of the Web augmentations reviewed for this article is to help users structuralise their Web work. Either by adding structure and displaying it, or by extracting structure already present and making it more visible. The displayed structure may be malleable, allowing the user to modify it. The Web augmentation tools reviewed for this article have been divided into four categories:

- ♦ Annotations/Discussion support
- ♦ Link creation and transversal
- ♦ Guided tours
- ♦ Structuring/Spatial

We claim no universality to this categorisation, but have found it handy when discussing the Web augmentation tools and their use.

A Web augmentation tool can be classified by the schema summarised in Table 1. Thus a tool can either be a part of a Web browser; it can be an closely integrated tool loaded on runtime; it can reside temporarily within the browser as an applet or an ActiveX control; it can be an application running the user's computer; or it can be located elsewhere (in that case more often than not as a HTTP server or proxy).

The Web augmentation tool can either have no storage requirements; it can store it data locally on the host computer, or remotely on a server.

Many of these Web augmentation tools modify Web pages, either to insert interfaces of their own or to add structure (e.g. links) to the Web page. This modification can take place at the Web server (perhaps a Web server translating a proprietary data format into HTML), by calling CGI-scripts that return modified pages, by using a special proxy, or by modifying the Web pages as or after they are displayed in the Web browser.

As for the collaborative aspect, a Web augmentation tool can either be strictly personal, i.e. relevant to a single user only; the created data or structures can be shared (e.g. send to another user or placed on a Web site); the structures can browsed or edited by turn-taking (asynchronously); or users can collaborate through the structures in real time (synchronously).

WEB AUGMENTATION TOOLS

In this section we will describe various Web augmentation tools focusing on the elements introduced by Table 1. The scope of this article does not allow for a comprehensive study nor a general survey, so only a subset of the existing Web augmentation tools is presented.

Annotations/Discussion support

Bush envisioned marginalia in the Memex [5], and the interest in annotations and how they should be supported by hypermedia has not diminished over the years, as witnessed by the investigation done by Marshall [25].

The first widely successful Web browser, NCSA Mosaic [30], gave the user the opportunity to create annotations to Web pages. The annotations were personal and stored locally. Later, this feature fell out of favour with Web browser developers.

Recognising that annotations whilst useful for the individual are even more beneficial for a community, several collaborative annotation tools have been created. Röscheisen *et al.*[35][36] have developed ComMentor, which have used for several purposes, including content rating and annotations. The system employs a Mosaic [30] browser, modified to provide an interface to the annotation server, which consists of a collection of CGI-scripts. The user has alongside with the browser a merge library, which inserts comments and links to comments into the Web pages. Annotations are stored in sets, of which the user may activate an arbitrary number. Collaborative annotation is supported through dividing users into groups that may share sets of annotations. A set of annotations can be set to be private, available to a group, or publicly available. Annotations are displayed using in-place markers (small pictures) indicating either the nature of the annotation or the author's identity. An annotation while write-protected may also be annotated.

Method of integration
A part of the browser Browser add-ons Within the browser (plug-ins, applets and JavaScript) Without the browser, local Without the browser, remote Within the proxy Within the Web server
Location of storage
No storage Local storage Remote storage
Web page modification
No modification At the Web server CGI-scripts at a Web server At the proxy In the browser
Level of collaboration supported
Personal Shareable Asynchronous collaborative Synchronous collaborative

Table 1 - A classification scheme of Web augmentation tools

Another system is CritLink Mediator, part of CritSuite [10], which is specialised to provide support for 'critical discussions'. CritLink employs predefined typed links (support, issue, comment, query) akin to many hypermedia systems, such as IBIS [34] and TEXTNET [40]. The comments are created by a mix of CGI-scripts, Web forms, and JavaScript and stored either on a designated server or in the user's own Web space as ordinary Web pages. Links to the comments are inserted into Web pages by use of a Web server effectively acting as a proxy server. The pages are also modified to include a tool bar used for navigation, annotation, and the launch of CritMap [39], a tool that generates maps of neighbourhood Web pages. Links in pages presented by the server are modified to go through the server. As comments are Web pages in their own right, they can also be commented on. There are only one set of annotations and no notion of groups, though the individual user is identified.

Creating links

As noted in the introduction, quite a few research projects have addressed the issue of adding external structures to Web pages. An excellent investigation of the various approaches taken by the open hypermedia community can be found in [2]. There are two main approaches: links (or other kinds of structural information) are either displayed alongside the Web page or inserted into the Web page. The former case requires either a program to display this structure or a browser window where the structure has been converted to HTML. The latter case involves modifying HTML pages on the fly, which can be done at three places: at the origin, in transit or at arrival, i.e. the Web server, the HTTP proxy, or the Web browser.

Chimera [1][2] is an example of a system, where experiments with either displaying structure information in a separate program (an applet) or making the structure server accessible through HTTP have been carried out. By modifying a Web server to interpret HTTP requests as requests to a Chimera server to which the Web server is hooked up and in turn translating the Chimera structures to HTML, a user is able to browse the hypermedia structure using an ordinary Web browser. This experiment was extended upon by the creation of a Java applet capable of displaying Chimera hypermedia structures. By the combination of a special Web server, CGI-scripts and cookies, this applet was inserted into all pages displayed in the Web browser, giving the user immediate access to Chimera services.

Hyper-G [26] is a more specialised system, as it to achieve full functionality relies on a special document format (HTF – Hypertext Format), a special server, and a custom browser. It is however possible to interface to the system using an ordinary Web browser using a special WWW-gateway, that will translates HTF document and hypermedia structure to HTML. The hypermedia system offers strong support for hierarchical structures and searching, and allows users without a special browser to create links using forms. In recent versions HyperWave [22] (as the system is now known) offers an advanced interface utilising Java-applets and JavaScript inserted into Web pages by the HyperWave server.

DLS [6] (Distributed Link Service) is based on the MicroCosm hypermedia system [7][9][20]. The first DLS systems used a wrapper to attach a link service menu to a browser (this integration being dependent on whether an integration existed for the user's browser), thus creating a (in the terminology of Whitehead in [43]) shim integration with a third party application. Links were followed by selection of text and selecting 'Follow Link' in the attached menu. This would cause the wrapper to contact the link server with an URL encoding the request, resulting in a Web page of the matching links. To address the problem of having to install special software and to make links more visible, an interfaceless version was developed that used a link server proxy to insert links in Web pages as requested by the user. The user used a form to configure which link bases to use and how the link should be presented in the document (to make the distinction between links belonging in the document and inserted links clear). Due to performance issues (beyond 'conventional' links, MicroCosm offers computation intensive links, such as keyword links, person links and citation links) and copyright and authors' rights concerns about adding content to Web pages, a new design was introduced with the AgentDLS [8]. Rather than offering synchronous links (presented together and simultaneously with the document), links are now displayed in a separate window. This improves the performance of browsing considerably, as the users' primary window of interest does not have to wait for links to be resolved. The linking service thus takes on a more advisory nature. This system is implemented by using a proxy that (as seen by the Web browser) acts as a normal proxy but also sends the displayed document to a link server agent that resolves the links relevant to the document. The display of these links is handled by having the AgentDLS browser window request a page from the link server agent with regular intervals.

The Devise Hypermedia group, of which the author is a member, has also made various Web integrations with its Dexter-based hypermedia system [14][15]. The first attempt was DHM/WWW [16]. The architecture consisted of a Java-applet communicating through a CGI-script to a DHM server. When the user requested a document by typing its URL in the applet, the applet would retrieve the document while querying the DHM server for endpoints in the document. The endpoints retrieved was inserted into the Web page as it was being downloaded and displayed in a Web browser window using JavaScript. All links in the Web page were modified so that a click on a link would result in the applet being invoked, allowing it repeat the above described process. The links and endpoints from the DHM server could also be inspected and browsed within the applet. This version had several shortcomings: it was dependent on the user not using bookmarks or entering URLs in the Web browser itself, as such actions would cause the applet to be terminated as its own page would be unloaded. Furthermore it was unable to handle frames (as the loading of a new 'top' frame set would also cause the unloading of the applet) and was limited by the 'sandbox' imposed for security reasons on Java applets[1]. While supported by the DHM server, the DHM/WWW applet could only handle one context (that is one set of hypermedia structures) and had no user concept. A second version, Navette [3], was developed to address some of these issues. Navette was a signed Java applet, allowing the system to use Web pages from arbitrary Web

[1] The Java 'sandbox' security limits a Java applet in various ways. Most crucial to DHM/WWW was the restriction of network contact exclusively to the originating web server, thus making DHM/WWW unable to work with web pages from other web servers.

servers. To speed up communication with the DHM server, TCP/IP and optimistic caching of hypermedia structures (e.g. retrieving a whole context rather than only resolving one link) were used. This version also handled multiple contexts and users. The frame problem remained, and Web pages were still displayed using JavaScript, which made for noticeable degraded performance when browsing with Navette. Simultaneously with Navette, the Webvise client [18], a custom integration with the Microsoft Internet Explorer [28] was being developed. Operating as an application rather an applet removed the limitations put on DHM/WWW and Navette, and using the Microsoft Internet Explorer [28] rather than the Netscape Communicator [31] allowed Webvise to insert links after the browser had displayed the document, thus improving performance considerably. This is done through DOM [11] and the COM-interface available through the Internet Explorer. A second version of Navette has been developed addressing the problems of prior releases using the Arakne framework, which will described below.

Guided tours

Guided tours and trails have been a part of hypermedia from the very beginning [5], when Bush introduced the concept of the trail linking related documents together. Trigg did more recent groundbreaking work in [41]. Several existing systems try to exploit this idea with Web documents.

Walden’s Paths [12][37] is a system designed mainly to be used in an educational setting, where a teacher composes trails for students to follow. The teacher uses either the Path Authoring Tool (a Java application) or VIKI [38] combined with a browser as an authoring tool. Trails are stored on a Path Server, which through the use of CGI scripts acts as a proxy while modifying the pages to provide an interface to the path. The interface consists of blocks in the top and the bottom of the page. This block is a graphical representation of (a part of) the path plus additional annotations written by the path author. As all documents go through the Path Servers (links in the documents are modified to achieve this), students can go ‘off path’ and still return to the path by pressing a button in the interface block. State is communicated by adding arguments into the URL given to the Path Server’s CGI-scripts. The Walden’s Paths has been extended with regards to collaborative aspects, allowing students to author and share paths of their own. Additionally work has been done to extend upon the linear path by adding conditional branches. The logic to support this is handled by the Path Server, thus still making all functionality accessible from a standard Web browser.

Another Web-based guided tour system is Ariadne [23], which is a Java-based applet. Ariadne operates in an external window to the browser and controls the browser through JavaScript. A guided tour in Ariadne is a directed graph, as opposed to the linear (with branches) path of Walden’s Paths. The Ariadne user interface supports both browsing and editing of guided tours. The tours are stored as composites on the Dexter-based DHM [15] server. Leaving the Web pages untouched has several advantages to the Walden’s Path approach, as 1) it reduces overhead and complexity as Web documents do not have go through an extra server, and 2) entering URLs or using bookmarks does not pose a problem. On the other hand users are required to use a Java-enabled Web browser rather than any Web browser, though that currently is not a strong requirement. The Ariadne system has recently been adapted to work within the Arakne framework, which will be described in more detail below.

Structuring/Spatial

Spatial hypermedia as described by Marshall & Shipman [24] and as implemented in VIKI [37] is a new kind of hypermedia application, where link structures are no longer explicit but rather implicit based on the spatial relationship between objects. This has become a very powerful tool for organising and structuring, and few hypermedia systems are in more need of organisation and structure than the Web.

Web Squirrel [42] is a URL management system, that uses a spatial metaphor to help users organise their URLs into ‘information farms’. The user creates Neighbourhoods onto which URLs are dragged and dropped. The Neighbourhoods and the URLs are arranged spatial as the user wishes, and are analysed by software agents that can create links between URLs according to user’s rating of the Web sites and maintain link integrity. The user can create new agents using a scripting language. The information farms are stored locally, but can be distributed to other users of Web Squirrel, exported as HTML, or converted to the Hot Sauce MCF format.

Hot Sauce [21] is a spatial hypermedia plug-in created by Apple. Hot Sauce displays a zoomable 2D representation of a collection of collections and documents. This structure is stored using the XML [47] based Meta Content Format [19], a general format to describe meta content (MCF has thus much wider application than its use in Hot Sauce). Links and collections of links are arranged spatial and the user can zoom in and out, move about, and open collections within the collection. If the user double-clicks on a link, the document is retrieved in a separate window, allowing the user to continue to navigate using Hot Sauce. The Hot Sauce is a media viewer and as such retrieves its MCF file from a Web server.

SUMMARY OF STRATEGIES FOR WEB AUGMENTATION

The tools and systems described above have addressed many problems pertaining to the current Web, and have utilised a lot of different techniques to attempt to solve these problems. The Web augmentation strategies are, using the schema introduced in Table 1, summarised in Table 2. We will below outline some general trends and describe some of the aspects of writing Web augmentation tools that make developing them hard.

Some patterns become apparent. All of the tools reviewed are responsive and need to be aware of the user’s actions, be it to record the URL of the current Web page or to perform link and endpoint computations. All need to provide the user with a user interface (though it might be very discreet at most times, as in the ‘interfaceless’ DLS). Most of the tools need to store data somewhere and most choose to do this on a remote server, thus raising the need to able to communicate over network. The communication is often handled by CGI-scripts, which is problematic, as the tool only gets data when it requests it – the server cannot notify the tool of changes. Many of the tools modify Web pages, and most of the implementations of this functionality would have a hard time interoperating with each other, as they in turn would modify pages and links, quite possibly corrupting each other's data. Most of the Web page modifications are not robust to things such as frames and JavaScript, and many have a problem with forms. The tools relying on modifying link with CGI-scripts rather than using a proxy are fragile to the use of bookmarks or directly entered URLs. The tools that modify Web pages through a proxy are hard to use with other tools that rely on proxies to be informed of the user's actions, unless either of the proxies can it be modified to use the other as a proxy. Furthermore the use of proxies requires the user to modify the Web browser configuration which can be unwieldy if the user does not wish to continually use the tool relying on the proxy. These problems to which there generally are no easy solutions, make it difficult for the developer to create Web augmentation tools.

Tool	Method of Integration	Storage	Web page modification	Collaboration support
ComMentor	Part of browser, CGI-scripts	Remote	Local proxy	Asynchronous
CritLink Mediator	Within browser, JavaScript, forms; CGI-scripts	Remote	CGI-scripts	Asynchronous
Chimera	Web server	Remote	No	Asynchronous
Chimera	Within browser, applet; Web server; cookies, CGI-scripts	Remote	Web server	Asynchronous
Hyper-G	Part of browser/Within browser, forms; web-server;	Remote	Web server	Asynchronous
HyperWave	Within browser, applet, JavaScript, forms; web-server	Remote	Web server	Asynchronous
DLS	Without browser, local; within browser, forms	Remote	No	Asynchronous
DLS	Within browser, forms; proxy	Remote	Proxy	Asynchronous
AgentDLS	Within browser, separate window; proxy	Remote	No	Asynchronous
DHM/WWW	Within browser, applet, JavaScript, CGI-scripts	Remote	Web browser	Shared
Navette	Within browser, applet, JavaScript	Remote	Web browser	Asynchronous
Webvise	Without browser, local	Remote	Web browser	Asynchronous
Walden's Paths	Within browser, JavaScript, forms; CGI-scripts	Remote	CGI-scripts	Shareable
Ariadne	Within browser, applet, JavaScript	Remote	No	Asynchronous
Web Squirrel	Without browser, local	Local	No	Shareable
Hot Sauce	Within browser, plug-in	Remote	No	Shareable

Table 2 - Summary of Web augmentation strategies

TOWARDS A COMMON FRAMEWORK

We are aware of no single "silver bullet", or all encompassing solution, that will solve all the problems described above. However, the similarities between the described Web augmentation tools would suggest that it should be possible to describe and model their functionality in a common framework. This could provide workers in the field with a tool for future conceptual and practical development. Trying to create such a tool, we have come up with the Arakne framework.

The Arakne framework is a conceptual model, which has been implemented as an environment for Web augmentation tools. The implementation is just one implementation of a general framework. The practical issues raised by the summary above will be dealt in the description of the implementation.

The Arakne framework is an object-oriented, component-based, three-layered model aimed at providing Web augmentation tools a unified access to structure servers, proxies, and Web browsers. It is an instantiation of the Common Reference Architecture (CoReArc) for open hypermedia systems, as described by Grønbæk & Wiil [17]. CoReArc divides the architecture of hypermedia systems into three layers: The content layer (displaying and handling documents, displaying structure), the service layer (handling navigation, integration, collaboration etc.), and the structure layer (storage and retrieval of structure).

The Arakne framework is aimed at modelling Web augmentation tools, and the elements contained in the model should now be familiar.

A diagram of the framework can be seen in Figure 1. 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 include 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. 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.

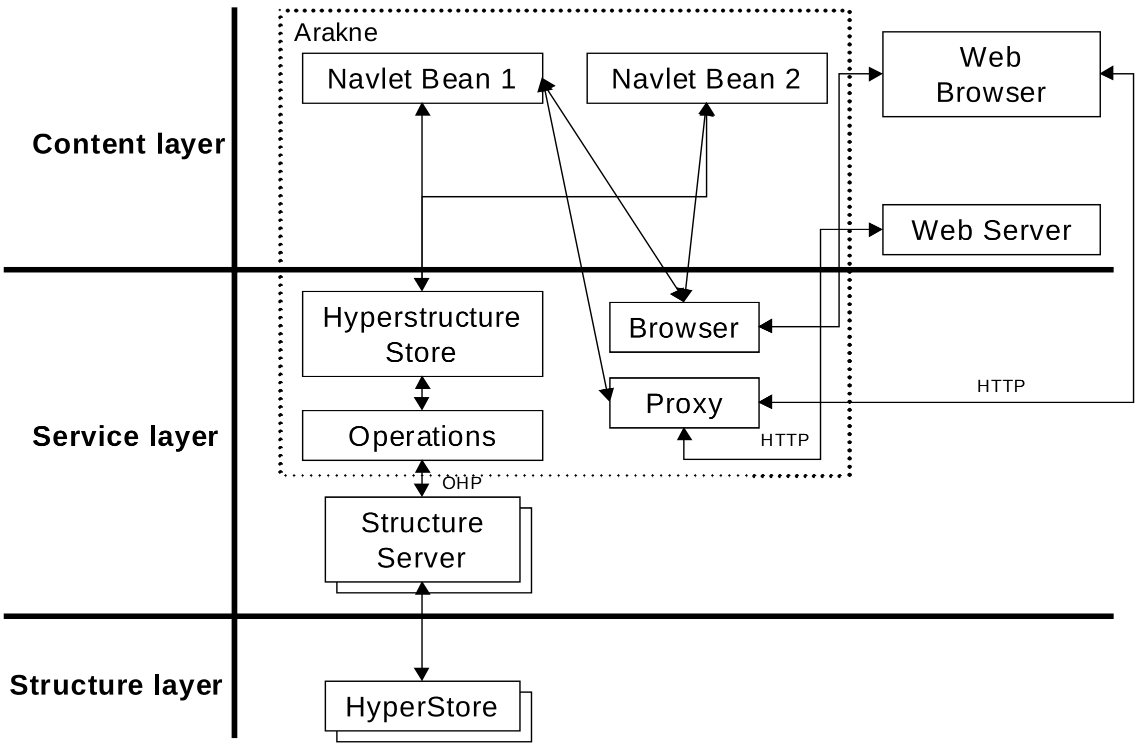

Figure 1 – The Arakne Framework

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 and the Proxy also handles this.

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 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.

The situation depicted in Figure 1 is a situation of two navlets, where Navlet Bean 1 is a link creation tool, and thus needs access to the Proxy in order to insert links into Web pages. Navlet Bean 2 is a guided tour tool and does not modify Web pages; and is not connected to the Proxy. Both however need to be able to tell and set the state of the currently displayed documents, as well as retrieving data from the structure server through the Hyperstructure Store.

Mapping Web Augmentation Tools to Arakne

If we review the models of the Web augmentation tools described in this paper, most can be mapped to the abstract Arakne framework. We will in this section argue for the general applicability of the Arakne framework in each of the general Web augmentation categories introduced earlier, and specify the mapping of one representative from each category to the Arakne framework.

Annotation tools are in their functional requirements similar to link creation tools and will dealt with as one. Both need to retrieve hypermedia structures stored on a server, which are handled by the Operations and the Hyperstructure Store components. Many of these tools need to modify Web pages in order to insert links or annotations, which is handled through the Proxy component. Support for collaboration is handled at different levels. The Hyperstructure Store component is able to handle sets of structures as well multiple users. Notifications from the structure servers are handled by the Operations component and forwarded to the Hyperstructure Store component, which notifies navlets, depending on what events they subscribe to.

The ComMentor system, described in [35], has three main elements apart from the structure servers and the Web servers; namely the merge library, the interactive renderer, and the user context control application. The merge library handles the tasks of the Operations, Hyperstructure Store, and Proxy components. The modifications made to the Mosaic browser combined with the code handling it in the user context control application makes up the Browser component.

The AgentDLS [8] architecture consists of a link server agent, that analyses Web pages sent to it by a HTTP proxy and based on this uses link resolvers and link bases to compile a list of relevant links. The role of the proxy is a simple case of the Proxy component with the twist that documents are sent to two parties. The link server agent acts as a combined Operations and Hyperstructure Store component. The analysis performed by link resolvers also would seem to belong to this component, depending on the set-up of the navlet. The Browser component's responsibility – letting the system set or get the state of the Web browser – is handled at two levels: in the Proxy component (what document is displayed now?) and in the AgentDLS window, where users can directly influence, what URL to display in their Web browser.

Guided tour tools require a structure server for storage of the tours as well as the ability to communicate with this server. This can be modelled through the Operations and Hyperstructure Store components. To keep track of where the reader is on the trail (and to put the reader back on track, if need be), it must be possible to set and read the state of a Web browser, which is modelled through the Browser component. Some guided tour tools, e.g. Walden's Paths, modify the Web pages to add comments and interface. While it may not be necessary for interface reasons in the Arakne framework (this could be handled by a navlet), it certainly can be done through the use of the Proxy component.

Could the current implementation of Walden's Path be fitted within the Arakne model? In this case, some of the Arakne components collapse into one. The Path Server acts as the Proxy, Hyperstructure Store, and Operations components, by handling both external structures (guided tours) and Web page modification. The Browser component aspect is handled by the modification of links that keeps the system (i.e. the Path Server) aware of the state of the Web browser. The interface aspects of the navlet are the header and footer of the displayed Web pages that give the user a possibility to interact with the system.

Spatial and structuring Web augmentation tools certainly need structure servers and information about the currently display Web page just as the rest of the above described tools. However, the main focus of the spatial/structure tools described in this paper has been on the user interface and the ability to analyse and process structure and relationships using user written scripts. The user interface is handled by the navlet itself. While the navlet of course would serve as the front-end, the scripting capability fits best within the Hyperstructure Store component, where all structure is stored during run-time and where the Operations component can be directly accessed.

AN IMPLEMENTATION OF THE ARAKNE FRAMEWORK

A system called the Arakne applet has been developed as a part of the Coconut project. The immediate goal of the implementation was to try out the soundness of the Arakne framework by integrating the guided tour tool Ariadne and the link creation tool Navette. The system has undergone some development and has proven robust to the interchange of components.

The system was originally developed as a Java applet running in Netscape Communicator [31]. The components within the dotted line in Figure 1 were a part of this particular implementation. This was an implementation choice; as can be seen from the previous section, the framework itself posits no such requirements on the location of the individual components.

The components in Figure 1 correspond to Java classes in the Arakne applet. The interfaces between components are handled in the MVC (Model View Control) idiom, where the 'view' (e.g. a graph displaying a guided tour) is loosely coupled through events to the actual data, the 'model'. Modifications of data is handled as requests through the 'controller' and if granted notified to the view through the use of events.

The Hyperstructure Store class caches structure retrieved through the Operations class and is the only interface to the structure servers given to the navlet beans. This is done to improve performance (by not retrieving the same information twice) and to ensure data integrity between the Hyperstructure Store and the structure servers. The Hyperstructure Store provides the navlet beans with a rich set of convenience methods, which are translated into queries to the structure servers by the Operations class.

The Browser class encapsulates the needed functionality to communicate with a Web browser, in the original case the Netscape Communicator. The interface is relatively simple and can be adapted to another browser.

The Proxy class is a small proxy running as a thread in the applet. The Netscape Communicator can via JavaScript be set up to use a proxy on the fly, and when the Arakne applet is running, the Proxy class acts as a proxy for the browser. The Proxy class uses whatever proxy the browser was configured to use, and when the Arakne applet is terminated, everything is returned to normal. The Proxy handles requests for Web page modifications and the correct display of frames (thus allowing user to link into frame sets).

All of the classes visible to the navlet beans, the Hyperstructure Store class, the Proxy class, and the Browser class generate events that the navlet beans may subscribe to. The navlet beans are currently restricted to being Java beans and to follow some design guidelines. The current implementation supports only compile time integration of navlet beans giving the user a fixed number of available navlet beans, but future versions of the Arakne applet should be able to load navlet beans on runtime so that the user can retrieve navlet beans from different servers.

The current version of Netscape Communicator[2] has a very limited API for browser integration as well as a faulty Java socket implementation. This lead to the abandonment of the Communicator as the supported Web browser at a time where most components as well as two navlets were finished or nearing completion. The Microsoft Internet Explorer was chosen as the new supported Web browser. This choice has brought some costs, as the Arakne applet is now not only browser-dependent (which was also the case before), but

[2] Version 4.5

now also platform dependent, as Java is not supported in the Microsoft Internet Explorer on anything but the Windows platform.

The migration to the Internet Explorer also served as a (unintended) test of how much code needed to be rewritten to accommodate the change. This code rewrite has been light. The Browser class has been redone, as the Internet Explorer is controlled not through JavaScript, but through a COM interface. Furthermore some unique features such as the ability to operate on selected text in a browser window through the use of right-click menus have been exploited in some changes in user interface, e.g. link creation.

The Internet Explorer does not allow proxy configuration through JavaScript or other means, which has led to some major changes in the Proxy component. The current version relies on the DHMProxy [18] for Web page modification. This is expected to change as the DOM model [11] gives excellent possibilities for Web page analysis and modification, after the Web page has been rendered by the Web browser. This work is expected to be heavily dependent on the experiences from the Webvise application [18].

The current version uses the Devise Hypermedia server as a structure server. This is changing rapidly however, as a new Hyperstructure Store class is being developed to use a new Operations component that utilises the Open Hypermedia Protocol [33]. This has numerous advantages. The OHP is a powerful protocol with a good and general data model. Among the interesting features of OHP is the support for sessions, so that e.g. link following happens simultaneously on several machines as demonstrated at the demo at Hypertext 98. Finally the open hypermedia community supports the OHP and the data model, so that the Arakne applet may communicate with other structure servers such as OHP compliant MicroCosm or HyperDisco servers.

The synchronous collaborative aspects of the Arakne applet have been put on hold until the OHP is fully integrated, though the system as it is supports asynchronous collaboration.

Currently the Arakne applet supports the Ariadne and Navette navlet beans. The user is thus able to create links in documents while creating guided tours, which was not possible before. Each of the navlet beans occupies an internal window in the Arakne applet. The Arakne applet provides the interface for logging on to structure servers and the selection of contexts.

The Navette navlet has recently been extended to support linking to and from segments in video and audio clips through the Mimicry system [4]. Most temporal media plugins do not have APIs well suited for the needs of open hypermedia, which has led to the development of the Mimicry player. The Mimicry player is a Java applet capable of handling most video and audio formats, and it provides a rich API for open hypermedia integration.

Future Plans

Future plans for the Arakne applet include more navlet beans. Currently a spatial navlet bean is under development. The navlets currently supported are not aware of each other, but future versions of Arakne will support inter-navlet communication. The Microsoft Internet Explorer is not expected to be the only Web browser supported by future Arakne applets. The upcoming Mozilla 5.0 holds great promise in this regard. Another area of interest to the developers is the implications of XLink and XPointer.

Experiences of the Development of the Arakne Applet

How does the current Arakne applet compare to the problems raised in the summary of strategies for Web augmentation? Network communication is handled through sockets by the Operations Component, and the client maintains a socket connection, so that the structure servers may contact it. The Arakne applet has so far only supported navlets orthogonal to each other, so there is currently no experiences with regards to two navlets trying to modify the same Web page. However the architecture has only one component, the Proxy component, allowed to modify Web pages, so it should be possible to contain and perhaps avoid most problems. The Arakne applet is currently dependent on a proxy with the usual advantages (all browsing activities are ‘captured’ by the system) and disadvantages (the user needs to configure the Web browser to use the proxy). The proxy can use other proxies without problems, though the use of two Web page modifying proxies certainly would lead to undefined results. The proxy handles frame sets and inserts links through the use of JavaScript and DOM, so that Web pages are (visibly) modified on arrival rather than in transit, which solves many problem regarding dynamically created documents and frame sets. The Arakne applet is a Java applet with the limitations imposed on Java applets. The security restrictions are handled through digital signing. The Arakne applet is fairly secure from being unloaded by mistake, as it runs in a separate non-resizable browser window with disabled menus and toolbars. A consequence of the integration with the Microsoft Internet Explorer is that a user can not just download and use the Arakne applet. Certain files have to be installed by the user to provide the Arakne applet with right-click menus. This process can however be largely automated.

Some of the solutions found in the Arakne applet are strictly browser and platform dependent. This is very unfortunate, and the only thing that can remedy the situation is a new standard Java API for Web browsers. This API should at least provide functionality similar to that of the API of the Microsoft Internet Explorer, but do so in a browser independent way. Given the current Web browser situation, we think that such an API is unlikely to appear anytime soon, so the next best solution would be a Web browser that provided a platform independent API. Whether or not Mozilla 5.0 will provide such an API remains to be seen.

CONCLUSION

Web augmentation tools will in all probability remain a part of the Web, as researchers and users will continue to explore the boundaries of what hypermedia is and what it can be used to. An understanding of the strategies employed in Web augmentation is needed to make the next generation of Web augmentation tools easier to envision, develop and use.

We have investigated the recurring themes and techniques found in current Web augmentation tools. Based on this, we have developed and described the Arakne framework, and shown that the Arakne framework accommodates existing Web augmentation tasks, and that most tools can be modelled using the framework. A shared framework for these tools could benefit analysis and communication as well as development, and it is hoped that the Arakne framework is a step in the direction of the creation of such a framework.

The interesting and challenging part of creating a Web augmentation tool is not the nitty-gritty of interfacing to a Web browser or writing a proxy. The interesting part, at least in the author's experience, is to create tools that can help users structure their work and their browsing, and by implementing the Arakne applet some of the work needed to provide a full infrastructure for Web augmentation tools have been developed.

The Web has succeeded through (more or less) strict adherence to open standards that have been jointly developed by the interested parties. This has led to the development of standard tools usable for all. The continuation of this trend is crucial to the future development of the Web. The Open Hypermedia System initiative if supported by the hypermedia community can be become a shared standard from which the entire hypermedia community will benefit.

ACKNOWLEDGEMENTS

The author is a member of the Coconut project (http://www.cit.dk/coconut/), a joint research project consisting of Department of Computer Science, Aarhus University and Tele-Danmark Internet. The Danish National Centre for IT-Research (http://www.cit.dk/) supports the Coconut project.

The author wishes to thank Jesper Jühne for valuable discussion of Arakne, René Thomsen for adding to the code, Kaj Grønbæk for valuable advice, and the anonymous reviewers for good suggestions to structure this paper.

REFERENCES

[1] Anderson, K. M., Taylor, R. N., and Whitehead JR., E. J. (1994). Chimera: Hypertext for heterogeneous software environments. In *Proceedings of ECHT 94 Conference*, pp. 97–107, Edinburgh, Scotland.

[2] Anderson, K. M. (1997) Integrating Open Hypermedia Systems with the World Wide Web. In *Proceedings of the ACM Hypertext 97 Conference*, pp. 157–166, Southampton, England.

[3] Bouvin, N. O. (1998). Designing Open Hypermedia Applets: Experiences and Prospects. In *Proceedings of the ACM Hypertext 98 Conference*, pp. 281–282, Pittsburgh, USA.

[4] Bouvin, N. O., and Schade, R. (1999). Integrating temporal media with open hypermedia on the WWW. Submitted for publication.

[5] Bush, V. (1945). As we may think. In *The Atlantic Monthly* 176, 1 (July 1945), pp. 101–108.

[6] Carr, L. A., De Roure, D., Hall, W., and Hill, G. (1995). The distributed link service: A tool for publishers, authors and readers. In *Proceedings of the 4th International World Wide Web 95 Conference*, Boston, USA.

[7] Carr, L. A., Hill, G., De Roure, D., Hall, W., and Davis, H. (1996). Open information services. In *Computer Networks and ISDN Systems,* 28, pp 1027–1036, 1996.

[8] Carr, L. A., Hall, W., and Hitchcock, S. (1998). Link services or link agents? . In *Proceedings of the ACM Hypertext 98 Conference*, pp.113–122, Pittsburgh, USA.

[9] Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R. (1992). Towards an integrated information environment with open hypermedia systems. In *Proceedings of the ACM Hypertext 92 Conference,* pp. 181–190, Milan, Italy.

[10] CritSuite. http://crit.org/http://crit.org/index.html

[11] Document Object Model. http://www.w3c.org/TR/REC-DOM-Level-1/

[12] Furuta, R., Shipman III, F. M., Marshall, C. C., Brenner, D., and Hsieh, H-W. (1997). Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths. In *Proceedings of the ACM Hypertext 97 Conference*, pp. 167–176, Southampton, England.

[13] Goland, Y., Whitehead, J., Faizi, A., Carter, S., Jensen, D. (1998). Extensions for Distributed Authoring on the World Wide Web – WebDAV. http://www.ics.uci.edu/~ejw/authoring/protocol/draft-ietf-webdav-protocol-08.pdf

[14] Grønbæk, K., and Trigg, R. H. (1994). Design issues for a Dexter-based hypermedia system. In *Communications of the ACM,* 37 (2) February, pp. 40–49, 1994.

[15] Grønbæk, K., and Trigg, R. H. (1996). Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects. In *Proceedings of the ACM Hypertext 96 Conference,* pp.149–160, Washington DC, USA.

[16] Grønbæk, K., Bouvin, N. O., and Sloth, L. (1997). Designing Dexter-based hypermedia services for the World Wide Web. In *Proceedings of the ACM Hypertext 97 Conference,* pp. 146–156, Southampton, England.

[17] Grønbæk, K., and Will, U. K. (1997). Towards a common reference architecture for open hypermedia. In *JoDI, Journal of Digital Information,* 1(2), 1997. http://jodi.ecs.soton.ac.uk/Articles/v01/i02/Gronbak/

[18] Grønbæk, K., Sloth, L., and Ørbæk, P. (1999). Webvise: Browser and proxy support for open hypermedia structuring mechanisms on the WWW. Submitted for publication.

[19] Guha, R. V., and Bray, T. (Eds.). Meta content framework using XML. http://www.textuality.com/mcf/NOTE-MCF-XML.html

[20] Hall, W., David, H., and Hutchings, G. (1996). *Rethinking Hypermedia: The Microcosm Approach.* Kluwer Academic Publishers, Norwell, USA.

[21] Hot Sauce. http://www.xspace.net/download/index.html

[22] HyperWave. http://www.hyperwave.com/

[23] Jühne, J., Jensen, A. T., and Grønbæk, K. (1998). Ariadne: A Java-based guided tour system for the World Wide Web. In *Proceedings of the 7th International World Wide Web 98 Conference*, Brisbane, Australia.

[24] Marshall, C. C., and Shipman III, F. M. (1995). Spatial hypertext: Designing for change. In *Communications of the ACM*, 38 (8) August, pp. 88–97, 1995.

[25] Marshall, C. C. (1998). Toward an ecology of hypertext annotation. . In *Proceedings of the ACM Hypertext 98 Conference*, pp. 40–49, Pittsburgh, USA.

[26] Maurer, H. (Ed.) (1996). *Hyper-G now, HyperWave: The next generation web solution*, Addison-Wesley, Harlow, 1996.

[27] Meyrowitz, N. (1989). The missing link: Why we're all doing hypertext wrong. In Barrett, E. (ed.). *The society of text: Hypertext, hypermedia and the social construction of information*, pp. 107–114, MIT Press, Cambridge, Massachusetts., USA, 1989.

[28] Microsoft Internet Explorer. http://www.microsoft.com/ie/

[29] Mozilla. http://www.mozilla.org/

[30] NCSA Mosaic. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/

[31] Netscape Communicator. http://home.netscape.com/download/index.html?cp=djudepart

[32] Nürnberg, P. J., Leggett J. J., Schneider, E., and Schnase, J. L. (1996) Hypermedia operating systems: A new paradigm for computing. In *Proceedings of the ACM Hypertext 96 Conference*, pp.194–202, Washington DC, USA.

[33] The Open Hypermedia Systems Work Group. http://www.ohswg.org/

[34] Rittel, H. and Webber, M. (1973). Dilemmas in general theory of planning. In *Policy Sciences,* Vol. 4, 1973.

[35] Röscheisen, M., Mogensen, C., and Winograd, T. (1995) Beyond browsing: Shared comments, SOAPs, trails, and on-line communities. In *Proceedings of the 3rd International World Wide Web 95 Conference*, Darmstadt, Germany.

[36] Röscheisen, M., Winograd, T., and Paepcke, A. (1995). Content ratings and other third-party value-added information: Defining an enabling platform. In *D-Lib Magazine*, August 1995, http://www.dlib.org/dlib/august95/stanford/08roscheisen.html

[37] Shipman III, F. M., Furuta, R., Brenner, D., Chung, C-C., and Hsieh, H-W. (1998). Using paths in the classroom: Experiences and adaptations. In *Proceedings of the ACM Hypertext 98 Conference*, pp. 267–276, Pittsburgh, USA.

[38] Shipman III, F. M., Furuta, R., and Marshall, C. C. (1997). Generating web-based presentations in spatial hypertext. In *Proceedings of the 1997 International Conference on International User Interfaces*, pp. 71–78, Orlando, USA.

[39] Stanley, T. (1998). Contextures: Focus + context + texture. In *Proceedings of the ACM Hypertext 98 Conference*, pp. 295–296, Pittsburgh, USA.

[40] Trigg, R. H. and Weiser, M. (1986). TEXTNET: A network-based approach to text handling. In *ACM Trans. Office Information. Systems* 4, 1 (Jan 1986), pp 1–23.

[41] Trigg, R. H. (1988). Guided tours and tabletop: tools for communicating in a hypertext environment. In *ACM Trans. Office Information. Systems* 6, 4, pp 398–414, 1988.

[42] Web Squirrel. http://www.eastgate.com/squirrel/Welcome.html

[43] Whitehead Jr., E. J. (1997). An architectural model for application integration in open hypermedia environments. In *Proceedings of the ACM Hypertext 97 Conference*, pp. 1–12, Southampton, England.

[44] Wiil, U. K. and Leggett, J. J. (1996). The HyperDisco approach to open hypermedia systems. In *Proceedings of the ACM Hypertext 96 Conference*, pp.140–148, Washington DC, USA.

[45] Wiil, U. K. and Leggett, J. J. (1997). HyperDisco: collaborative authoring and Internet distribution. In *Proceedings of the ACM Hypertext 97 Conference,* pp. 13–23, Southampton, England.

[46] Wiil, U. K., and Østerbye, K. (1998). Using the Flag taxonomy to study hypermedia system interoperability. In *Proceedings of the ACM Hypertext 98 Conference*, pp. 188–197, Pittsburgh, USA.

[47] XML – Extensible Markup Language. http://www.w3c.org/XML/

[48] Østerbye, K., and Wiil, U. K. (1996). The Flag taxonomy of open hypermedia systems. In *Proceedings of the ACM Hypertext 96 Conference*, pp.129–139, Washington DC, USA.