Beyond proxies: XLink support in the browser

Angelo Di Iorio
University of Bologna
Via Mura Anteo Zamboni 7
40100 Bologna (Italy)
+390512094871
diiorio@cs.unibo.it

Gabriele Montemari
University of Bologna
Via Mura Anteo Zamboni 7
40100 Bologna (Italy)
montemar@cs.unibo.it

Fabio Vitali
University of Bologna
Via Mura Anteo Zamboni 7
40100 Bologna (Italy)
+390512094872
fabio@cs.unibo.it

Abstract
In this paper, we describe some issues that every implementation of external linkbases using XLink has to consider, and provide a number of possible solutions. We particularly consider as relevant the issues connected to providing navigation-related features, that is, adding external links to the actual web pages before displaying them, and to providing creation-related features, that is, the user interface elements necessary to users to create new links, or modify and delete existing ones. Examples are carried out based on our experience with two different architectures for XLink application, a proxy-based solution called XLinkProxy, and a browser-based solution for Mozilla browsers called XLinkZilla, which is described in this paper for the first time.

Categories and Subject Descriptors
H.5.4 [INFORMATION INTERFACES AND PRESENTATION]: Hypertext/Hypermedia – *Architecture.*

General Terms
Design

Keywords
XLink, external links, linkbases, Mozilla browsers.

1. Introduction
One of the reasons personalized links are not gaining importance in hypertext systems could be reasonably attributed to the lack of adequate tools to create custom links on hypertext content. It is pretty obvious to say that the expressive power of a hypertext system primarily depends on the expressive power of the underlying linking mechanisms.

Drawing from the taxonomy proposed in [7] and [9], three main models for storing link information on a document set can be identified: the embedded link model, the anchor table model and the external link model.

In the *embedded link model* the source anchor is embedded within the source document and contains explicit information about the destination anchor. This is the model taken in HTML and presents some significant limits: only users having write permissions on a document can add this link, no external or read-only resource can be linked and no links set management is supported.

The *anchor table model* (a.k.a. external links with named endpoints) is the one where information about the link is stored outside the documents it connects, possibly in a central link database, but a link contains references to permanent elements of the linked documents. This model offers a large flexibility but relies on the persistence of the association and allows the links to only those fragments that are associated to a persistent name.

The *external link model*, finally, is the model where information about the link is stored outside the document it connects and the link contains information to allow future identification (usually through counting or searching) of the exact linked fragment. This model allows the greatest flexibility, since the documents are not modified in any way by the existence of the links. External linkbases allow the creation of links on media on which no modification can be performed, either because it is read-only (e.g., a CD-ROM), or because we do not have the rights to modify it. Furthermore, they may be used to provide separate and independent link sets on the same set of documents, to implement intensional links, and in particular generic links (i.e., links associated to a string, rather than a position) and to act as a centralized and controllable database of links that can be verified and updated together when the document set changes or evolves. On the other hand, an external link mechanism has to be supported by a flexible referencing system: any change to the linked documents, in fact, can easily make the link information incorrect if the system does not carefully deal with the changes. Yet, it is very difficult to coherently update the link information.

A substantial flexibility in link models has been finally made available to the WWW through the XLink standard [11], which enhances links in web pages by introducing many improvements to the plain HTML link model, among which the possibility to express a link externally to the resources it connects. So far, though, web browsers have been slow to adopt this innovation. Mozilla, for instance, claims support for XLink, but only limited to simple links (which are conceptually identical to plain HTML links).

Deleted: that

While waiting for support for sophisticated link types to become an integrated feature of popular browsers, many research prototypes have circumvented the problem by converting the set of sophisticated link features available with XLinks into plain HTML anchors (possibly enriched with some JavaScript and CSS to account for the additional services available with XLinks, see [14]) and placing them on-the-fly in the original unmodified document before the result is shown within a standard WWW browser. Another feature to be considered is providing the users with solutions and user interfaces to create and update these links, allowing readers to create personal links during the browsing.

Proxies are a popular architecture for this doctoring of web pages with additional links before sending them to standard browsers.

Proxies in fact are easy to write and set up, and they are a regular part of the HTTP chain of connections. Our own XLinkProxy [4], as well as other projects such as Goate [14] or Webvise [15], are proxy-based experiments with XLink. Proxies have also difficulties and drawbacks, though, mainly related to subtle complexities in providing a user interface, and to the restriction policies that some networks might impose on their use.

Many current generation browsers, on the other hand, allow custom widgets such as menu items and sidebars to be installed locally and perform additional functions on documents and web sites. These widgets intrinsically provide a user interface, and would make the whole doctoring process completely client-side, thus completely avoiding the difficulties related to proxies.

This paper introduces and describes XLinkZilla, a linkbase application for XLink-based external links. XLinkZilla is composed of an extremely simple server-side CGI application for storing and delivering XLinks, and two installable add-ons for Mozilla browsers providing all the needed functionalities and interfaces for a complete external link management. [Non solo Mozilla??]

In section 2 we will examine some related works, especially regarding external linkbases and XLink applications. In section 3 we will present and discuss a minimal set of features that an XLink application needs to perform when coupled with a standard HTML browser, and the problems that may occur with this. In section 4 we discuss pros and cons of the proxy architecture in providing such features, based on the experience we have had with our own XLinkProxy [4]. Section 5 introduces and discusses XLinkZilla, our new research prototype based on client-side add-ons.

[Che innovazione portiamo???]

2. Related works

In the literature a lot of different works exploited the concept of external linkbases to enhance their linking mechanisms, starting from the first proposal of Amy Pearl, Sun's Link Service [Pea87], an independent link service based on a strong separation between content and links. Microcosm [6] is an open hypermedia system enabling users to browse through and query collections of linked multimedia information. Microcosm (and subsequently Webcosm) allows the connection between different resources created with different desktop tools and by different authors. From the same research came out Distributed Link Service [2] a modular link service for the World Wide Web, subsequently, enhanced with alternative transport mechanisms for links [10] and with automatic discovery mechanisms [3]. Hyper Wave [17] enables a more structured organization and visualization of Web content and links, even if there is no direct way to create multi-ended links. The interface is transparent to the user and the presentation of links is in traditional HTML syntax. Aquarelle [20] introduced an application to automatically and centrally verify the correctness of links and MMM [18] addressed the synchronization of multiple views of the same document though external links and a proxy architecture. The Open Hypermedia Protocol [8] (subsequently evolved in the Fundamental Open Hypermedia Protocol [19]) proposed models to express links independently of their storage location. Devise and Webvise [15] implemented similar functionalities on the Dexter Hypermedia Model.

The evolution of the hypertext should have been parallel with the evolution of the linking mechanism but this is not completely true. All these systems, for instance, have been eclipsed by the World Wide Web. Nevertheless the Web relies on a simple and limited linking model [1] and only in the recent period has welcomed the use of the external linkbases.

The proposal that should change things for the better is certainly XLink [11], the linking language for XML that, in addition to simple links, introduces the concept of extended links. An extended link is an XML fragment whose elements are labeled with a specific linking role (through an attribute belonging to a specific namespace): the resources are anchors local to the document where the link resides, the locators are anchors on different documents and the arcs are navigable directions in the link. Thus, external, multi-destination and bidirectional links can be easily expressed by XLink.

XLink is based on XPointer, a fine-grained addressing mechanism [12], allowing addressing arbitrary fragments in a XML document regardless of the structure of the document. In addition to single and whole elements (addressable by XPath) strings, points, and ranges can be identified (note that these fragments correspond to text selections in some user interfaces).

Many different implementations of XLink exist. Both the Amaya and Mozilla browsers claim to support simple XLinks (and a limited subset of XPointers). X2X [26] is a tool to easily create, manage and store external links that are re-inserted into documents on the fly. X2X does not use XPointer but exploits proprietary functions such as substring and subword to allow sub-node links in the anchors. Implemented in Java and based on an ODBC/JDBC enabled database, X2X was designed to be integrated into any static or dynamic application or service.

Another tool providing the same specific functionalities as a generic module to plug in more complex systems is the Fujitsu XLink Processor [13]. Furthermore the Fujitsu Processor provides full support for XPointer. XLinkit [27], on the other hand, is a complete application that makes use of XLink to provide a rule-based verification mechanism for link consistency in a document base. Related to XLinkit, XTooX [29] is a free XLink processor that turns out-of-line links into inline links. It takes a linkbase as its input and puts the links into the referenced documents.

Given that current Web browsers only support HTML or only have a limited support for XML technologies such as XLink, several proposals are based on intermediary and transparent application that turn external links into a set of internal links, understandable by the common browsers, and merge them with the original document without altering it. For instance, Goate [14] embodies this solution through a HTTP proxy. Goate does not implement any particular linking language, but instead provides a translation service between various “high-level” languages (the authors have used this term to indicate languages, such as XLink, allowing external, multi-ended and bidirectional links) and HTML. Although the first implementation of Goate only deals with XLink documents, his general purpose language CLING is planned to be used with future (and more powerful) linking languages too.

Deleted: s

Deleted: s

Deleted:

Deleted:

Finally, Xspect [5] is an implementation of XLink based on standard XSLT and Javascript that provides navigational hypermedia functionalities, a SVG interface to guided tours and an authoring environment for annotations. Xspect uses XLink, but can convert to and from other hypermedia formats, such as OHIF [16] and FOHM. The attention is focused on the browsing and displaying of the linkbases rather than their creation or the surfing monitoring, provided for instance by Goate, XLinkProxy and XLinkZilla.

3. Using XLinks in web browsers

Current Web browsers only support HTML links or only have a limited support for XLink links (e.g., only for inline simple XLinks). Any implementation supporting more than the simplest features of XLink on the standard browser is therefore necessarily based on the conversion of external sophisticated links into internal, simple, unidirectional ones, most probably through the intervention of an intermediary and transparent application that convert external data and inserts it into the main document. Obviously the new simple links, understandable by the plain browser, need to be merged within the main document without altering its original content.

Independently of the implementation choices and the whole architecture of the system, it is possible to give a high-level description of the features that any typical XLink application for the plain browser must provide currently.

These features can be divided in two classes: navigation-related features and creation-related features.

3.1 Navigation-related features

A complete implementation of extended XLinks should be able to manage multi-directional, external, intensional links (including generic links).

More precisely, a *multi-directional link* connects one or several sources to one or several destinations in different web documents. An *external link* is not stored within the source document, but in some external resource that is loaded or known to the application. An *intensional link* specifies one or more of its anchors as formulas to be computed on the document, rather than an actual address of a fragment of the content. In particular, *generic links* have one or more anchors referring to all the instances of a given word or string in the document, and may correspond to several actual positions in the document. XLink links and XPointer addresses are perfectly capable of expressing these references.

Assuming that the links already exist and are stored in a well-known external linkbase, navigation-related features include the following services:

- Monitoring the navigation activities on the main browser window, and identifying any new access to a web page
- Finding from the active linkbases all XLinks relevant to the new web page
- Transforming XLink information into one or several plain HTML links that can be fruitfully understood by browsers
- Adding the links to the document, and delivering it to the browser
- Providing support for the selection of destinations in multi-destination links.

3.1.1 Monitoring the navigation activities

Pages enriched by external links are composed of two independent parts, the original document on its origin web server and the external links to be added on-the-fly. So the first of the navigation-related features necessarily consists of finding out that a new URL request has been performed by the browser. The request can be intercepted (e.g. by a proxy implementation) and carried out directly by the XLink implementation for the browser or just monitored for a parallel link acquisition phase to be activated. It is important to notice that XLink implementations do not need to act in the requesting phase, but just need to add information to the resource after it has been downloaded from the origin server.

Deleted: H

3.1.2 Finding all relevant XLinks from the active linkbases

The appropriate linkbases are selected among all the available ones. This selection can be automatically made by the system based on user's authentication or explicitly by the user who identifies the linkbases he/she wants to be consulted. Since all XLink linkbases are actually XML documents, this corresponds to performing a query on an XML database: this in turn can be as sophisticated as a full XQuery on a native XML database or as simple as executing a plain XPath search on the file of the linkbase. In all cases, this corresponds to looking for XLinks whose "from" locators have the same URL as the document being displayed on the browser.

3.1.3 Transforming XLink information into HTML

The XLink information drawn out of the linkbases is not immediately usable, given the simplicity of the HTML linking model. Therefore, in order to use current generation browsers, intensional references and multiple anchors have to be resolved into several separate instances of plain HTML anchors. After having detailed each instance of each multiple anchor, other problems need to be solved regarding counting, overlapping anchors and nested links.

Deleted:

By sequentially inserting new elements within the document, in fact, its internal structure changes, so that every XPointer location after the first one may end up pointing to the wrong elements, in particular if the XPointer uses counting to identify the anchor. Strategies have to be implemented to keep the link information consistent and reliable and to insert each link in its correct location, regardless of the intermediate modifications to the DOM.

It could further happen that two external links refer to partially overlapping text fragments, or that an external link refers to a text fragment which is partially overlapped by another element in the document. In this case it is necessary that the overlap is resolved, possibly by creating multiple fragments for the anchor, before actually inserting the anchor in the document.

Finally, some problems may derive from nested anchors. HTML, in fact, does not allow A elements within other A elements, and Web browsers behave erratically in these situations. On the other hand, though, it is inappropriate and technically impossible to impose constraints to the user when creating new links. A reasonable solution to this problem consists of extending the innermost anchors to completely overlap the outermost anchor, and to treat them as additional destinations of it. This solution is imprecise, but it handles the situation with a minimum of hassle.

Deleted: allows to

3.1.4 Adding the links to the document

Once these situations have been considered and the problems have been resolved, it is time to actually add the elements to the documents. Note that links with multiple anchors or destinations are already split in different items and counting XPointers have been corrected. So the next step consists of inserting a new A element in the correct location within the document for each computed link, and making sure that it contains all the relevant information. Having merged all the external links, a modified version of the document is ready to be delivered to the browser. Before doing so, a few more items may be further added to the document, such as references pointing to the external files where JavaScript and CSS support to links might be provided.

3.1.5 Support for selection of multiple destinations

XLinks leaves to the implementation any decision about the meaning and user interface widgets associated to the activation of a link with multiple destinations. A popular and well-motivated choice [25] is to add a pop-up menu to the link anchor where all destinations are listed and can be individually selected. JavaScript provides many ways to implement such pop-up menus in standard browsers.

3.2 Creation-related features

Besides navigating the web using links that are not specified in the document itself, the real advantage of external links should be to allow each individual reader to add his/her own links and annotations to the pages being visited.

The possibility to create new links on documents, and to share them with friends and colleagues, is what makes external linkbases more than a bizarre storage policy for link information.

[Ripetere all'inizio, per segnare l'importanza di Xlink????????????????????????????]

Creating new links is done through an additional interface widget, for instance an independent frame of the page, or a sidebar, or a toolbar, or a browser menu, or even an independent application. We may want to support direct anchor specification (i.e., using selections in the browser window), as well as manual insertion of the XPointer string. Thus the widget needs to monitor and access data about the events in the main browser window during the browsing activities.

In order to manage user-created links, creation-related features include the following services:

- Selecting the start- and the end-points of the link
- Calculating the appropriate XPointer of the endpoints
- Setting options for the new link
- Saving link information in the relevant linkbase
- Modifying and deleting existing links

3.2.1 Selecting the start- and the end-points of the link

Whenever a user makes selections on the main text, a JavaScript object is created in modern browsers containing the selected string and the position of the string within the document. This is not true for old browsers (Netscape 4.x has a simpler DOM and does not reveal the position of the selection within the document) and, obviously, for browsers not supporting JavaScript. This data should be made available to the link creation widget. Since in most cases the start- and end-points will belong to different documents, and since there could be many start- and end-points in the same link, it is important that the link creation widget keeps persistent information on past selections and past pages so as to be able to create the full link once the author identifies all the appropriate data.

3.2.2 Calculating the appropriate XPointer of the endpoints

The link creation widget must include a library implementing the conversion of addresses in a HTML DOM to an XPointer address. In creating links, the library need only to deal with the simplest XPointers, explicit address of strings within the document. More complex XPointers (e.g. intensional links and particularly generic links) are difficult to create with direct selections on the documents, and may need to be entered manually through some appropriate forms.

3.2.3 Setting options for the new link

The link creation interface widget needs to include forms with buttons and input fields to set further XLink options for the link. For instance, the user may want to specify the title of the whole link and of every single anchor, or dictate behavior attributes such as "show" (indicating the desired presentation of the ending resource of the link) or "actuate" (indicating the desired timing of traversal from the starting-point to the ending-point).

3.2.4 Saving link information in the relevant linkbase

Once determined, the link data need to be sent to the external linkbase, expressed according to the XLink specifications. Here it will be saved within the linkbase, which could be as simple as a plain XML file on the file system of the server, or as sophisticated as a full fledged XML database. The linkbase is probably best placed on a third-party server accessed via an HTTP request. If the service provides support for registration of users and authentication, the client-side application needs to provide the appropriate certificates before being allowed to save.

3.2.5 Modifying and deleting existing links

Although modification and deletion functionalities to existing links could be provided with other and simpler means (e.g., via plain web forms accessible on the web server of the linkbase server), it may be felt elegant to provide some interface to select and access the link to be modified or deleted from within the link-creation widget itself. How does the user select the link to be modified or deleted? Relying on finding and remembering IDs, titles or XPointers for the identification of the link clearly is heavy burden on the user. A possible elegant solution is to automatically add an item in the popup menu connected to the anchor, allowing the user to modify and delete the selected link. Of course this item needs to be given failsafe features and a graphical aspect that clearly differentiates it from plain destination items.

4. Issues in the implementation of XLink applications

Any application providing external linking services is composed of a pair of collaborating modules placed on different machines: a linkbase server, providing some kind of storage of link documents and searching of relevant links to a hypertext document, and a user interface providing access to the creation-related features described in section 3.2.

Deleted: e

Deleted: do not actually belong in the document being shown

Deleted: storing

Formatted: Italian Italy

Formatted: Italian Italy

Formatted: Italian Italy

Deleted:

Deleted: n

Deleted:

4.1 Linkbase servers

The monitoring of navigation activities and the nature of the HTTP protocol impact on the possible architectures for linkbases, leaving only two candidates: proxies and plain CGI applications.

4.1.1 Proxy-based solutions

In [4] we presented XLinkProxy, a full-featured proxy-based external linkbase management system completely supporting XLink and a relevant part of the XPointer language (a few functions were missing). The proxy-based architecture implemented in the XLinkProxy prototype can be considered analogous to what can be found in other projects such as Goate [14], Xspect [5] and Webvise [15], more or less providing similar functionalities. We believe that an analysis of the features of XLinkProxy could be useful in order to determine the advantages and disadvantages of the proxy-based architecture for external linking *per se*.

The basic working of XLinkProxy is as an HTTP proxy, i.e., as an intermediary between a browser and a Web server. Whenever XLinkProxy receives a resource request from a browser, it forwards it to the appropriate HTTP server, and upon receiving the response, it forwards the document to the requesting browser. XLinkProxy is a transparent proxy (i.e. a proxy that does not change in any way the response of the origin HTTP server) for all resources that are not HTML or XML documents. For documents that are either HTML or XML, XLinkProxy can add external links to it, retrieved from the linkbases stored on the proxy.

The execution cycle of a proxy-based XLink implementation is basically as follows: every received request is sent to the origin server for downloading the unmodified document; upon receiving the document, all links relevant to the requested URL are searched in the local linkbases, and the relevant information extracted. The original document and the link information are then merged, and the resulting document is sent to the user agent as response to its initial request.

This architecture has many points in its favor:

- it relies on server-side technologies such as PHP, Perl, ASP or Java, which are sophisticated, easy to code and substantially more stable and reliable than client-side technologies;
- the computation intensive insertion of link data into the requested document is done on the proxy, with minimal workload left to the browser, and with large caching possibilities;
- navigation-related features are completely independent of the client technology, and can work with any browser, since the browser only receives a very plain HTML document (but this is not completely true if we consider multi-destination links, which require JavaScript, as discussed in section 3.1.5).

Proxy-based XLink applications provide proxy-side support for most of the navigation-related features, as described in section 3.1, but rely on the browser to provide support for the selection of multiple destinations and for all the creation-related features required by all XLink applications. For these, in fact, browser-specific customization steps need to be considered.

The most evident advantage of proxy is to give a natural and easy solution to the issue of monitoring the navigation activities of the browser: the user can activate the link service by simply setting his/her browser to use this specific proxy, and from then on every document request will be filtered by the proxy-side XLink application.

The advantages are clear and immediate: no special set-up, installation process or navigation is required for this, and every single browser currently available provides support for proxies and can make use of proxy-based functionalities.

On the other hand, there are some drawbacks that need to be considered as well.

- **Network load and timeouts**: the first and most evident drawback regards the time elapsed for each request. Interposing a proxy between the browser and the server doubles all network connections, since the browser will have to ask the proxy, which in turn will ask the server, for every resource; this clearly slows down the response time and doubles the risk of timeouts. Furthermore, the proxy will be consulted not only for HTML and XML documents, which is appropriate, but also for every other web resources, such as images, animations, JavaScript and CSS files, not to mention binaries and downloads of all sorts. It is true, on the other hand, that smart proxy implementations exist that reduce proxy-caused latency in HTTP response-request pairs. In particular, special configuration files (using Proxy Auto Config format [Pac96]) can be specified to have the browser select different proxies (or no proxy at all) depending on properties of the URL to be requested, such as the presence of the strings ".xml" or ".html" at the end of the address. But this solution can give problems, too, for three different reasons: first, PAC is not a standard but a proposal by Netscape silently accepted by some (but not all) other browsers; second, writing PAC files requires a certain technical skill and should not be approached light-heartedly; third, and more importantly, the use of PAC files is in conflict with the general WWW principle of URI opacity [23], according to which it is not correct to infer the nature of a resource by inspection of its URI. So an URL ending with the string ".html" does no guarantee that the resource actually **is** an HTML page, or more evidently, URLs with no extension at the end can be of any data type: the HTTP protocol does not constrain the MIME Type to any element of the path component of the URL and the server is free to return any data in any format to any URL.
- **Proxy chaining**. A second problem with proxy-based solutions is that a proxy as an elective intermediary for content modification will not work in those situations were another proxy is being used (e.g., when the browser is behind a firewall), unless an explicit chain of proxies is set up and managed. This is not as easy as it seems: the problems associated with proxy chaining need to be solved at the infrastructure level, i.e., convincing the managers of the closest proxy to specify the second one in the connection chain. This may pose some difficulties especially dealing with managers responsible of firewalls protecting large organizations or whole countries. All in all, using proxy for on-the-fly addition of content or links appears difficult and somewhat prone to malfunctions or restrictions policies.
- **Server-side parsing of bad HTML**. A third problem with proxy-based solutions is that the proxy needs to parse the

Deleted: is

Deleted:

Deleted: to

Deleted:

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0,25" + Indent at: 0,25"

original document in order to find the correct positions where the links are to be added. When documents are in XHTML or reasonable HTML, this is hardly a problem. Sometimes, though, the HTML is sufficiently badly formed that no simple approach is possible. Of course browsers are sufficiently sophisticated to be able to deal with rather bad HTML code, and quite often the document has only been tested to display correctly, and not to be correct according to the HTML specifications. A number of tools exist to inspect and add to a bad HTML document using a server-side language. For instance, in XLinkProxy we systematically converted the original HTML into XHTML by means of HTML Tidy, the handy utility by Dave Raggett [22]. Unfortunately, not all existing HTML documents can be amended by HTML Tidy, so that in those situations we preferred not to add links, and to leave the document untouched. The same problem exist with many other server-side tools[I browser sono molto più forti: parsano di tutto].

4.1.2 Plain CGI applications

In section 5 of this paper we present XLinkZilla, a full-featured browser-based external linkbase management system completely supporting XLink and a small but relevant part of the XPointer language. In XLinkZilla, most of the XLink features discussed in section 3 are implemented by the browser module, and very little is left for server-side handling.

In fact, the XLinkZilla linkbase server is a very simple and small Perl application activated by plain HTTP requests. Its functions are limited to creation and deletion of linkbases, and searching and adding of new links in selected linkbases. A simple request for all links relevant to a given URI in a given list of linkbases is responded with an XML fragment containing all XLinks identified in the list of linkbases whose "from" locator is the one specified in the request.

The most evident disadvantage of CGI applications for external link management is that they give **no** solution to the issue of monitoring the navigation activities of the browser (and, for that matter, to all navigation-related issues but 3.1.2: finding all relevant XLinks from the active linkbases): CGI applications simply receive HTTP requests for XLinks and give lists of XLinks as the response to those requests.

Moving to the client all navigation-related issues means, more than anything else, that access to XLink linkbases becomes a browser-dependent feature and thus, by definition, needs to be implemented from scratch for every browser (or, at least, for every operating system in which the browser is run). This opens the possibility of ending up with different, non-interoperable implementations of the same functionalities, and of leaving in the cold the users of some browsers (or operating systems) for which the implementation is either impossible or deemed to be too much of a hassle.

[non è restrittivo e XLinkZilla non è inutile]

In the next section we give a few glimpse of the issues connected to client-side support for the navigation-related features that CGI applications provide no support for.

4.2 User interfaces

As mentioned, no XLink application could be acceptable with at least some user interface mechanisms to provide support for the creation of links, and the selection of the destination in multiple destination links. Furthermore, client-side solutions such as XLinkZilla, as described in section 5, also move client side most of the navigation-related features discussed in section 3.1, that also have to be dealt with in the user interface part.

The complex interaction between the main navigation activities and the (most likely sporadic) link-creation activities of the user leave (to our knowledge) three possible architectures for the implementation of the user interface mechanism:

4.2.1 Frames

In a frame-based implementation the main browser window is divided in two frames, respectively containing the XLink interface and the main HTML document being navigated to. Whenever the user navigates and selects links, the response is transformed into a frame request for which the requested page is one of the elements. This is the solution we selected for XLinkProxy, as discussed in [4], and an example is shown in fig. 1. The creation of the frameset within which the actual frames are shown is obviously done server-side and for every document, so it is naturally applied by the proxy-based linkbase server.

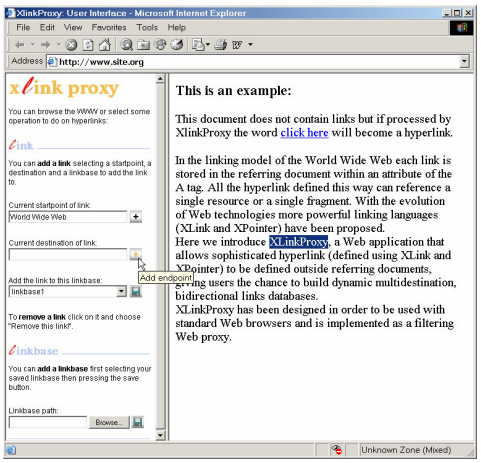

Fig. 1: the frame-based interface of XLinkProxy

One advantage of this solution is that more or less all browsers now support frames, and with some fiddling and tweaking of the JavaScript scripts necessarily involved in this solution it is possible to have the same implementation work on multiple browsers. But there are several drawbacks, too.

The first and most important drawback regards URL management. Suppose in the following that the linkbase server is available at the URL http://www.linkbase.org/app.cgi, the user interface is an HTML page available as http://www.linkbase.org/interface.html (in XLinkProxy it actually belonged to a different domain name, but for the moment we will not delve into this), and the main web page the user navigated to is http://www.site.com/index.html.

A simple implementation is to have the user access a URL like http://www.linkbase.org/app.cgi?u=http://www.site.com/index.html. The `app.cgi` application responds with a frameset document with the two individual frames pointing to the appropriate documents. This solution does not even require a proxy to be set up, since the URL specified for the main document frame can be a

Deleted:

Deleted: ¶

Formatted: Italian Italy

Deleted: n

transformation already of the index.html document with the appropriate XLinks added.

On the other hand, the URL now is significantly different than the one the user expected to see in the address box of the browser, and significantly harder to read. Furthermore, because of forbidden characters in the URI syntax, it would not actually look like the URL shown previously, but something much more unreadable and unwieldy such as http://www.linkbase.org/app.cgi?u=http%3A%2F%2Fwww.site.com%2Findex.html. Finally, all relative URLs of the main document (especially images) had to have their base explicitly specified.

A more sophisticated solution implies fiddling with the actual URL of the main document. XLinkProxy, that implemented this policy, was rather radical: whenever the browser requested the URL http://www.site.com/index.html, the frameset document was first returned as the response, and in the meantime the proxy had some time to request the actual document from the origin server and transform it by adding the XLinks. Then the main document was stored internally by the proxy with a local URL that had already been included in the frameset document. The browser would then request it as a URL local to the proxy server (so it will share the same domain name as the user interface frame, the usefulness of which will be clear in a moment), and of course relative URLs to images and other documents will also need to be identified and modified. The advantage of this is that the user perceives the page as actually *being* http://www.site.com/index.html, as requested, and that all URL fiddling happens behind the scene.

Unfortunately, this solution does not work well for documents that are framesets themselves, or where the relative URLs are created by computation (e.g. via a JavaScript script) rather than plainly available in the HTML code.

The solution of fiddling with the URLs has the added advantage of having the browser believe that the interface page and the main page belong to the same domain. In fact, it is necessary for the interface to access internal data of the main page (such as the current selection) in order to create new links. Security features in browsers, on the other hand, prevent pages belonging to different domains being accessed via JavaScript to the internals of each others' data. If, on the contrary, the main document appears to belong to its actual domain, it would be impossible for the XLink interface page to be able to determine the current selection in the main document page, and it would be impossible to create new links.

All in all, the XLinkProxy has taught us that the frame solution is unstable, not general, and rather complex, both client-side and proxy-side. Thus, although we used it for XLinkProxy, we now believe it to be inappropriate for a solid XLink application.

4.2.2 *External applications*

A different solution would be to create the XLink user interface as an independent application (i.e., a self-standing executable or a plug-in) to be installed on the client machine. Through inter-process communication APIs, this tool would be able to monitor the browser activities and send this information to the XLink server for further processing.

Two points are relevant here: whether the API for inter-process communication exists and how sophisticated it is (not really an issue for more modern browsers) and how would the modalities intrinsic to switching between applications interfere with the way the user worked with the tools: switching between different windows covering each other would most probably be perceived as to cumbersome for the XLink interface to be acceptable, so floating windows or other kind of solutions would be required in this case.

All in all, external applications seem to provide no further advantage to browser widgets which will be discussed in section 4.2.4.

4.2.3 *JavaScript additions*

The effect of the floating window obtainable with the external application can also be obtained by having the proxy add some JavaScript scripts to the requested document. These scripts would then create all the required interface elements, including a floating window where the widget for the creation-related features would be displayed.

In part, both XLinkProxy and XLinkZilla use this approach, although limited to the interface elements necessary to select among the destinations of a multiple destination link.

In XLinkProxy, in fact, a JavaScript pop-up menu (i.e., the JavaScript Menu Component by Gary Smith [21]) is added by the proxy to every multiple link in the page and it is invoked via the onclick event of the A element. The resulting interface element is shown in fig 2. XLinkZilla actually adds some XUL code to the link, as will be discussed in section 5.

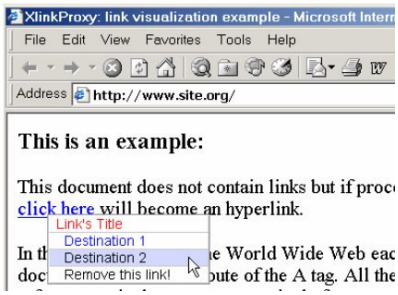

Fig. 2. a pop-up menu for a multiple destination link in XLinkProxy

Although neither XLinkZilla nor XLinkProxy insert the full XLink interface into the requested document, it would in fact be easy to do so: before displaying the document in the main window it would be possible to attach the JavaScript code (which could reside within a different document) to the DOM of the main document, and have the `onLoad()` event of the main window activate the required interface elements.

We have decided against this solution for two main reasons: first of all, it acts invasively on the internals of the document, and it may be cause problems when interacting with pre-existing JavaScript code in the main document; second, the solution relying on browser widgets is just as easy to provide, and much less invasive and uncertain.

4.2.4 *Browser widgets*

Recent versions of both major players (Microsoft Internet Explorer and Mozilla Navigator) can be extended with new user defined interface objects such as toolbars, sidebars, menu items, etc. These items have full access and control on the document and

Deleted: s

Deleted: to

Deleted: complicated

Deleted: provide

the URL shown in the main browser window. Mozilla requires some familiarity with XUL [30], the language internally used to describe skins and other interface-related aspects of the browser, and some JavaScript; Internet Explorer, on the other hand, actually requires some COM programming [32] to provide basic access to the internals of the main document.

Returning to our discussion, this means that it is possible to exploit these extensions to give the users an interface to select anchors, send links to a server, update linkbases and so on. Residing directly in the browser, these objects do not have problems of communication, interaction and security as mentioned in the previous section; furthermore, not being within the document, they do not act invasively and do not interfere with existing scripts in the document itself. The only drawbacks we have found are of course the fact that they are specific to a single browser platform, and that they require an explicit installation for the user to be able to make use of its features.

[E' veramente un problema? Trade off.]

In conclusion, we have found that browser widgets are easy to create, flexible to manage and lacking the limitations that other solutions have to face. For this reason, XLinkZilla has been implemented using this approach.

5. XLinkZilla: a client-side XLink module

XLinkZilla is a client-side application running on Mozilla that provide users an environment for creating and navigating on external links on normal Web pages. XLinkZilla is not based on a proxy, as XLinkProxy, but it relies on a browser addition that realizes the steps described in the section 3, getting through its task with mostly client-side code. XLinkZilla relies on a very simple CGI application (written in Perl) for server-side support (such as linkbases storage and querying).

As mentioned, Mozilla allows customization in its interface elements, such as sidebar, toolbars, menu items, etc., that can be used for our purposes. Few hundred lines of XUL [30] allow the creation of an application that provides some interface elements in a sidebar, monitors the URL of the document loaded in the main browser window and has a full control on its DOM.

Installation and configuration is easy, since one only has to drop an auto-install package on an open Mozilla window. Automatically a sidebar is added to the Mozilla interface, a couple of new commands are added to the right button and the application starts to monitor the navigation being performed on the main browser page. In figure 3 we show the main XLinkZilla sidebar, as well as the right-button command to create XLink locators out of the fragment currently selected in the main window.

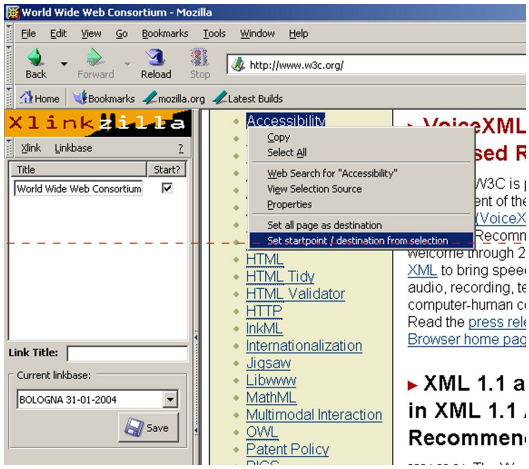

Fig. 3: the sidebar and menus of XLinkZilla

5.1 Navigating with XLinkZilla

Navigation features are provided by means of a hidden module, called XMonitor, that watches over the navigation activities of the main page, and calls a registered handler in the sidebar whenever a new URL is loaded in the browser (see section 3.1.1). The XLinkZilla sidebar contacts (with independent HTTP requests) all currently active linkbases, querying for links originating from the URL being loaded in the main browser (see section 3.1.2). URLs are compared in their canonical form, as discussed in [31], according to which the URLs http://www.site.com/, http://www.site.com (no trailing slash) and http://www.site.com:80/ (explicit mention of default port) are considered equivalent, but http://www.site.com/index.html is not.

Many parallel and independent HTTP requests are performed: while the browser is loading the actual document from the origin server, the sidebar verifies with all XLinkZilla servers the existence of appropriate links. This architecture saves much of the time we would have spent had we used a proxy architecture: network connections are still doubled, in fact, but they are executed in parallel. Furthermore the sidebar does not interfere with the normal navigation activities (as a frame, for instance, would do) and gives no problem in the management of URLs.

This architecture also allows for progressive display of links: the visualization of the main web page, in fact, is performed independently of the addition of external links. As soon as the links are collected, in fact, they are added on-the-fly to the DOM of the document, even if it is already shown on the browser.

Each XLink (as downloaded from the linkbase servers) is transformed into HTML fragments as discussed in section 3.1.3 by means of appropriate JavaScript modules in the XLinkZilla sidebar, and added to the DOM of the document (see section 3.1.4). External links are shown with a different style than plain links (a light yellow background for start points, a light blue-green background for destinations), to let the user understand the links are not originally from the document being shown.

XLinkZilla does not require the document to be well formed according to the XML syntax: while XLinkProxy had to rely on

Deleted: to

Deleted: the implementation of XLinkZilla has been brought forth

Deleted:)

Deleted: A f

Deleted: s of

Deleted: s

Deleted: are transformed

Deleted: ,

HTML Tidy [22] to obtain a well-formed document for the XLink to work, XLinkZilla relies on XPointerLib [28], an internal library of Mozilla that can solve XPointers within any web page. This is a great advantage over XLinkProxy, which could only manage a limited set of real web documents, while XLinkZilla can deal with all the documents that can be displayed by the Mozilla browser. [ripetere l'osservazione sui browser più potenti??????????????]

In order to manage links with multiple destinations (see section 2.1.5), a XUL menu is added to the contextual (right mouse button) menu, invoked by the `onFocus()` event of the corresponding A element, as shown in fig. 4.

Fig. 4: the contextual menu for multiple destinations in XLinkZilla

5.2 Creating a new XLink with XLinkZilla

Creating a new link requires selecting the "New Link" command in the sidebar XLink menu. The main section of the sidebar is cleared, and all new locators will be added to the new link.

The user selects a text fragment on the currently displayed page, and by opening the contextual menu (see figure 3) selects the command "set startpoint/destination from selection". The sidebar, called by the appropriate event handler of XMonitor, transforms the information of the current selection into an XPointer, by means of the same XPointerLib [28] mentioned previously, and shows it in the main sidebar section.

The user then may freely navigate on the web. Once a new interesting fragment is found, the selection is taken and transformed into an XPointer as before. As many locations can be selected and added to the current link. Once all necessary link information are inserted, the XLink is saved on the selected linkbase by clicking on the "Save" button.

[Anche link su più pagine, quindi guided tour e viste personalizzate??????????????]

By default, XLinkZilla proposes the first location as the start point of the link, and all the subsequent locations as destinations. This can be modified by selecting the appropriate checkboxes in the main section of the sidebar.

An additional dialog box can be reached by a contextual menu item associated to the locator line, allowing the user to select advanced options for the XLink, as shown in fig. 5. XLinkZilla implements and supports all XLink features, including the "show" and "actuate" parameters. These control when the traversing of the locator should take place, and whether the traversal should substitute the currently displayed document, create a new window or embed the destination fragment within the source document.

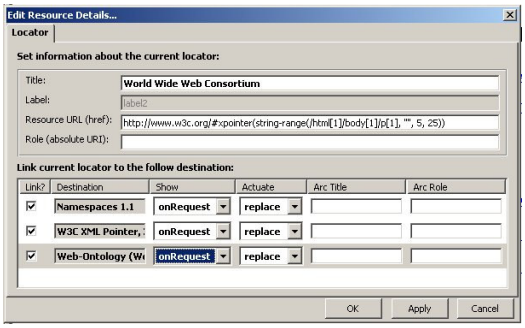

Fig. 5: the XLinkZilla dialog box for advanced link settings

The XLinkZilla sidebar has been tested on a number of Mozilla implementations, in particular on Windows, Mac OSX and Linux. The XLinkZilla server, written in Perl, has been tested on Linux (with the Apache HTTP server) and Windows (with both IIS and Apache as HTTP servers). Both applications are rather stable and reliable, and can be downloaded at the URL: http://xpointer.web.cs.unibo.it/.

Conclusions

[Che innovazione portiamo??]

XLink-based applications still have to build a market form themselves, and surely have a long way before they become mainstream and widely available. One of the problems they have to face is providing a powerful, yet easy to master interface for the kind of innovative linking functionalities that it aims to provide.

In this paper we have examined a number of services that any XLink linkbase application should aim to provide, and a number of possible implementation choices. The most important of these choices is surely the one between proxy and browser-based implementation. We have been through both types, with XLinkProxy first, and XLinkZilla now.

~~Without any doubt we tend to suggest browser-based architectures~~ for the ease of implementation, the simplicity of the interface mechanism, and the undeniable fact that at least some browser-based interface has to be supplied, so it may as well be worth implementing it all in the browser.

XLinkZilla our proposal for a browser-based XLink application, is a robust implementation of all XLink functionalities, and has been tested on a number of different operating systems (including Windows, Mac OSX and Linux). It can be downloaded and tested at the address http://xpointer.web.cs.unibo.it/.

The development of XLinkZilla is not finished yet. In the next months we will be adding an access control mechanism to the linkbases, and support for inline substitutions (as embedding of resource-type elements of the XLink link), as well as providing a linkbase management system for the Perl server-side application. But we are seriously convinced that these are just minor additions to a strong and already quite powerful tool.

Formatted: Normal, Space Before: 0 pt

Deleted:

Formatted: Italian Italy

Acknowledgement

We would like to thank Federico Folli, who co-authored the XLinkProxy engine, and helped in determining the limitations of that architecture and helped in shaping the new client-side architecture. We would also like to thank Marco Giovannini, who has authored an extension of the XPointerLib of Mozilla, that, although it is not being used in XLinkZilla, surely shed bright light on the use and quirks of the standard library distributed with the browser.

References

[1] Bieber M., Vitali F., Ashman H., Balasubramanian V., Oinas-Kukkonen H. (1997) "Fourth Generation Hypertext: Some Missing Links for the World Wide Web", *International Journal of Human-Computer Studies*, 47, 1997, pp. 31-65.

[2] Carr, L.A., De Roure, D.C., Hall, W., Hill, G.J., "The Distributed Link Service: A Tool for Publishers, Authors and Readers", in: *Proceedings of the Fourth International WWW Conference*, Boston, MA, The Web Journal 1(1), O'Reilly and Associates, 1995, pp. 647-656.

[3] Carr, L.A., W. Hall, S. Hitchcock, "Link Services or Link Agents?", *Hypertext 98 Proceedings*, ACM Press, pp. 113-122

[4] Ciancarini, P., Folli, F., Rossi, D. and Vitali F., "XLinkProxy: external linkbases with XLink", *Proceedings of the 2002 ACM symposium on Document Engineering*, ACM Press, 2002, pp. 57-65.

[5] Christensen B. G., Hansen F. A., Bouvin N. O. "Xspect: bridging open hypermedia and XLink", *Proceedings of the twelfth World Wide Web Conference*, May 20-24, 2003, Budapest, Hungary, pp.490-499.

[6] Davis, H.C., Hall W., Heath I., Hill G., Wilkins R., "Towards an integrated information environment with open hypermedia systems", *Proceedings of the ECHT 92 Conference*, ACM Press, pp. 181-190.

[7] Davis, H.C., "To Embed or Not to Embed…", *Communications of the ACM*, 38(8), 1995, pp. 108-189.

[8] Davis H. C., Rizk A., Lewis A. J. "OHP: A draft proposal for a standard open hypermedia protocol". In *Proceedings of the 2nd Workshop on Open Hypermedia Systems*, ACM Hypertext '96, Washington, D.C., March pp. 16-20.

[9] Davis, H.C., "Referential Integrity of Links in Open Hypermedia Systems", in: *Proceedings of ACM Hypertext '98*, Pittsburgh, PA, 1998, pp. 207-216.

[10] De Roure, D.C., N.G. Walker, L.A. Carr, "Investigating Link Service Infrastructures", *Proceedings of the Hypertext 2000 Conference*, ACM Press, pp. 67-76

[11] De Rose, S.J., Maler, E., Orchard, D., "XML Linking Language (XLink) Version 1.0", World Wide Web Consortium, Recommendation REC-xlink-20010627, 2001.

[12] De Rose, S.J., Maler, E., Daniel, R., "XML Pointer Language (XPointer) Version 1.0", World Wide Web Consortium, Candidate Recommendation CR-xptr-20010911, 2001.

[13] Fujitsu laboratory, "Fujitsu XLink Processor", 2003, http://www.labs.fujitsu.com/en/freesoft/xlip/index.html, last visited 6 February, 2004.

[14] Martin, D. and Ashman, H. "Goate: XLink and beyond". *Proceedings of ACM Hypertext '02*. 2002.

[15] Grønbæk K., Bouvin N.O., Sloth L., "Designing Dexter-based hypermedia services for the World Wide Web", *Hypertext 97 Proceedings*, ACM Press, pp. 146-156

[16] Grønbæk K., Bouvin N. O., Sloth L., "Open hypermedia as user controlled meta data for the Web".*Computer Networks*, (33), 2000, pp. 553-566.

[17] Maurer M. "HyperWave — The Next Generation Web Solution". Addison-Wesley, Reading, Massachusetts, 1996.

[18] Ladd B., Capps M., Stotts D., "The World Wide Web: what cost simplicity?", *Hypertext 97 Conference proceedings*, ACM Press, pp. 210-211

[19] Millard D. E., Moreau L., Davis H. C., and Siegfried Reich. "FOHM: a Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains". In *Proceedings of Hypertext 2000*, pp. 93-102.

[20] Rizk A., Sutcliffe D., "Distributed Link Service in the Aquarelle project", In *Hypertext '97 Proceedings*, ACM Press, pp. 208-209

[21] Smith G., "The JavaScript Menu Component, Creating Cross-Browser Dynamic HTML Menus", http://developer.netscape.com/viewsource/smith_menu/smith_menu.html, last visited 6 February 2004.

[22] Raggett D., "HTML Tidy Library Project", http://tidy.sourceforge.net/, last visited 6 February 2004.

[23] Jacobs, I., "Architecture of the World Wide Web", First Edition, W3C Working Draft 9 December 2003, http://www.w3.org/TR/webarch/.

[24] Vitali F., Folli F., Tasso C. "Two implementations of XPointer", in *Hypertext 2002 Proceedings*.

[25] Weinreich, H., Obendorf, H. and Lamersdorf, W. "The look of the link – Concepts for the user interface of extended hyperlinks". Proceedings of ACM Hypertext 01. pp. 200.

[26] Empolis Hungary, "X2X, the XML XLink Engine", http://support.eqnet.hu/~empolis/products_x2x_en.html, 2002.

[27] Systemwire, "XLinkit", 2004, http://www.systemwire.com/xlinkit/

[28] Mozilla Organization, "XPointerLib", 2004, http://xpointerlib.mozdev.org/

[29] Nentwich C., "XTooX", http://www.xlinkit.com/xtoox/index.html

[30] Mozilla Organization, "XML User Interface Language (XUL)", 2003, http://www.mozilla.org/projects/xul/

[31] Berners-Lee T., Fielding R., Masinter L., Uniform Resource Identifier (URI): Generic Syntax, Internet Draft, 16 February 2004, http://www.ietf.org/internet-drafts/draft-fielding-uri-rfc2396bis-04

[32] Microsoft Corp., "Browser Extensions", MS Developer Network, 2004,

Deleted: :

Deleted: , 2000.

Formatted: Justified

Deleted: ages

Formatted: Justified

Formatted: Justified

http://msdn.microsoft.com/workshop/browser/ext/overview/overview.asp

[33] Röscheisen, M., Mogensen, C. and Winograd, T., Beyond browsing: Shared comments, SOAPs, Trails, and on-line communities, in *Proceedings of the Third International W3 Conference, Darmstadt*, Apr. 10-14, 1995, pp. 739-749,

Formatted: Justified, Indent: Before: 0", Hanging: 0,25", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0,25" + Indent at: 0,25"

Deleted: ¶