Processing Link Structures and Linkbases in the Web's Open World Linking *

François Bry
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
francois.bry@ifi.lmu.de

Michael Eckert
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
michael.eckert@stud.ifi.lmu.de

ABSTRACT

Hyperlinks are an essential feature of the World Wide Web, highly responsible for its success. XLink improves on HTML's linking capabilities in several ways. In particular, links after XLink can be "out-of-line" (i.e., not defined at a link source) and collected in (possibly several) linkbases, which considerably ease building complex link structures.

Regarding its architecture as a distributed and open system, the Web differs significantly from traditional hypermedia systems. Modeling of link structures and processing of linkbases under the Web's "open world linking" require rethinking the traditional approaches. This, unfortunately, has been rather neglected in the design of XLink.

Adding a notion of "interface" to XLink, as suggested in this work, can considerably improve modeling of link structures. When a link structure is traversed, the relevant linkbase(s) might become ambiguous. We suggest three linkbase management modes governing the binding of a linkbase to a document to resolve this ambiguity.

Categories and Subject Descriptors

H.5.4 [**Information Systems**]: Hypertext/Hypermedia—*Navigation*; I.7.4 [**Document and Text Processing**]: Electronic Publishing

Keywords

Hyperlink, XLink, Link Modeling and Processing, Linkbase

1. INTRODUCTION

Hypertext links are an essential, if not the most important feature of the World Wide Web. They make it possible to build structures relating contents and/or pages that might be seen as a Web counterpart to paper books. Link structures are used on almost all Web sites and Web-based applications: company and organization presentations, community sites, catalogs, Web stores, etc. Undoubtedly, much of the Web's success is due to HTML's linking capabilities, in particular the capability to link from some content to *any other* content on the Web. One might say that the Web supports an "open world linking."

Such an "open world linking" makes the Web significantly different from Open Hypermedia Systems (OHS) and systems built after the Dexter Hypertext Reference Model [15]. In general, an OHS assumes that all resources, i.e., documents or document parts, an application might refer to, are known in advance and addressable through unique identifiers. One might say that OHS assume a "closed world linking." Thus, OHS are not adapted to the open world linking of the Web.[1]

XML links, as specified in the XLink recommendation, improve on HTML's links in several ways. In particular, a link after XLink can be "out-of-line," i.e., specified in another document than its source. Out-of-line links make linkbases, i.e., specifications of links outside the data (Web pages or content) the links refer to, possible. In turn, linkbases considerably ease modeling complex link structures as encountered in context-adaptive systems (e.g., teaching systems) or in versatile information systems (e.g., eCommerce catalogs). For this reason, most OHS offer out-of-line links and support linkbases. For a simple processing of linkbases they exploit the fact that all resources the links of an application might refer to are known in advance and have unique identifiers. As a consequence, the processing of out-of-line links and linkbases of OHS is not applicable on the Web.

This article is devoted to processing out-of-line links and linkbases under the open world linking assumption of the Web. This issue has been quite neglected: While advanced linking constructs have been proposed with XLink in a proposal that has reached the status of a recommendation already more than four years ago, the processing of these constructs under the conditions of the Web, i.e., its open world linking, has not received much attention.

*This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

[1]OHS have been called "open" because they aim at interlinking documents from arbitrary applications, requiring minimal or no hyperlink support from the applications.

This article first shows how the modeling of link structures can be considerably improved by adding a notion of "interface" to XLink. Then it shows that, as linkbases are currently defined, the relevant linkbases might become ambiguous as a link structure is traversed. Three complementary modes for the binding of a linkbase to a document are proposed —"transient," "permanent," and "temporary"— as well as an algorithm for processing these bindings under the open world linking of the Web. With this approach, the relevant linkbases are never ambiguous and the variety of behaviors required by Web applications can nicely be achieved. These claims are demonstrated on three usage scenarios: "guided tour and their composition," "superimposed guided tours," and "linkbase-aware bookmarks."

This article consists of 6 sections. Section 1 is this introduction. We briefly review linking on the Web with HTML and XLink in Section 2, and contrast it with linking in Open Hypermedia Systems in Section 3. Section 4 is concerned with providing interfaces for link structures. Section 5 explains the need for linkbase management and introduces our approach to linkbase management. Research perspectives in Section 6 conclude this article.

2. LINKING ON THE WEB

2.1 Links with HTML

HTML Hypertext links are specified within their origin using an anchor element. E.g., the following anchor element binds its content (denoted as ...) to the Hypertext '05 page:

```
<a href="http://www.ht05.org/"> ... </a>
```

HTML also has non-Hypertext links that might be called "conceptual links." Such links are specified within the header section of HTML documents using empty link elements such as:

```
<link rel="stylesheet" href="my_rendering.css" />
<link rel="alternate" hreflang="ja"
      href="index_ja.html" />
<link rel="prev" href="page4.html" />
<link rel="next" href="page6.html" />
```

In its current versions (HTML 4.01 and XHTML 1.1), HTML supports 15 types of conceptual links. Current user agents such as browsers usually interpret only conceptual links to style sheets.

HTML links, are they Hypertext or conceptual, are all:

- *inline*, i.e., specified within the HTML documents that are their origins (i.e., contain the link source);
- *uni-directional* and *outbound*, i.e., each link points to only one destination which can be remote from the link's origin;
- *restricted in their types*, as HTML supports only a fixed set of 15 link types (e.g., `stylesheet`, `next`) permitting no associations with meta-data.

```
<xlink xlink:type="extended"
       xmlns:xlink="http://www.w3.org/1999/xlink">

  <res xlink:type="resource" xlink:label="Res1">
     Content of a <em>local</em> resource
  </res>

  <loc xlink:type="locator" xlink:label="Loc1"
       xlink:href="http://example.com" />
  <loc xlink:type="locator" xlink:label="Loc2"
       xlink:href="http://www.w3.org" />

  <arc xlink:type="arc"
       xlink:from="Res1" xlink:to="Loc2" />
  <arc xlink:type="arc"
       xlink:from="Loc2" xlink:to="Loc1" />
</xlink>
```

Figure 1: Extended link in XLink's XML syntax

2.2 Links with XLink

This section briefly recalls salient features of XLink relevant for this work. For a more detailed explanation of XLink, refer to the W3C recommendation [9]. The following shift in terminology is worth stressing:

HTML-flavor links, i.e., single-source, single-destination hypertext links specified in the source document are called *simple links* in XLink.

An *extended link* is a collection of *arcs*. An XLink arc is a single-source, single-destination hypertext "link," not necessarily specified in the source document. Note that a simple link corresponds to an extended link that contains exactly one arc and is specified at this arc's source.

Simple links are subject to the same limitations as HTML links (cf. 2.1). Extended links overcome these limitations in several ways and can be:

- *out-of-line*, i.e., placed in external documents called linkbases (note that a linkbase usually does not contain the link sources);
- *multi-directional*, allowing not only for *outbound* arcs, i.e., arcs defined at their source (as in HTML), but also *inbound* arcs, i.e., arcs defined at their destination, and *third-party* arcs, i.e., arcs defined separately from their source and destination;
- *typed* and *associated with meta-data* freely. This covers both, machine-readable information in the form of a Uniform Resource Identifier (URI), and human-readable information in the form of XML text. Note that such an association with meta-data is a promising approach for Semantic Web applications.

XLink can be incorporated into arbitrary XML vocabularies. Any normal XML element can become a linking element by carrying the `type`-attribute from the XLink-namespace (`http://www.w3.org/1999/xlink`, assumed to be bound to the prefix `xlink:` in the following). All linking information is provided in attributes; element tag names have no XLink-specific meaning and can be chosen according to an application's needs. One also says that XLink is an "enabling vocabulary."

Extended Links are denoted by having the attribute `xlink:type` set to `extended`. Figure 1 depicts an example in XLink syntax. An extended link associates a number of *participants* (XML documents, fragments of XML documents, or any other kind of Web resources) using *arcs*.

Arcs pair up two participants in a directed relationship. One participant is designated as starting point of link traversal (source), the other as ending point (destination). In this respect arcs express hyperlink navigation as known from HTML's hyperlinks. Unlike HTML however, a (sub)resource, e.g., a word in an XML text, can be the source of several arcs. Arcs are denoted by `xlink:type="arc"`.

Arcs can carry attributes expressing *when* the hyperlink is followed (on user request or automatically) and *where* the destination is displayed (replacing the source, in a new window, or embedded). These attributes are very important in practice, but are not relevant to this work.

Participants of a link are resources such as XML documents or fragments of XML documents, but also pictures, video or any other data. The resources can either be *local*, that is, XML data contained in the extended link itself and surrounded by the *resource* element (`xlink:type="resource"`), or *remote*, that is, Web resources identified by URI references given in a *locator* element (`xlink:type="locator"`). To identify fragments of XML documents, XPointer expressions [13] can be used.

Separation of content and link structure is possible with XLink. By using only *third-party arcs*, i.e., arcs between remote resources, an author can strictly separate content and link structure: content resides in one XML document while a linkbase (which is also XML) provides the links.

All linking elements can be associated with meta-data by assigning them a *role* in the form of a URI. For arcs this is done using the `xlink:arcrole`-attribute. There is only one predefined role for arcs in XLink; it is used to locate linkbases. We will discuss this arcrole in Section 5.1.

Keep in mind that in XLink the important navigation information —the "hyperlinks"— is given by the arcs. The extended link element (`xlink:type="extended"`) is only a mere container holding the participants and arcs. This also means that two resources can participate in the same extended link, though one cannot navigate from one to the other. In the example of Figure 1 this is the case for `Res1` and `Loc1`, as there is no arc specifying a "hyperlink"/"hypertext navigation" between them.

In the remainder of this work, we will explicity speak of "extended link" and "arc" when referring to XLink linking elements. When speaking of hyperlinks in a more general fashion that is not XLink-specific, we will use the term "link" or "hyperlink."

2.3 Other Linking Technologies

HTML and XLink are not the only Web languages that allow to specify links or other relationships between Web resources. Other languages include the Resource Description Framework (RDF), Topic Maps, or the Web Ontology Language (OWL). They share with XLink the ability to specify relationships out-of-line, i.e., outside the resources involved in a relationship. In fact, XLink's arcs can be directly mapped into RDF triples [7].

While we present our ideas as extensions to XLink in this work, we suggest that they are also transferable to these other Web languages. XLink's special arc-role to locate linkbases (which we will meet in Section 5) even has an equivalent in RDF, the `rdfs:seeAlso`-property, and the two share the same problems.

3. OHS VS. WEB

XLink's main advantage over HTML links lies in the out-of-line links, which allow a separation of content and links. It can significantly simplify modeling and managing complex link structures.

Separation of content and links allows simple but effective user adaption of hypermedia documents: by applying different linkbases to the same document, one can display the same text with different hyperlinks, depending on the needs of the individual reader. But different linkbases not only provide adaption of the navigational structure (the visible hyperlinks): Style-sheets, which can provide adaption of Web documents w.r.t. their presentation, are usually associated with a document by means of links (cf. 2.1). Thus, the choice of the linkbase influences the choice of the style-sheets and so in turn the presentation of a document.

Out-of-line links also allow to add links (both inbound or outbound) to resources in file formats without native hyperlink support, e.g., images in GIF, PNG, or JPEG format. With an appropriate XPointer-scheme, out-of-line links could thus replace HTML's rather awkward image map mechanism [18, section 13.6].

The advantages of a separation of content and links, as possible with XLink, are by no means new. It has been realized in many before-web Hypermedia Systems such as Sun's Link Service [17], Microcosm [11], and is a matter of course in the OHS community. XLink's expressiveness is similar to that of the Open Hypermedia Interchange Format [12] and the Fundamental Open Hypermedia Model [16], though these are somewhat better targeted towards hypermedia and provide also higher structuring mechanisms [6].

However, the mechanisms used to process links and linkbases in OHS are not directly applicable to Web. We will now outline important characteristics of the Web, contrast them to OHS, and discuss how the differences influence linking on the Web.

3.1 Decentralized and Open World

The Web is designed as an open, distributed and decentralized system. As long as one has access to some portion of namespace, one can add new resources at will; no registration or announcement is needed. The Web works without any central entities, there are no central registries having knowledge of all resources, nor central repositories.

In OHS, in contrast, all resources are known to some central repository that also stores all links. Finding the relevant links for some given content is a simple matter of querying the repository. There have also been approaches to use directory services to provide a distributed link service infrastructure (DLS [8]).

However, central repositories or directory services are against the Web's philosophy: they can lead to scalability problems and also pose non-technical issues such as a suitable price structure for their usage and accounting.

Finding out-of-line links on the Web cannot be a simple matter of querying one (or several) server(s) *known in advance*. Instead, collections of out-of-line links (linkbases) should be treated as normal Web resources, i.e., be retrievable just like normal documents using HTTP. Consequently, links have to be found and collected as we traverse a number of hypertext documents. This is an important motivation behind our proposal of linkbase management laid out in Section 5. Note that linkbases can be dynamically generated resources in the same way hypertext documents can. Thus treating linkbases as retrievable Web resources is no limitation; directory service-like infrastructures can be built "behind the scenes."

3.2 Uniform Resource Identifiers

Resources on the Web are identified by Uniform Resource Identifiers (URI) [3]. A resource does not necessarily have one unique URI, it can be accessible at various URIs, and its URI might change. There is, in general, no way to tell just from URIs itself whether two URIs point to the same resource or not. Also, when accessing some resource through one URI, this does not provide a way to find out all its other URIs.

In OHS, resources (and subresources) are typically identified by means of *unique* identifiers. This assumption is also inherent to the Dexter Hypertext Reference Model [15].

Our notion of interfaces introduced in Section 4 can ease problems faced when building link structure containing resources with URIs that are either unknown (e.g., the resource has not been created yet) or that might change in future (e.g., the resource is moved to a different server).

3.3 No Write Access

The Web is, from a user's perspective, a read-only system. Only the owner (usually the author) of a resource it allowed to modify it.

Virtually all OHS allow a user to add annotations or links to documents, or even restructure documents. A user's modifications are managed by the system (i.e., in the resource itself or again in a central repository) and can be shared with other users. This in essence, allows a user to become author.

This cannot be done that easily on the Web. Any modification a reader might want to apply, such as making an annotation, adding a link, or building his own reader's view of a page [4], cannot be stored in the resource itself. Instead the modifications have to be stored in a new resource to which the reader has write-access. This raises the issue of establishing a connection between the original resource and the resource containing the modifications. Allowing to establish such a connection without write-access to the original resource —i.e., supporting open world linking— is another important motivation behind Section 5's linkbase management.

3.4 Remark

While the Web's characteristics pose challenges on advanced linking mechanisms, it is worth noting that the Web also derives its greatest strengths from them: it is an open, flexible, heterogeneous, and highly fault-tolerant system providing access to an incredible amount of content.

4. INTERFACES FOR LINK STRUCTURES

In modeling a collection of semantically related documents, a hyperlink rarely comes alone. Instead, a number of links together generally impose some higher structure on the documents they connect. Examples of such link structures include:

- *guided tours*, sequencing a number of independent documents in one or several manners, hinting a reader an order in which they are best read, e.g., for eLearning purposes;
- *link catalogs*, supplying a document, e.g., a table of contents, with many outbound links;
- *glossaries*, linking occurrences of terms in a set of documents to their definitions in a glossary-like document;
- *cross-references*, ranging from simple cross-references to sections within an article or a book to complex cross-references, e.g., in code, API-documentations or meta-data annotations in Semantic Web applications.

Such structures can benefit highly from a separation content (document) and hypertext structure (linkbases) as possible with XLink. By applying different linkbases, a single document can be viewed in different contexts, e.g., stand-alone, as part of a guided tour, with links into a glossary, or enriched with personal annotations, depending on the reader's needs.

Hypertext structures easily suffer from dependence on the URIs of the link endpoints. When building, URIs have to be known in advance by the structure's author, and the structure has to be modified in case a URI changes. Interfaces, as presented here, decouple structure from a document's "physical" location (URI)[2] by providing an indirection.

Also, often the need to combine structures into composite structures arises [14]. Our primary example for this will be

[2]Arguably, there are URIs that do not provide a location for the resource they identify; however, for the hypertext documents considered in this work such URIs are unsuitable since the documents have to be retrieved to be displayed.

the combination of two guided tours into a composed guided tour being the succession of the smaller ones. By signaling well-defined junctures, our interfaces facilitate composition.

We can imagine hypertext space as a directed graph with documents as nodes and links (i.e., in XLink terminology the arcs) as directed edges. Our interfaces are nodes just like normal resources; they are however transparent to the user and will not be displayed. When a user traverses an arc inbound to an interface, instead of displaying it, an outbound arc will be traversed automatically.

Using the example of a guided tour, we will now first show how a link structure can define its interfaces. Then we will show how these interfaces can be referred to by other structures, in order to reuse an already defined structure in building a composite structure. The example provided will be the combination of two guided tours into a sequence. Finally, using the example of a university's course catalog, we will show how our interfaces decouple structure and physical locations aiding authoring, esp. collaborative authoring, and maintainability.

4.1 Defining Interfaces

A guided tour provides an order over a number of independent documents, e.g., chapters of an online book, by linking them in a chain. It is a common metaphor used in teaching systems or personal adaptive systems. Seen from an outside perspective, it features a start and an end. Instead of just using the first and the last document of the chain, we wish to expose the start and the end more explicitly: as interfaces.

A guided tour can be written in XLink as one big extended link containing locators to the documents as participants and arcs chaining the documents. Interfaces are defined inside the extended link as additional participants. An arc connects from the start interface to the guided tour's first document. Similarly, the last document is connected to the end interface by an arc.

As a conservative extension to XLink, the interface definitions are introduced as new linking elements, signified here by `xlinkext:type="interfacedef"`. We require interface definitions to carry an identifier (in `xml:id`) that is unique within the XML document containing the extended link. This identifier will be used when referring to interfaces (cf. 4.2).

The following gives a flavor of a definition of a guided tour (element tag names are arbitrary for sake of readability):

```
<guidedtour1 xlink:type="extended">
  <start xlink:type="other"
         xlinkext:type="interfacedef"
         xml:id="gt1-start"
         xlink:label="mystart" />

  <arc   xlink:type="arc"
         xlink:from="mystart" xlink:to="doc1" />

  <!-- ... -->
</guidedtour1>
```

Interface definitions are similar to local resources (`xlink:type="resource"`). Unlike local resources however, they are not displayed *as part of a document*, but *stand on their own* and are transparent to the user (i.e., not displayed).

4.2 Referring to Interfaces

Suppose now we have defined two or more guided tours and want to reuse them by sequencing them into one big composed guided tour. Using interfaces this can be done easily. The composed guided tour is another extended link, again having a start and end interface. Arcs provide connections from the end of one component tour to the start of the next component tour in the order.

The starts and ends of the component tours are interfaces as described in the previous section. Inside the composed tour's extended link we need a way to refer to these externally defined interfaces.

Analogous to interface definitions, interface references are introduced as new linking elements and denoted here by `xlinkext:type="interfaceref"`. A URI reference in `xlink:href` provides the interface location and name. The following illustrates the discussed example of a composed guided tour (tag names again arbitrary):

```
<composedguidedtour xlink:type="extended">
  <!-- ... -->
  <gt1e xlink:type="other"
        xlinkext:type="interfaceref"
        xlink:href="http://.../gt1.xml#gt1-end"
        xlink:label="gt1end" />

  <gt2s xlink:type="other"
        xlinkext:type="interfaceref"
        xlink:href="http://.../gt2.xml#gt2-start"
        xlink:label="gt2start" />

  <arc  xlink:type="arc"
        xlink:from="gt1end" xlink:to="gt2start" />
  <!-- ... -->
</composedguidedtour>
```

A composed guided tour modeled this way will automatically reflect changes that are made in any of its component tours, e.g., the insertion of a new document at any place (including the beginning or the end of a component tour). The only requirement for the component tours is to keep their interfaces stable.

4.3 Decoupling Structure and Locations

To show how interfaces can be used to decouple a link structure from the physical locations of the documents it interlinks, we will consider the example of a university's course catalog. The catalog lists all courses offered in a semester and links to their corresponding course homepages. The individual course homepages are maintained by different authors and hosted by the departments offering the courses. The authors of the catalog and the course homepages need to collaborate to create and maintain the links.

HTML linking requires the author of the course catalog to find out the URIs of all course homepages in order to link them. If a department changes the structure of its web site (i.e., the URIs change), the catalog author needs to be informed and modify the catalog accordingly to avoid broken links.

Using XLink's external linkbases can help a little: each department can offer a linkbase and the course catalog simply uses the linkbases. Now however, the linkbases must be aware of the catalog's URI and the locations inside the catalog where the links are to be anchored. Especially any restructuring of the catalog, e.g., splitting it into multiple pages or renaming it from `"current.xml"` to `"spring2005.xml"`, will break the linkbases of all departments.

Having well defined interfaces, as we suggest in this work, can distribute the responsibilities and ease collaborative authoring. In this respect, we believe that link interfaces can play a role in authoring hypertext as important as subsystem interfaces do in building software systems.

Instead of linking directly from the catalog to the course homepages, we can use interfaces to insert an indirection. An interface provides a symbolic name for a course homepage, thus making the links less vulnerable to changes in the physical locations of the documents. The authors only have to agree upon the names of the interfaces and the location of the linkbases.

4.4 Interfaces vs. HTTP Redirects and DOIs

An effect similar to the indirection our interfaces provide can be achieved by using "HTTP redirects." Redirects are a facility of the Hypertext Transfer Protocol (HTTP) [10] allowing a request for a Web resource to be answered not with the resource but an indication of a different location (URI) where to find the resource. In fact, this way of providing symbolic names has gained widespread use (e.g., [2]).

However, using HTTP redirects has serious drawbacks. Link authors require access to Web server configuration to create them and, more importantly, redirects work only well for destinations of arcs. It would be possible to define the start interface of a guided tour (cf. 4.1) using an HTTP redirect as the start is the destination of an arc in the composition. However, the end interface is the source of an arc in the composition. If a document with some given URI is currently viewed in the browser, we would need to resolve the URIs of the sources of all known arcs to find out if any of them is redirected to the given URI. Resolving a (possibly massive) amount of URIs causes unnecessary network traffic and is practically infeasible.

This problem is also shared with other approaches to symbolic identifiers on the Web, e.g., Digital Object Identifiers (DOI) [1]: While it is easy to locate a resource given its DOI, the converse does not hold.

Redirected URIs, DOIs, etc., are suitable to identify a single resource on the Web, but not to identify a conceptual access point, i.e., an interface, of a link structure. We are convinced that interfaces should be an essential part of a link structure

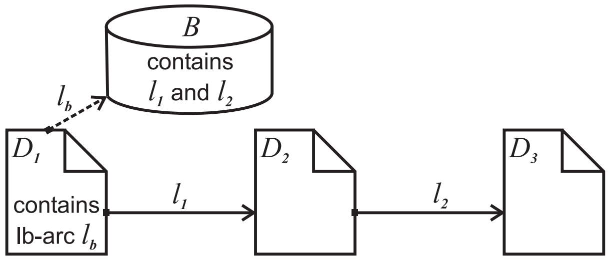

Figure 2: Ambiguity of linkbase-arcs in XLink

and defined on the level of the structure in linkbases.

5. LINKBASE MANAGEMENT

The separation of content and hypertext structure into document and linkbase(s) allows a document to be viewed in different contexts, e.g., as a stand-alone document, as part of a guided tour, with links into a glossary, or enriched with personal annotations, depending on the reader's needs.

However, this separation poses a new challenge: how can the linkbases relevant for a document be found in an open world scenario, and which of the found links should actually be displayed? The procedures defining this are referred to as *linkbase management* here.

5.1 Arcs to Linkbases after XLink

A peculiarity of XLink is that linkbases are found using arcs (called linkbase-arcs henceforth) with the special arcrole `http://www.w3.org/xlink/properties/linkbase`. Sensibly, this introduces no new constructs to XLink. However, it also introduces a circularity, as the linkbase-arcs underly linkbase management themselves.

To see how XLink's linkbase-arcs work, consider the situation in Figure 2. Linkbase B contains a link l_1 from document D_1 to D_2 and another link l_2 from D_2 to D_3. To display l_1 when a user is viewing D_1, his browser needs to be aware of linkbase B; there has to be some connection between D_1 and B.

Such a connection can be established with a linkbase-arc l_b from the document (D_1) to the linkbase (B). Given that D_1 is the first document a user opens, such a linkbase-arc has to reside inside D_1, or else there is no way for the browser to be aware of it.

The interesting question now arising is what happens if the user follows the link l_1 from D_1 to D_2, arriving at D_2 not containing a linkbase-arc to B. Will the link l_2 from D_2 to D_3 be displayed, or not? Both behaviors are conceivable, the XLink recommendation is underspecified in this respect.

Allowing a linkbase to be valid only for documents that refer to it directly (by means of a linkbase-arc residing inside the document) or indirectly (by means of chaining linkbases, cf. 5.4), requires a document to know all its linkbases. This renders imposing link structures such as making a document part of a guided tour impossible, if one does not have write access to the document or one of its linkbases. Also, once the document is part of a guided tour, it is not possible to view it as a stand-alone document without the guided tour links or use it in some other guided tour only with the links of that other guided tour.

Collecting links in a "greedy" fashion, i.e., never forgetting link information, on the other hand, allows a document to be presented with different links at different times. However, this can confuse and disorient users. By collecting links greedily, a document could end up containing many irrelevant or unwanted links and be hard to read. Also, storing a vast amount of links and processing them for presentation can mean high demands in memory and computation time.

These two approaches to linkbase management have been called *conservative* (as a document appears always the same) and *inflationary* (as more and more links are collected) in [5]. They are opposite end extremes, and it should be clear that neither of them is satisfying.

The linkbase management mechanism we propose in the following is based on three complementary modes — "transient," "temporary," and "permanent"— for the binding of a linkbase to a document or set of documents. The modes are set individually per linkbase, so a user agent (browser) can have several linkbases loaded, each in a different mode. We adhere closely the original ideas in XLink and use arcs with special arcroles to establish the connection between documents and linkbases. XLink's original linkbase-arcrole is replaced with three new arcroles that signify the mode used to load a linkbase:

```
http://www.pms.ifi.lmu.de/xlinkext/linkbase/transient
http://www.pms.ifi.lmu.de/xlinkext/linkbase/temporary
http://www.pms.ifi.lmu.de/xlinkext/linkbase/permanent
```

The linkbase-arcs with these arcroles will be referred to as `transient`-arc, `temporary`-arc, and `permanent`-arc.

5.2 Transient and Permanent Bindings

A linkbase loaded in the `transient` mode is bound to the current document only. It "lives" for the shortest time possible, i.e., only the current document. When a user leaves the document by traversing a link, the linkbase and all link information contained in it are discarded. Note that this includes any `transient`-arcs residing inside the linkbase. This mode should be used for linkbases known in advance to the document author.

A Linkbases loaded in the `permanent` mode is bound to the current document and *all* future documents. It "lives" the longest time possible: once it is loaded it stays in memory "forever" (more precisely, up to the end of the browsing session). This mode should be used for linkbases containing links we always want to see, such as a user's annotations or links into a glossary.

5.3 Temporary Binding

Our third mode, called `temporary`, is positioned between the extremes of living for the current document only (`transient`) or living forever (`permanent`). A linkbase is bound to a restricted set of documents, called its validity range. It remains valid (i.e., its links are available) while only documents from its validity range are viewed, and is discarded when this range is left. The linkbase "lives" for a limited time that ends when the user navigates to a document outside the validity range.

The validity range of a linkbase is expressed with `temporary`-arcs going from all the documents in a linkbase's validity range to the linkbase. When a user traverses a link to a new document, a non-permanently loaded linkbase is discarded, *unless* there is a `temporary`-arc from the new document to the linkbase. In contrast to the `transient`-arc, a `temporary`-arc does not have to reside in the document, but can also reside in the linkbase itself. This means that linkbase information provided by a `temporary`-arc is not immediately discarded when the linkbase it resides in is discarded.

The `temporary` mode is especially useful for building link structures interconnecting a number of documents, e.g., guided tours. The link information for a structure can be placed in a linkbase along with `temporary`-arcs declaring the linkbase's validity range. As long as the user stays within the documents that are in the validity range, they are displayed as part of the current structure (i.e., with the links from the structure's linkbase). Once the user leaves the structure by navigating to a document outside the validity range, the structure's link information is discarded and the documents function as stand-alone documents in case they are visited again.

5.4 Linkbase Processing

Having presented the functioning of our linkbase management modes as they need to be understood by link authors, we will now discuss the processing of linkbases under these three modes. Special consideration will be given to chaining of linkbases, i.e., a linkbase can be the starting point of a linkbase-arc, and dynamic linkbases, i.e., linkbases that are not static XML documents but are generated dynamically and should be retrieved newly every time a link is traversed.

Figure 3 shows a possible algorithm (in pseudo-code) for linkbase processing. The basic idea is that a browser maintains a set of the active linkbases (*activeLB*). The document currently being displayed is always a member of this set; initially, when there are no permanently loaded linkbases, it is the only member. Whenever a user leaves the current document by following a link to a new document (*newDocument*), the algorithm is run to update the set of active linkbases.

By retrieving linkbases that are destinations of linkbase-arcs —traversing linkbase-arcs— the algorithm grows the set of active linkbases. For a linkbase-arc (a) to be traversed, its source ($a.from$) must be a member of the set of active linkbases while its destination ($a.to$) is not. When a linkbase-arc is traversed, its destination is retrieved and becomes a member of the set of active linkbases. The retrieval can yield new linkbase arcs that need to be processed. As a linkbase-arc can have its source ($a.from$) in any of the active linkbases (not just the document to be displayed), linkbase chaining is supported.

In principle, computing this set of active linkbases is the

On leaving of the current document:

$lbArcs$:= {all `temporary`-arcs}
DISCARD all non-permanently loaded linkbases
RETRIEVE $newDocument$
$activeLBs := \{newDocument\} \cup permanentLBs$
$lbArcs := lbArcs \cup \{\text{all lb-arcs in } newDocument\} \cup \{\text{all lb-arcs in } permanentLBs\}$

while $\exists a \in lbArcs.\quad a.from \in activeLBs$ **do**

CHOOSE $a \in lbArcs$ WITH $a.from \in activeLBs$
if $a.to \in activeLBs$
$lbArcs := lbArcs \setminus \{a\}$
if $a.arcrole =$ `permanent`
$permanentLBs := permanentLBs \cup a.to$
endif
else
RETRIEVE a.to
$lbArcs := (lbArcs \cup \{\text{all lb-arcs in } a.to\}) \setminus \{a\}$
$activeLBs := activeLBs \cup \{a.to\}$
if $a.arcrole =$ `permanent`
$permLBs := permLBs \cup a.to$
endif
endif

end

Figure 3: Linkbase-Processing Algorithm

same as computing a reflexive, transitive hull in the graph where documents and linkbases are nodes and linkbase-arcs are edges. Note however that this graph is not known in advance but has to be explored: adding a new linkbase to the set of active linkbases can add new linkbase arcs to the graph that start or end in arbitrary (possibly previously unknown) nodes.

Once the set of active linkbases cannot grow anymore, all links have been discovered and the document is ready to be displayed. When the user later leaves the document by traversing a link, the set of active linkbases has to be computed again; its initial members are the new document and any permanently loaded linkbases.

This approach based on growing a set of active linkbases is open to dynamic linkbases. As the user navigates from one document to another, linkbases that stay active are not kept in memory; rather they are "reobtained," i.e., their content is retrieved newly from the web. Of course, caching can be employed to reduce network traffic for static linkbases.

To explain the algorithm in further detail, we will look at how the linkbase-arcs are treated differently according to their type.

Processing `transient`-arcs is straightforward. Linkbases loaded by means of a `transient`-arc are bound to the current document only. When a user leaves the current document by traversing a link to a new document, the linkbase and its links are discarded.

Processing `permanent`-arcs is similarly straightforward. Linkbases loaded by means of a `permanent`-arc apply to the current and all future documents. They are maintained in the set *permanentLBs* which is joined to the set of active linkbases (*activeLBs*) for every document.

For processing of `temporary`-arcs, recall their functioning: A linkbase that is non-permanently loaded is discarded when the current document is left, *unless* there is a `temporary`-arc from the *new document* (or any linkbase that becomes active for the new document) to the linkbase.

Technically, we give `temporary`-arcs a slightly longer "lifetime" than normal arcs. When a user leaves the current document, all information from non-permanently loaded linkbases is discarded. However, the `temporary`-arcs, are retained and available for traversal when the set of active linkbases grows. This longer lifetime is the only difference between `temporary`-arcs and `transient`-arcs. They can otherwise be treated the same.

Termination and Complexity

The algorithm presented here tolerates cycles and terminates, as long as there are no infinite chains of linkbases.[3] Of course, a practical implementation might want put further limits on the length of linkbase chains or the number of active linkbases. The algorithm's time complexity is $O(mn^2)$ where m denotes the number of active linkbases after execution and n the number of linkbase-arcs the algorithm is or becomes aware of.

5.5 Scenarios

To demonstrate the potential of our linkbase management mechanism, its usage will now be demonstrated in a three scenarios. The scenarios are chosen with an emphasis on the `temporary` mode, as this is the most interesting and useful.

5.5.1 Guided Tours and their Composition

As discussed above (cf. 4.1), a guided tour can be formulated as one extended link in a linkbase, separate from the documents participating. This linkbase has to stay active as long as the user views documents from the tour. We can achieve this by using `temporary`-arcs the following way: the linkbase contains a `temporary`-arc for each document (and interface) participating in the tour. Each arc goes from the document to the linkbase itself.

As the arcs reside inside the linkbase, this only ensures that the linkbase stays active once it has been loaded. If a user accesses the tour's first document, the linkbase will not be loaded, as this document does not contain an arc to the linkbase. Note that this behavior is actually intended, as we still want to be able to view the document as stand-alone, i.e., as not being part of the guided tour.

This issue is resolved easily: we need an additional document "offering" the guided tour. This document contains an arc

[3]Chains of infinite length are impossible with static resources, but a theoretical possibility with dynamic resources: picture, e.g., a query `http://example.com/query?param=`*n* with an integer value *n* as parameter that delivers a linkbase-arc to itself but with the parameter *n+1*. However, this causes no real problems since a simple restriction of chain length can be assumed in practical implementations.

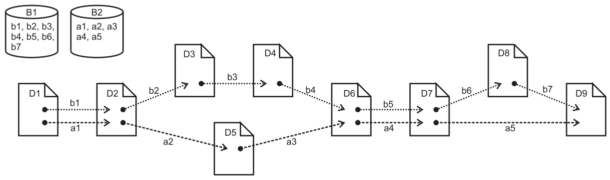

Figure 4: Superimposed guided tours

to the linkbase (`transient` or `temporary`, both work) and a link to the start of the guided tour (start interface or first document). Note that the document offering the tour can actually be the linkbase itself, as linkbases are arbitrary XML documents and thus can also contain information to be displayed (e.g., XHTML).

It would also be possible to realize the guided tour by loading its linkbase once at the beginning with a `permanent`-arc. This however has two drawbacks: Firstly, when the user decides to leave the guided tour (e.g., by following a link leading outside the tour), the tour's linkbase stays in memory, even though it is not needed anymore. Secondly, there is a problem if we allow composition of guided tours as in Section 4.2.

For this, consider a document being part of two different guided tours that also apply different annotations to this document. In order to compose these tours we need to distinguish whether the document is displayed in the context of the first or the second tour. When we load the linkbases in `permanent` mode, the document will display the links from both tours at the same time, thus not distinguishing the context. In particular, a user has two "next document" links available and can accidentally shortcut the composite guided tour or run in circles by selecting the "wrong" link.

Using `temporary` mode, this is not an issue. The first guided tour's linkbase is discarded before the second guided tour's linkbase is loaded.

5.5.2 Superimposed Guided Tours

Guided tours can even be superimposed on each other: Figure 4 depicts a situation where two guided tours are available: say one covers basic material (arcs bn in linkbase B1, drawn dotted in the figure) while the other covers more advanced material (arcs an in linkbase B2, drawn dashed) of the same subject. The guided tours share some documents. Superimposing the guided tours makes the arcs from both tours available to the user and he can jump freely between the basic and the advanced guided tour.

Making use of linkbase chaining, we put `temporary`-arcs from B1 to B2 and from B2 to B1 into a third linkbase B3. This way, both linkbases stay active as long as documents from either of them are viewed. Of course, this linkbase B3 has to take care that itself stays active, so additionally an arc from B1 to B3 or from B2 to B3 is needed. Alternatively, B3 could be loaded permanently.

Superimposing is not limited to guided tours. In fact, any link structures can be superimposed using the linkbase management mechanism proposed in this work. We can thus use many structures in parallel, e.g., enriching a document with annotations from various sources or enriching documents in guided tours with cross-references. This illustrates that the mechanism suggested here is fairly general and powerful.

5.5.3 Linkbase-aware Bookmarks

Applying different linkbases to the same document allows us to view this document in different contexts, e.g., as standalone, as part of a guided tour, enriched with annotations, or combinations thereof. When bookmarking such a document we should not only bookmark its URI, but also remember its context, i.e., the linkbases applied.

A bookmark can be an extended link containing a local resource providing a title for the bookmark and a locator to the bookmarked page; an outbound arc connects the two. To make our bookmark linkbase-aware, we need additionally locators to the linkbases and linkbase-arcs.

When the bookmark is created, some linkbases are active non-permanently and some permanently. Permanently loaded linkbases appear in the bookmark extended link as destinations of `permanent`-arcs, non-permanently loaded linkbases as destinations of `transient`-arcs; both types of linkbase arcs originate from the bookmarked page.

This however does not suffice, yet. A `temporary`-arc from the bookmarked page to the document containing the bookmark is needed. This arc is essential; if it does not exist the following happens when the outbound arc of the bookmark is traversed: the link information from the bookmark document is discarded and with it the linkbase-arcs are lost. When the bookmarked page is then displayed, the linkbases are *not* loaded.

This need for the additional arc might seem a little peculiar at first. It illustrates a basic principle behind linkbase management with linkbase-arcs as used in this work: Linkbase-arcs have arbitrary resources (e.g., documents) as sources and linkbases as destinations. A linkbase-arc is followed, i.e., the linkbase it points to becomes active, when the resource it originates from is loaded.

We believe that other, complementary means of linkbase management that are not based on such linkbase-arcs could simplify modeling of some link structures such as the linkbase-aware bookmarks. In particular, we hope to investigate in future mechanisms where the following of a (navigational) hypertext link *directly* triggers changes in the browser's active linkbases. In the current mechanism this is not the case: changes in active linkbases are triggers directly by the loading of resource (which only in turn was triggered by the user following a hypertext link to it).

6. CONCLUSION AND PERSPECTIVES

Our work is motivated by allowing reuse and composability of structures under an open world linking assumption. A document can be seen in different contexts, i.e., as part of different link structures. It should be possible for this context to be supplied not only by the author but also users who have no write-access to the original document.

Open world linking on the Web differs significantly from closed world linking in other hypermedia systems: The Web is completely decentralized; information should be hosted in Web resources, no assumptions should be made about central institutions or even communicating distributed institutions such as directory services. The Web uses URIs as identifiers and these might not be unique or a resource's URI might change. Users of a Web resources have, in general, no write-access to a given resource; any modifications or additions to the resource need to be stored separately in a new resource.

This article has introduced the concept of interfaces for link structures and a linkbase management mechanism based three binding modes. We have shown how XLink can be extended conservatively, i.e., only additions to the language are made, to accommodate both ideas.

Undoubtedly, there are many other methods of aiding reuse and composability. Especially different linkbase management mechanisms are easily imaginable. We hope to investigate these in future.

While our ideas have been presented as extensions to XLink, we suggest that the raised issues and the presented solutions are also relevant for other Web and Semantic Web languages, e.g., RDF, Topic Maps, OWL. Especially, the connection of data and meta-data spread over different Web resources is still an underexplored and maybe neglected issue in Semantic Web research deserving further investigation.

7. REFERENCES

[1] The Digital Object Identifier system. `http://doi.org`.

[2] Persistent URL homepage. OCLC Persistent Uniform Resource Locator service. `http://purl.org`.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic syntax. Internet Engineering Task Force, August 1998.

[4] F. Bry and M. Kraus. Advanced modeling and browsing of technical documents. In *Track on Electronic Books for Teaching and Learning, Proc. 17th ACM Symp. on Applied Computing*, 2001.

[5] F. Bry and M. Kraus. Perspectives for electronic books in the World Wide Web age. *The Electronic Library*, 20(4), 2002.

[6] B. G. Christensen, F. A. Hansen, and N. O. Bouvin. Xspect: Bridging open hypermedia and XLink. In *Proc. 12th Int. World Wide Web Conf.*, 2003.

[7] R. Daniel Jr. Harvesting RDF statements from XLinks. W3C note, World Wide Web Consortium (W3C), September 2000.

[8] D. C. De Roure, N. G. Walker, and L. A. Carr. Investigating link service infrastructures. In *Proc. 11th ACM Conf. on Hypertext and Hypermedia*, pages 67–76. ACM Press, 2000.

[9] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), June 2001.

[10] R. Fielding et al. RFC 2616: Hypertext Transfer Protocol (HTTP/1.1). Internet Engineering Task Force, June 1999.

[11] A. M. Fountain, W. Hall, I. Heath, and H. Davis. MICROCOSM: An open model for hypermedia with dynamic linking. In *European Conf. on Hypertext*, pages 298–311. Cambridge University Press, 1990.

[12] K. Grønbæk, L. Sloth, and N. O. Bouvin. Open hypermedia as user controlled meta data for the Web. *Computer Networks*, 33:553–566, 2000.

[13] P. Grosso, E. Maler, J. Marsh, and N. Walsh. XPointer framework. W3C recommendation, World Wide Web Consortium (W3C), 2003.

[14] F. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Com. ACM*, 31(7):836–852, 1988.

[15] F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Com. ACM*, 37(2):30–39, 1994.

[16] D. E. Millard, L. Moreau, H. C. Davis, and S. Reich. FOHM: A fundamental open hypertext model for investigating interoperability between hypertext domains. In *Proc. 11th ACM Conf. on Hypertext and Hypermedia*. ACM Press, 2000.

[17] A. Pearl. Sun's Link Service: A protocol for open linking. In *Proc. 2nd annual ACM Conf. on Hypertext*, pages 137–146. ACM Press, 1989.

[18] D. Raggett et al. Html 4.01 specification. W3C recommendation, World Wide Web Consortium (W3C), 1999.