INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstraße 67 D–80538 München

Ludwig
Maximilians-
Universität
München

Processing Hypertext Links after XLink

Michael Eckert

Projektarbeit im Rahmen des Fortgeschrittenenpraktikums

Beginn der Arbeit:	01.03.2004
Abgabe der Arbeit:	16.06.2004
Betreuer:	Prof. Dr. François Bry

Zusammenfassung

Die XML Linking Language (XLink) des W3Cs bietet Möglichkeiten, Hypertext zu verlinken, welche die der Hypertext Markup Language (HTML) bei weitem übertreffen. XLinks größter Fortschritt sind *out-of-line* Links. Solche Links sind nicht an ihrer Quelle oder ihrem Ziel definiert, sondern in sogenannten Linkbases, separat von den Dokumenten, die sie verknüpfen.

Jedoch lassen XLink und verwandte Standards viele Fragen offen. Einige, wichtige dieser Fragen werden in dieser Arbeit besprochen. Dann werden zwei davon näher beleuchtet und Lösungen in Form von Erweiterungen von XLink vorgeschlagen.

Der erste Erweiterungsvorschlag führt Interfaces als Linkteilnehmer ein. Die Interfaces stellen wohl-definierte Zugangspunkte zu Linkstrukturen zur Verfügung und erleichtern Wiederverwendung und Kombination von Linkstrukturen. Sie bieten eine symbolische Adresse für Dokumente an und können auch dazu benutzt werden, Dokumente von ihrer physikalischen Adresse zu entkoppeln.

Der zweite Erweiterungsvorschlag ist ein Linkbase Management-Mechanismus. Linkbase Management befaßt sich mit der Frage, wie externe Linkbases gefunden werden und welche Links wann angezeigt werden. Der vergeschlagene Mechanismus basiert auf drei Linkbase-Arcs. Diese Arcs erlauben es, eine Linkbase nur für das aktuelle Dokument, für ein bestimmten Bereich von Dokumenten oder für alle Dokumente zu laden.

Abstract

The W3C's XML Linking Language (XLink) offers hypertext linking capabilities exceeding those known from the Hypertext Markup Language (HTML) by far. XLink's biggest contribution is to allow *out-of-line* links. These are not defined at a link's source or destination, but in a linkbase separate from the documents they link.

However, XLink and related standards leave many questions open. After reviewing some of these questions, two are looked at closer and extensions to XLink are proposed as solutions.

The first contribution is the introduction of interfaces as link participants. These interfaces give well-defined access points to link structures and ease reuse and combination of link structures. They provide a kind of symbolic address and can also be used to decouple documents from their physical location.

The second contribution is the suggestion of linkbase management mechanism. Linkbase management is concerned with finding external linkbases and deciding which links are to be displayed. As an extension to XLink, the linkbase management mechanism introduces three linkbase arcs. The arcs allow a linkbase to be applied to the current document only, to a limited range of documents, or to all documents.

Contents

List of Figures

— use that definition to supress them

1 Introduction

Hyperlinks are an important, if not the most important, backbone of the World Wide Web. Hyperlinks guide human users through the vast amount of information offered on the Web. Users can jump from one information to another information anywhere on the web by following a hyperlink, they “surf” the Web.

Links are not only used by humans. In the present, search engines follow links to discover and rank information on the Web. In the prospect of a Semantic Web, a Web providing information that can be understood and processed by machines, links will be an important way to find information.

The links provided in the Hypertext Markup Language (HTML) of today’s Web are very simple. To improve upon this, other linking mechanisms have been suggested, such as the XML Linking Language (XLink).

1.1 Goals of this Work

The XLink standard leaves many questions open and its ideas give rise to new, interesting research issues. After outlining some of these issues, this work tries to contribute to two of them.

The first contribution is concerned with the structuring of links and how link structures can be provided with interfaces to ease combination of link structures.

The second contribution is concerned with links placed in external linkbases, i.e. outside the resources they link. It is discussed, how such linkbases can be found and when which links should be valid. The mechanisms determining this are called Linkbase Management here.

1.2 Organization of this Work

The remainder of this work is organized as follows: Section 2 will introduce the conceptual ideas behind linking on the Web with HTML and XLink. Important characteristics of the Web that have influence on linking, and differences between the Web and other hypertext systems are discussed.

A brief introduction to XLink and standards it relies on is given in section 3. Issues left open by XLink and related standards will be surveyed in section 4.

Interfaces for link structures are elaborated on in section 5 and possible solutions are proposed. Linkbase Management is addressed in section 6 and appropriate extension to XLink are suggested.

Finally, a discussion and an outline of future research directions in section 7 conclude this work.

2 Linking on the Web

The Web's architecture has important implications on linking. This section introduces the concepts of linking resources on the web using HTML, the currently most common linking mechanism. HTML is then contrasted with the new linking mechanisms devised in XLink. Finally, the fundamental properties of the Web, its implications on linking, and its differences to other hypertext systems are considered.

2.1 Links of the Present: HTML

Currently, most information on the Web is provided in the form of pages written in the Hypertext Markup Language (HTML, [RHJ99] and [P$^+$02]), or linked from HTML pages.

The basic HTML link is a link to another page or within the current page that can be followed —traversed— upon user request:

```
<a href="www.w3.org/TR/Xlink/#intro">The XLink Standard</a>
```

When the link source, that is the text inside the `<a>`-tag, is activated (e.g. by a mouse click), the link destination is opened. Usually, it is displayed in the current window replacing the old document. The `target`-attribute allows to change this behavior, for example to open the new document in a new window.

Links are also used to embed objects, such as images, executable programs, or other HTML documents, into a page:

```
<img src="icon.png" />
<iframe src="embedded_page.html" />
```

Finally, Links can be used to express a logical relationship between a page and another object that possibly has no hypertext behavior at all. A page can e.g. link to its stylesheet, an alternative language version of itself, or the next and the previous page in an ordered set of pages:

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

In its current versions (HTML 4.01 and XHTML 1.0), HTML supports 15 types of such logical relationships. However, the only link actually interpreted by most current user agents such as HTML Browsers is the link to a page's stylesheet.

All different links in HTML share the following characteristics, they are:

- *inline*, i.e. defined directly in the page of the link's source at this link's source

- *uni-directional* and *outbound*, i.e. they point only in one direction, away from their definition at the source
- *single-headed*, i.e. they have exactly one destination
- *restricted in their types*, as HTML supports only a fixed set of 15 types, and supports *no association with metadata*
- *restricted to predefined anchor points*, i.e. they can only link to a destination location inside a document if this document provides an anchor point at this location. The anchor point has to be specified by an `id`-attribute on any element. [1]

Compared to other hypertext and hypermedia systems like Intermedia ([YHMD88]), Sun's Link Service ([Pea89]), Mircocosm, [FHHD90]), HyperDisco ([WL96]), or the Dexter Hypertext Reference Model ([HS94]) and the HyTime standard ([HY97]) the linking capabilities of HTML are very limited and simple. However, the simplicity is also a strength: HTML's links are easy to understand and write for a page's author and require only minimal processing effort for user agent software.

2.2 Links after XLink

Within its XML Activity to provide a flexible text format to meet the challenges of electronic publishing and exchange of data on the Web, the World Wide Web Consortium (W3C) has developed the XML Linking Language (XLink, [DMO01]). XLink seeks to improve on HTML's linking capabilities in several ways:

- Links can be *out-of-line*, i.e. they can be placed in external documents, so-called linkbases, not containing a link's source.
- Links are not restricted to the outbound direction, but can be *multi-directional* allowing *outbound*, *inbound*, and *third-party* links (the latter two are *out-of-line* links).
- *Multi-headed* links are possible, i.e. a source can be associated with more than one destinations.
- Links can be *typed* and *associated with metadata* 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.
- In connection with the XML Pointer Language (XPointer), a link can *address arbitrary locations* inside an XML document. XPointer allows very fine-grained addressing, even below the level of nodes of a document tree.

Out-of-line links allow a separation of content and link structure. The content is provided in an XML document while link structure is provided in a linkbase. This is analogous to the common separation of content and presentation between a document and a style sheet.

An introduction to XLink and related standards will be provided in the next section.

[1]In versions HTML 4.01 and earlier this was usually done using an `<a>`-element with `name`-attribute. In XHTML 1.0 this is deprecated an replaced by the `id`-attribute that can be used on any element.

2.3 Important Properties of the Web

XLink's improvements on HTML linking are not new ideas. They have been discussed and implemented in the hypertext system community for quite some time.

The separation of data and links is especially popular in the Open Hypermedia Systems (OHS) area. Many Systems, such as Sun's Link Service ([Pea89]), or Microcosm ([FHHD90]) manage links separated from their source and destination. This idea is also inherent to the Dexter Hypertext Reference Model ([HS94]). A nice overview of these systems can be found in chapter 3 of [CH02]. The HyTime standard ([HY97]) also allows the separation of links from their source and destination.

There are, however, differences between the World Wide Web and these Open Hypermedia Systems. The following will outline the Web's properties important in the context of this work. These properties affect assumptions commonly used in other hypertext and hypermedia systems.

A good source of the Web's architectural properties are Tim Berners-Lee's notes on the Web's Design Issues ([BL]).

Open World

The World Wide Web is an open world. As long as one has access to a portion of namespace, one can add resource at will. No registration or announcement of the resources is needed.

Many hypertext systems implement a function to query a resource for all links having it as a destination. In the World Wide Web such a query is impossible as there is no entity having knowledge of *all* resources or links.

Note that while the term Open Hypermedia Systems contains the word "open", this refers to an openness to applications, that is documents from arbitrary applications can be interlinked. However, the systems usually do have a registry managing all links and sometimes also all documents. Thus they do not make an open world assumption allowing to add resources without registration.

Weak Identifiers

Resources on the Web are identified and accessed using Uniform Resource Identifiers (URIs). One resource can be accessible at various URIs. There is, in general, no way to tell only from the URIs itself whether two URIs point to the same resource or not.

This has impact on linking mechanisms allowing to identify a link's source with a URI. Consider a user agent having knowledge of a link with the source `http://www.example.com/index.html` and a user requesting to view `http://www.example.com/`. Both URIs identify the same resource. Displaying the link requires intelligence on the side of the user agent to establish this equality and might not be possible at all in some cases.

Subresources, that is parts of a whole resource such as a fragment of an XML document, are identified by appending a fragment identifier to their enclosing resource's URI. If a subresource moves inside a resource, a fragment identifier formerly identifying it, might now point to a different subresource or none at all. Syntactically different fragment identifiers might identify the same subresource or different subresources. In general, the only way to find out, is to actually evaluate the fragment identifiers on the resource.

Traditional hypertext systems usually do not face these problems with identifiers. They assume *unique* identifiers for every resource or sub-resource participating in a link. This assumption is also made in the Dexter Hypertext Reference Model ([HS94]).

Dynamic Resources

Many resources on the Web are dynamic, i.e. they can change their content over time. If the same URI is retrieved at two different times, there is no guarantee that the same content is returned. This also implies that a fragment identifier that worked once is not guaranteed to work the next time it is used.

To cope with changes in the content of resources, many hypermedia require changes to maintain unique identifiers assigned to subresources. The Web's fragment identifier mechanisms, such as XPointer, are much more flexible. Being in essence a queries against the content, such a general assumption is impossible.

Volatile Resources

Resources can not only change their content over time. They can also disappear temporarily, move, or disappear completely. Any linking mechanism for the Web has to live with the possibility that one of its participating resources is not accessible; broken links have to be tolerated.

Many hypertext system implement consistency checks and delete broken links. In the Web this is not possible.

Decentral and Distributed World

Resources on the Web are distributed all over the world hosted on different servers. The Web works without any central entities, so there is no central registry having knowledge of all resources. Ideas of central link registries serving all links are thus infeasible and against the Web's architectural philosophy.

Even on a smaller, limited scale, such as a company's or university's web site, central entities are questionable. While technically feasible, they can conflict with organizational structures of the outside world. For example, a company might want to distribute responsibilities, e.g. the authoring of web pages, to its departments and allow them a degree of autonomy, e.g. having their own web server fitting their technical needs best.

Many hypertext and hypermedia systems, e.g. Sun's Link Service ([Pea89]) or Microcosm ([FHHD90]) assume a central repository storing all structural information. This is a key difference to the Web's decentral architecture.

No Write Access to Documents

From a user's perspective, the Web is a read-only system. As a general rule, readers can only view resources with their browser; only the author and owner of a resource is allowed to modify.

Any modifications a reader might want to apply to a resource, such as placing an annotation, or building his own reader's view of a web page ([BK01], [BK02a]), 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.

For a browser to display the modifications, this requires first to establish the connection between the original resource and the resource containing the modifications. Then informations from both resources need to be merged for viewing.

While the Web's properties just discussed pose difficulties on linking mechanisms, it is worth noting that the Web also derives its greatest strength for them: it is an open, flexible, heterogenous and highly fault-tolerant system providing access to an incredible amount of content.

3 The W3C XML Linking standards

This section gives a brief overview of the World Wide Web Consortium's standards for XML Linking. Along with the standards, a graphical visualization of link specifications will be introduced. This graphical visualization will be used later on in this work to avoid hard-to-read XML syntax.

3.1 Overview

In linking Web resources, several standards are used cooperatively. The XML Linking Language (XLink) provides means to express links between web resources. The resources can be of various kind, most prominently XML documents or fragments thereof, but also other media such as pictures, audio, or video.

To link resources, these resources need to be identified. The standard way to identify resources on the web is to provide a resource's Uniform Resource Identifier (URI), or an Internationalized Resource Identifier (IRI). Sometimes a URI reference is relative, that is, it is identifies a resource relative to another resource's URI, a so-called base URI. XML documents can establish base URIs using the XML Base standard.

A resource identified by a URI can be composed of subresources. For example, an XML document can be viewed as a tree of nodes or a stream of characters. The XML Path Language (XPath) and the XML Pointer Language (XPointer) can be used address parts of a whole XML document.

3.2 Identifying Resources: URIs and XML Base

Uniform Resource Identifiers (URIs, [BLFM98]) are simply strings identifying resources on the web. As of writing this, URIs are being replaced by Internationalized Resource Identifiers (IRIs, [DS04]) which are a generalized form allowing non-ASCII characters.

The first part of a URI indicates the URI scheme used to interpret the schema specific part following. This is an example of a URI in the `http:`-scheme:

$$\overbrace{\texttt{http:}}^{\text{scheme}}//\overbrace{\texttt{www.w3.org}}^{\text{authority}}\overbrace{\texttt{/TR/xlink/}}^{\text{path}}$$

This example belongs to the most common class of URI, the Uniform Resource Locators (URLs). A URL identifies a resource by providing a representation of its primary access mechanism ([UPIG01]). The scheme indicates the protocol to be used for retrieval of a resource, while the rest of the URL (authority and path) determines a resource's location on the Internet.

The above URI is an example of an absolute URI, it refers to a resource independent of the context it is used in. Often URIs also occur in relative forms only describing the difference to an absolute base URI. This is especially useful when a set of XML- or HTML-documents are designed together in a filesystem-like hierarchy and refer to each other by means of URIs. Relative URIs allow for the tree of documents to be moved without changing the URI references in the documents.

The base URI needed to resolve all relative URI references contained in a resource can be provided by the resource itself in its content. XML documents can do so by using the `xml:base`-attribute ([Mar01]), HTML documents by using the <BASE>-Tag ([RHJ99]). If a resource does not supply a base URI, the URI that was used to retrieve it becomes the base URI.

3.3 Addressing Fragments: XPath and XPointer

URIs refer to resources as a whole. Often it is desirable to address subresources within a resource, e.g. a fragment of document constituting a chapter or section.

HTML allows a document's author to name nodes of the document tree using the `id`-attribute. These can be referred to by the combination of a URI and a fragment identifier, as in this example:

$$\overbrace{\texttt{http://www.w3.org/TR/xlink/}}^{\text{URI}}\#\overbrace{\texttt{extended-link}}^{\text{fragment identifier}}$$

HTML's mechanism has some shortcomings. A document's author has to provide the `id`-attributes, it is not possible to identify fragments of a document not carrying `id`-attributes. Also only whole nodes in a document tree can be addressed, the sub-node level is inaccessible.

The W3C's XML Pointer Language (XPointer) efforts seek to overcome these shortcomings. XPointer allows various so-called schemes for addressing into XML documents ([GMMW03b]), most prominently the `xpointer()`-scheme ([DMD02]).

The `xpointer()`-scheme addresses the internal structure of XML documents using path expressions similar to those of the XML Path Language (XPath, [CD99]). Most importantly XPointer extends XPath with the ability to address points and ranges at sub-node-level, e.g. single words within a text node, and to use string matching functions.

Here are some examples of valid XPointer-expressions:

$$\overbrace{\texttt{book.xml}}^{\text{relative URI}}\#\overbrace{\texttt{xpointer(}}^{\text{scheme}}\overbrace{\texttt{/book/chapter[@title='xml']/para[1]}}^{\text{path expressions locating a node}}\texttt{)}$$

$$\texttt{book.xml\#xpointer(}\overbrace{\texttt{/book/chapter[3] to following:chapter[5]}}^{\text{path expression locating a range}}\texttt{)}$$

$$\texttt{book.xml\#xpointer(}\overbrace{\texttt{string-range (//chapter[last()], "XPointer")}}^{\text{match for string ``XPointer'' in last \texttt{chapter} element}}\texttt{)}$$

A full description of XPointer goes beyond the scope of this overview. For the problems discussed in this work, it suffices to have an intuitive understanding that XPointer allows to locate arbitrary ranges in an XML document, similar to what a user could select using the mouse if the document was displayed in a browser.

Also XPointer raises a couple of interesting research questions such as the construction of "good" XPointer expressions that are robust to small changes in a document. These questions are outside this work's scope.

3.4 Linking resources: The XML Linking Language (XLink)

Knowing how to identify resources and subresources one can finally turn to linking resources. From now on, the term "resource" will be used to refer to both, whole resources (e.g. whole XML documents) and subresources (e.g. fragments of XML documents).

XLink is designed to be an "enabling vocabulary" ([Mal]) for XML; it is not meant to be used by itself but should be incorporated into one's own XML vocabularies. Arbitrary XML elements can become linking elements by carrying the `type`-attribute from the XLink-Namespace

```
http://www.w3.org/1999/xlink
```

This namespace is assumed to be bound to the prefix `xlink:` for the remainder of this text.

Simple Links

Two forms of links are present in XLink: *simple links* and *extended links*. Simple links closely resemble HTML's links; they are inline, outbound, and single-headed. Here is an example:

```
<element xlink:type="simple"
         xlink:href="http://www.w3.org/TR/XLink/#intro">
  Some Content
</element>
```

Simple links are only a shorthand notation and can equally be expressed by means of extended links.

Note that the tag `<element>` can be chosen arbitrarily; for XLink only the attribute `xlink:type` matters to make the element a simple link.

Extended Links

Figure 1 gives an example of an extended link expressed in XLink's XML syntax, figure 2 depicts a graphical representation of that extended link.

Extended link elements are denoted by the attribute `xlink:type` with value `"extended"`. In the syntax of figure 1, the element-tag names `<xlink>`, `<res>`, `<loc>`, and `<arc>` are chosen according to the element's XLink-type. This is only for better readability and has no meaning to XLink. XLink-specific meaning is only derived from the `xlink:type`-attribute.

An extended link is only a mere container. In the visualization of links used here, it is illustrated as a rectangle with rounded corners.

It contains an arbitrary number of elements describing the link's participants. A participant can either be a local resource or a remote resource. Participants carry a label (in the `xlink:label`-attribute) used to identify them within the extended link.

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

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

    <loc xlink:type="locator" xlink:label="Loc1"
         xlink:href="http://example.com" />
    <loc xlink:type="locator" xlink:label="Loc1"
         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: An extended link in XLink's XML syntax

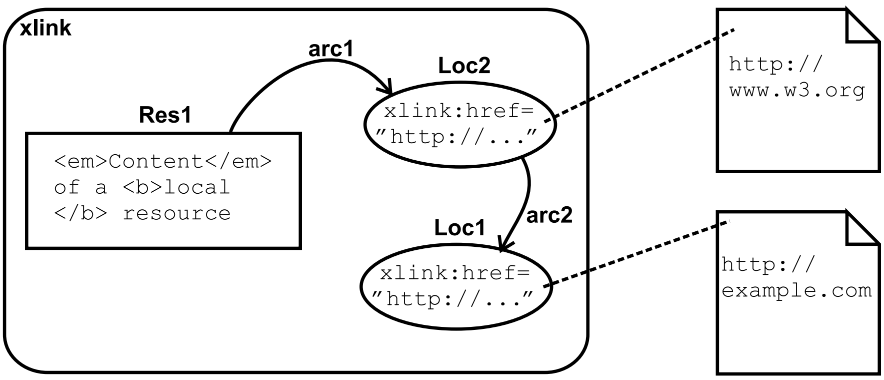


Figure 2: Visualization of an extended link

Local Participants: Resources

Local resources are XML data actually contained in the link itself. The XML data is surrounded by a linking element of the type resource (`xlink:type="resource"`). Rectangles containing the textual representation of the XML data are used to illustrate local resources.

Identifying Remote Participants: Locators

In contrast, remote resources can be any kind of resource that can be identified by means of a URI reference, e.g. XML documents, fragments of XML documents, images, or audio. This includes resources that sit in the same document as the extended link. It also allows to identify and link to other extended links, though this has no special meaning according to the XLink standard; they are treated as any other XML fragment.

In order to identify remote resources, an extended link contains locator elements (`xlink:type="locator"`) providing the remote resource's URI in the `xlink:href`-attribute. In the visualization, locators are displayed as circles or ellipses containing the URI. Sometimes a dashed line will be drawn from the ellipses to indicate the actual resource the URI points to.

An extended link can contain an arbitrary number and combination of locator elements and resource elements. This includes the extreme cases where the participants are only local resources or only remote resources. In fact, the latter case will likely be very common.

Specifying Traversal: Arcs

So far, an extended link has only collected its participants, but not information is provided how to navigate from one participant to another. This information is supplied by arc elements (`xlink:type="arc"`) contained in the extended link element.

An arc pairs up two participants in a directed relationship, i.e. one participant is starting point, the other ending point for traversal. Syntactically, an arc uses the `xlink:from`- and `xlink:to`-attributes for this. The attributes take a locator's or resource's label as value to indicate the corresponding starting or ending participant.

Arcs are called *outbound*, if the originate *from a local resource*, *inbound* if they point *to a local resource*, and *third-party* if the connect *two remote resources.*

Due to some syntactic shorthands XLink allows, it is impossible to have an extended link completely without arcs; the absence of arcs is interpreted as arcs from all participants to all participants.

Expressing Simple Links as Extended Links

It now can be seen that a simple link can easily be mapped to an extended link: the simple link's content becomes a local resource, the destination a locator, and one arc specifies an outbound traversal behavior. Figure 3 shows a visualization the simple link from earlier.

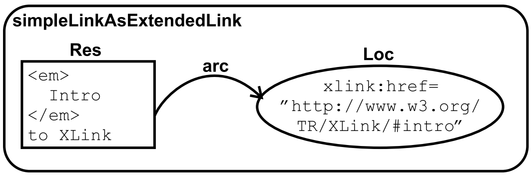


Figure 3: An extended link that is equivalent to a simple link

Behavior of Arc Traversal

Additionally to specifying the direction of traversal, arcs can hint the behavior of traversal. This, of course, makes only sense if XLink is used in a hypertext setting.

The attribute `xlink:show` indicates how the ending resource will be displayed. With the value `embed` the ending resource will be displayed embedded *within* the starting resource. This is comparable to the behavior of images linked from HTML documents. With the value `new` it will be opened in a new presentation context, e.g. a new browser window. The value `replace` opens the the ending resource in the current presentation context, replacing the the starting resource.

Note that displaying a subresource (e.g. XML fragment) usually means displaying the whole resource (e.g. the XML document) containing the subresource. Typically some focus is put on the subresource, e.g. by scrolling to the XML fragment or highlighting it). This is analogous to the behavior of HTML links.

The attribute `xlink:actuate` indicates when an arc will be traversed. With the value `onLoad`, an arc will be traversed as soon as the starting resource is loaded. With the value `onRequest`, an arc will be traversed on user request only, e.g. when the user clicks the starting resource.

Both attributes allow additionally the values `none` and `other` to indicate that either no behavior information has been given, or that an application should look for behavior information elsewhere, e.g. in application-specific attributes.

Associating metadata with Links

All linking elements discussed so far, the extended links, their participants, and their arcs, can be associated with metadata, both machine-readable and human-readable.

Human-readable metadata can be provided with the `xlink:title`-attribute and title elements (`xlink:type="title"`). Title elements were introduced as means for a linking element to have several titles, allowing easier internationalization.

Machine-readable metadata can be supplied using the `xlink:role` and `xlink:arcrole` attributes. Their values are arbitrary URIs possibly providing more information when resolved. The only arcrole predefined by XLink is:

```
http://www.w3.org/1999/xlink/properties/linkbase
```

It is used to locate linkbases and will be discussed in the context of Linkbase Management in section 6. Unfortunately XLink does not specify a default set of roles or arcroles for the use in hypertext analogous to HTML's values for the `rel`-attribute (cf. section 2.1).

3.5 Summary

XLink connects resources using the concept of a simple or an extended link. An extended link can, in essence, be seen as a directed graph: the link participants are the nodes, and the arcs the directed edges.

XLink's terminology of links and arcs tends to be a little confusing. A (hyper)link is usually understood as a connection between resources that can be traversed. In XLink this is an arc; XLink's extended links are only "containers" for arcs and cannot be traversed as such.

This work tries to explicitly speak of "XLink links", "extended links", and "arcs" when the precise distinction is important. The term "link" is used when speaking of hyperlinks in a more general fashion that is not XLink-specific.

Participants of an extended link can be local resources contained in the link itself or remote resources. Locators identify remote resources using URIs and fragment identifiers such as XPointer expressions.

XPointer provides a powerful language to address fragments of an XML document. XPointer allows very fine grained addressing, below the level of nodes in the XML document tree.

Arcs put two resources in a directed relationship signaling that a user can traverse from one to the other. Arcs can give behavior information as to *when* traversal is to be initiated and *how* the destination resource is to be presented.

While XLink is primarily targeted at hypertext applications ([BL02]), it also can be and has been used to link arbitrary data not necessarily intended to be navigated like hypertext (e.g. [MM03]).

4 Issues Left Open in XLink and Related Standards

While the XLink standard defines a clear syntax and gives powerful mechanisms for linking, it also leaves many questions open. This section illustrates some of the open issues and how they can give rise to new, interesting research. Many of the issues are not shortcomings of the XLink standard itself, but result more from lack of integration into other standards.

4.1 Presentation of Links

The `<a>`-element and the other link elements of HTML allowed no overlapping or nesting of link sources and only a single destination per link source. Although HTML does not prescribe how a link source is to be presented, rendering it in a way obvious to the user, e.g. blue and underlined, has been established as a best practice.

With XLink, XML fragments that are link sources can overlap or nest. Currently there are no standards or best practices hinting how overlapping or nested link sources are to be presented visually in a manner illustrating the overlapping or nesting.

When a link with multiple destinations is activated, one possibility is to allow the user to choose one of the destinations. This solution is preferred by many hypertext systems and also suggested as a possibility in the XLink standard. However, other possibilities, such as presenting all destinations in a split browser window, are imaginable.

A nice overview of different approaches to visualization of links can be found in [WOL01].

If links are used to embed an object into a document, they affect the layout of the presentation of this document. In HTML, link elements embedding other objects can give hints on the layout using attributes. For example, the `<img>`-element can specify the image's alignment with respect to the surrounding text using the `align` attribute. For links specified with XLink no such facilities exist.

These presentational issues are not a shortcomings of the XLink standard itself. They rather stem from a lack of integration with other standards such as XHTML and Cascading Style Sheets (CSS, [BÇHL04]). An integration with these standards, however, is crucial to a success of XLink's improved linking mechanisms.

4.2 Hyperlinks and Styling

HTML mixes content with information for its presentation, such as layout and style, in one document. Today's XML world tries to separate these concerns: Content is specified in XML using a suitable vocabulary of elements and attributes. To present the content, it it is transformed —"styled"— for presentation using a style sheet. The two most common style sheet languages are Cascading Style Sheets (CSS, [BÇHL04]) and Extensible Stylesheet Language Transformations (XSLT, [ABC+99]).

Using CSS, XML documents can be attributed with presentation information while leaving the document structure unchanged.

Alternatively, an XML source document can be transformed into into a presentation document containing presentation information. Typically such transformations are made with XSLT having XHTML or XSL-Formatting Objects (XSL-FO, [ABC+01]) as target language.

In contrast to CSS, XSLT allows drastic transformations of the source document's structure, such as duplicating, adding, deleting or reordering elements.

If an XML document is to be linked with XLink, it is desirable to specify the links on the presentation-independent source level. If links are placed on the source level, a transformation of the document for presentation should also transform the links. Preferably this should be done automatically with no effort for the stylesheet author, but allow some intervention such as suppressing certain links.

Addressing Parts of a Document

Both style sheet languages, CSS and XSLT, access and refer parts of a document, using CSS-selectors and XPath, respectively. In doing so, their smallest unit of addressation is a *node* in the document tree.

Using XPointer, XLink allows finer grained addressation. XPointer can select points and ranges ([KRC+00]) in an XML document that do not coincide with nodes or node sets. Ranges can even produce document fragments that are not well-formed in the XML sense. The example from [Wal01] in figure 4 illustrates this.

```
<doc>
<p>This text demonstrates
<emph>a link</emph> that spans a <emph>not well-formed</emph>
range.</p>
</doc>
```

Figure 4: The underlined XML fragment is a range that is not well-formed

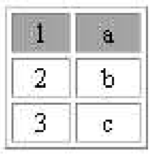
Coping with an addressation having no direct mapping into nodes or sets of nodes can present an immense challenge to any style sheet language.

Forward Transformations

In order to transform links while transforming a source document to its presentation, a style sheet processor must be able to establish a correspondence between source level and presentation level elements. In other words, for every endpoint of link on the source level, the processor must find a corresponding endpoint on the presentation level.

For a style sheet language like CSS that does not change the structure of the document this does not pose a big problem. However, in a style sheet language allowing transformation of the document structure, such as XSLT, defining and establishing such a correspondence can be hard.

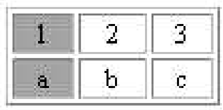
It is not necessarily a one-to-one correspondence: an endpoint on the source level might not be present on the presentation level, or it might be duplicated on the presentation level. One can also imagine cases such as the inversion of a table, where it is not clear what correspondence between endpoints means at all. Consider the example of a link on a row in a table as in figure 5. When the table is inverted, the row becomes a column and thus the link endpoint must be split to cells.



(a) table containing a link on a row (highlighted region)

```
<table>
  <tr> <td>1</td> <td>a</td> </tr>
  <tr> <td>2</td> <td>b</td> </tr>
  <tr> <td>3</td> <td>c</td> </tr>
</table>
```

(b) the table's HTML code; linked region (underlined) on row-element



(c) inverted table with link now on a column

```
<table>
  <tr>
    <td>1</td> <td>2</td> <td>3</td>
  </tr> <tr>
    <td>a</td> <td>b</td> <td>c</td>
  </tr>
</table>
```

(d) the inverted table's HTML code; linked region is split to cells

Figure 5: Inversion of a table containing a link on a row

Backward Transformations

So far, it was discussed how to do a forward transformation of links from the source to the presentation level. Symmetrical is the problem of backward transformation, that is transforming link endpoints from the presentation level back to the source level.

A backward transformation is important if a user is allowed to interact with the presentation level of a document, e.g. to place annotations. The user selects parts of a document on the presentation level as one endpoint of the annotation link. In order for this link to be visible also in other presentations, it is necessary to store the endpoints on the source level. The endpoint on the presentation level needs to be mapped to an endpoint on the source level.

This issue is similar to the problem of view updates in relational database theory: from some tables in a database (source) a view (presentation) is generated by means of a query (transformation). Now an update in the view has to be propagated back into the original tables. An attempt to the view update problem in hypertext is made in [FNP00].

Note that a backward transformation is simpler than the complete view update problem: it is only necessary to identify corresponding points, not to propagate changes in content.

A discussion of the problems with links on source and presentation level can be found in chapter 2.5 of [Kra00]. An attempt to suggest language extensions to XSLT for cooperation with XLink is made in [Wal01].

4.3 Querying Linked XML

XLink can be used to link arbitrary XML data, not necessarily intended for hypertext representation. Consider an XML file containing information about students while another XML file contains information about classes. Links can be used to link students with the classes they are registered for. These links can be stored in either of the two files, or in a third file.

Used in such a manner, XLink can be understood as a generalization of the ID/IDREF intra-document references. It is a common assumption and simplification not to consider points or ranges and restrict link endpoints to nodes (e.g. by allowing only XPath to locate XML fragments).

Running queries against linked data is different from styling documents with links: one does not want to transform the links from a source to a presentation level, but *access* the information at a link's destination in the query. In other words one needs to dereference links in a query.

Manual Dereferencing

Although the W3C's XML Query Requirements ([CFMR03]) state that "3.4.12: Queries MUST be able to traverse intra- and inter-document references", the XML Query Language (XQuery, [BCF+03]) offers no support for this. It is up to the query author to identify link endpoints and dereference links manually.

Dereferencing ID/IDREF links manually is relatively easy. Presence of an IDREF attribute signals a link endpoint and the destination can be found using a value join. For example, the XPath expression

```
//*[@id = /myref/@idref]
```

locates the element with the ID `id` referenced by the element at `/myref` using the IDREF `idref`

Dereferencing XLink's extended links manually is much harder. In the case of out-of-line links, link information is not present at the endpoint itself, but at location where the link was specified. Also the link information has to be accessed in its syntax.

Using a Dereferencing Operator

To dereference ID/IDREF links, the XML query language QUILT ([CRF00]), introduced a special axis for its XPath-like node selection expressions. Using this axis, the value join example can be rewritten as

```
/myref/@idref->myid
```

It locates the element referenced by `idref` at `/myref`, adding a check that it has the element name `myid`.

Dereferencing XLink links could be done in a similar manner, providing a link dereferencing axis in XPath. For example

```
/myelement/xlink::*
```

could select all elements that are destinations of links with source at `/myelement`. The axis also should allow constraints on a link's metadata, e.g. to follow only links with a specific `xlink:role`-Attribute.

Transparent Dereferencing

A different approach to querying linked documents is to dereference links in a transparent manner. From the perspective of the query, if an element is linked to other elements, these would be accessible in the same way as the element's childs are. Such an approach to transparently dereference XLink's links is described in [MM03].

Note that transparent dereferencing of XLink's links can mean that a directed graph, not necessarily a tree, is queried. This can pose new challenges, e.g. if a graph contains cycles.

4.4 Support for Generic Links

Many hypertext systems support the concept of generic links (e.g. Microcosm, [FHHD90]), which are also sometimes called global links ([GSO99]).

A *generic link* specifies a single fixed destination, but a number of generic sources that are applied globally to any document. Consider a set of web pages that link to a person's homepage whenever the person's name appears somewhere in the text. Without generic links this requires authors to manually find every page and every appearance of a name and create the links. With generic links an author will only specify a link from any source matching the name as text string to the person's homepage.

Typically, the sources of generic links are specified by means of text string matching. Generic links can ease link authoring and have a number of applications such as cross references in dictionaries or automatic links to definitions of terms listed in a glossary.

XLink, together with XPointer, has limited support for generic links: XPointer provides a very flexible way to specify a link source. This includes the simple string matching commonly used, but also allows more complicated specifications, e.g. taking into account the document structure.

However, while a link author can use XPointer to generically identify resources *within* an XML document, he has still to enumerate the URIs of all XML documents this link applies to. There is no way to say that an XPointer expression is to be applied to all documents.

4.5 Links and Structuring

A hypertext link rarely comes alone. Rather, a number of links together impose some higher structure on the documents they connect.

An excellent example for for such a structure is a guided tour: It puts a number of independent documents into a sequence, hinting a reader an order in which they are best read or giving a serialization of the documents for printing.

The importance of providing readers with guided tours through a set of documents was already anticipated in Vannevar Bush's first vision of a hypertext system ([Bus45]). Guided tours exist in many present-day hypertext systems under different names such as path, tour, or trail. They do not have to be sequential, but can also contain branches where a reader must choose or where the system chooses automatically based on some conditions ([Zel89]).

Other examples of link structures include links into a glossary, cross-references, or a table of contents supplying many outgoing links.

XLink's basic structuring concept is an extended link that groups arcs, i.e. directed relationships of two resources, together. Section 5 will show how link structures can be formulated as extended links and how link structures can be provided with an interface mechanisms.

As will be shown, an interface mechanism is a simple and efficient way to build composite structures from link structures, such as combining two guided tours into on big guided tour. Interface mechanisms can also help maintainability by decoupling documents from their physical location.

4.6 Management of Linkbases

XLink's support for out-of-line links is often seen as the biggest improvement on HTML linking ([DeR99]). Links can be authored and stored in linkbases, separately from the documents and resources they connect.

The question that arises is *how* these linkbases can be found and under *when* (which circumstances) are their links to be displayed. Treatment of this question will be called linkbase management ([BK02b]). XLink answers linkbase management questions only insufficiently.

Section 6 will show the influence of linkbase management on documents being part of one or more link structures and propose extensions to XLink for linkbase management.

5 Interfaces for Link Structures

An interface is a "place at which independent and often unrelated systems meet and act on or communicate with each other" ([MW]). In building big systems, defining the interfaces of the subsystems is an extremely important task.

A link structure, such as a guided tour, can be viewed as a subsystem in a big hypertext system. Providing link structures with interfaces, similar to interfaces in programming languages, can help building composites of link structures greatly. Building composites from hypertext documents, links and other composites has been recognized as a very important issue for hypertext in Halasz' "Seven Issues for the Next Generation of Hypermedia Systems" ([Hal94]).

XLink has no notion of interfaces or other means to build composite structures. After motivating examples, possible solutions will be discussed from a conceptual point of view. Finally a concrete solution will be suggested as an extension to XLink.

5.1 Motivation

Two examples will illustrate how interfaces can be beneficial. The first example, the composition of two guided tours, will show how interfaces can be used to build composite structures. The second example will show how interfaces can help decoupling structure from physical locations of documents.

Composition of Guided Tours

A guided tour provides a reader with an order in which a set of documents are best read. In terms of linking this means linking each document to the next document in that order. Figure 6 shows how a guided tour can be expressed as an extended link in XLink. It is not obvious to formulate the guided tour as one big extended link, instead many small extended links would also make sense. However, one big extended links avoids duplication of the locators and demonstrates that the documents belong together.

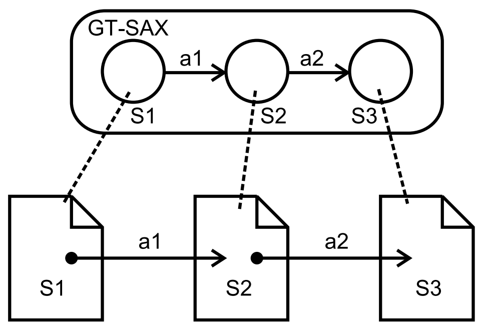


Figure 6: Formulating a guided tour as an extended link

To make this example more vivid, let's say that this guided tour, called GT-SAX, provides an introduction to Java's SAX-API. Let's also assume that another guided tour GT-DOM about Java's DOM-API is available.

Introductions to Java and XML (e.g. [Mac02]) often first explain SAX followed by an explanation of DOM. Being aware of the above guided tours, providing a big guided tour to SAX and DOM can be a simple matter of composing the two guided tours into a sequence.

Composing the two guided tours means that a user will be shown the first document of the DOM-tour after finishing the last document of the SAX-tour. In other words, a link between these two documents is needed as shown in figure 7.

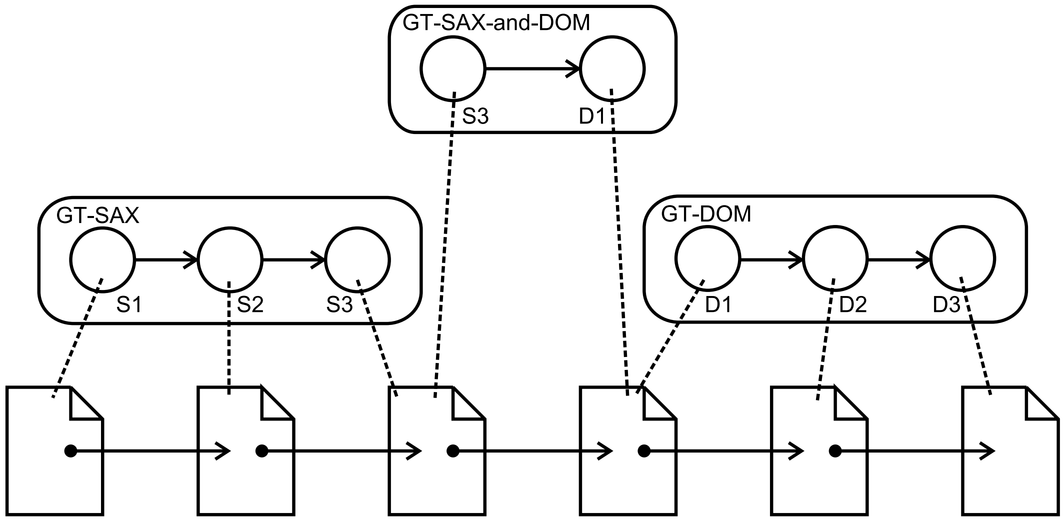


Figure 7: Composition of guided tours without interfaces

To create such a composition link, the URI of the last document S3 of GT-SAX and the first document D1 of GT-DOM must be known. Changes to a guided tour, e.g. adding another document S4 at the end of GT-SAX, break the combined tour (see figure 8). It would be much cleaner and less vulnerable to changes to be able to refer to the "end of GT-SAX" instead of S4's URI. In other words, well-defined access points, or interfaces, to the links are desirable.

Course Catalog

Consider the example of a course catalog shown in Figure 9. A page hosted and maintained by a university's computer science department shows all courses offered this semester and also provides links to the courses' homepages. The course homepages however are hosted and maintained by the research and teaching units offering the course. These units might have very different structures on their respective web sites.

HTML linking requires the author of the course catalog to find out the URIs of all course homepages and link to them in the catalog. If a teaching unit changes the structure of their web site, the catalog author needs to be informed and adjust the catalog to avoid broken links.

Using XLink can help a little: each teaching unit can offer a linkbase and the course catalog simply uses the linkbases. Now however, the linkbases must be aware of the location of the catalog and the places where to anchor the links. Especially any restructuring of the catalog,

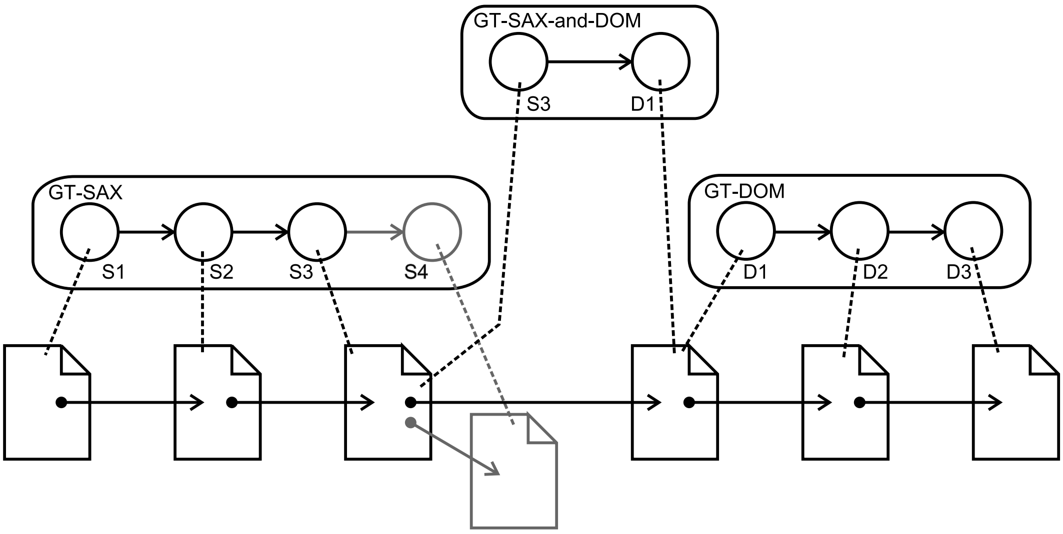


Figure 8: Broken composed guided tour after adding the resource S4 in GT-SAX without readjusting GT-SAX-and-DOM

like splitting it into multiple pages or renaming it from `"current.xml"` to `"summer2004.xml"`, will break the linkbases of all teaching units.

Having well defined interfaces can distribute the responsibilities and ease authoring. Interfaces can provide a symbolic name to a course homepage (as opposed to "physical address" in form of a URI) The authors only have to agree upon the names of the interfaces and the location of the linkbases. Such a symbolic name provides an indirection mechanism that makes the links less vulnerable to changes in physical structure.

5.2 Possible Solutions

Interfaces are access points to the structure an extended link provides; they are thus participants of extended links. Two approaches to providing an interface mechanism for XLink are now discussed: designating *existing* link participants as a link's interfaces or introducing *new* special participants as interfaces.

Naming Participants

To designate an existing link participant as a link's interface, this participant needs to be given a name. The name is used to refer to it when accessing the interface. For example the guided tours GT-SAX and GT-DOM from figure 7 would name their first and last pages as GT-SAX-start, GT-SAX-end, GT-DOM-start, and GT-DOM-end, respectively. The combination GT-SAX-and-DOM would then refer to the resources by their names, not their URIs. This implies an indirection: when accessing GT-SAX-start, the name has to be looked up in GT-SAX and resolved to a URI.

Such a naming mechanism has a parallel to redirects in the Hypertext Transfer Protocol (HTTP, [FGM+99]): a request for a resource is not answered with the resource, but an

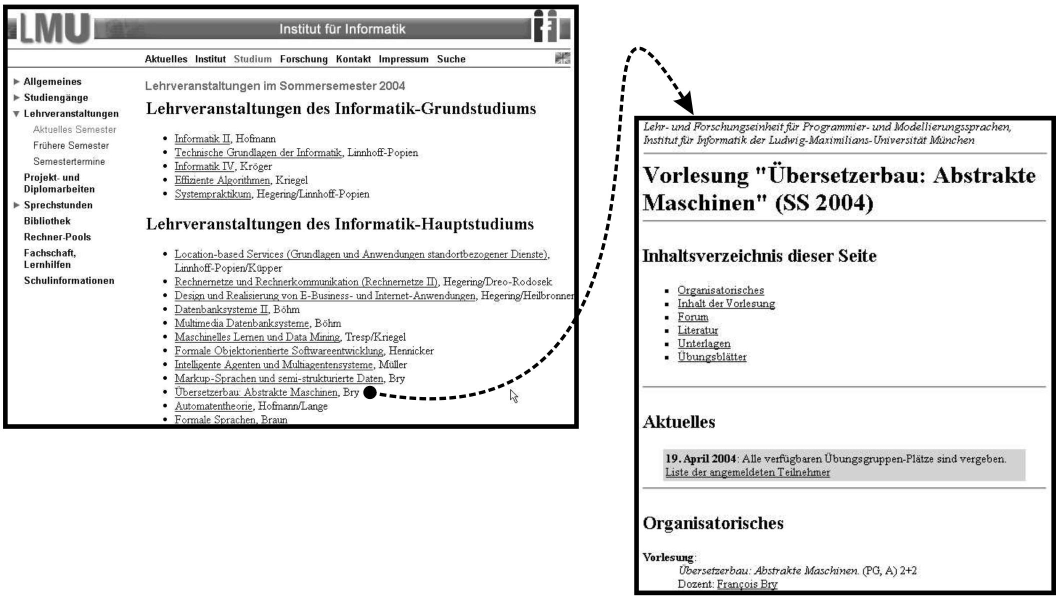


Figure 9: A course catalog hosted by the computer science department links to a course's page hosted by one of the department's research and teaching unit

indication of a different location (a URI) where the resource is to be found. It also is similar to the use of Uniform Resource Names (URNs, [Moa97]) that only supply a name, no physical location.

Introducing Interfaces as Participants

Simply naming link participants does not make the interfaces of an extended link very explicit. Also a resource will exist under various names and URIs, adding burden for a link processor.

An alternative is to introduce special link participants that only serve the purpose of providing an interface to access a link from the outside. Such interfaces are nodes in the hypertext structure just like normal resources; they are however transparent to the user and will not be displayed.

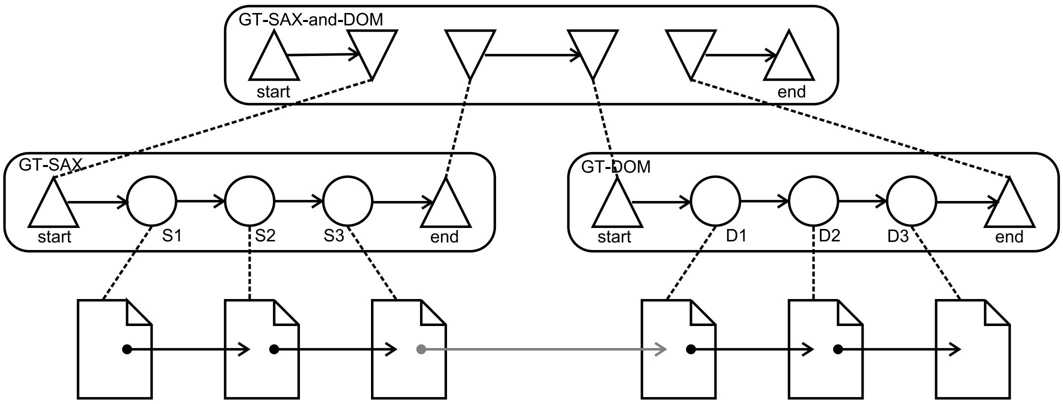
(a) Extended link from two resources to two resources

(b) Equivalent link with interface participant (depicted by a tip-up triangle)

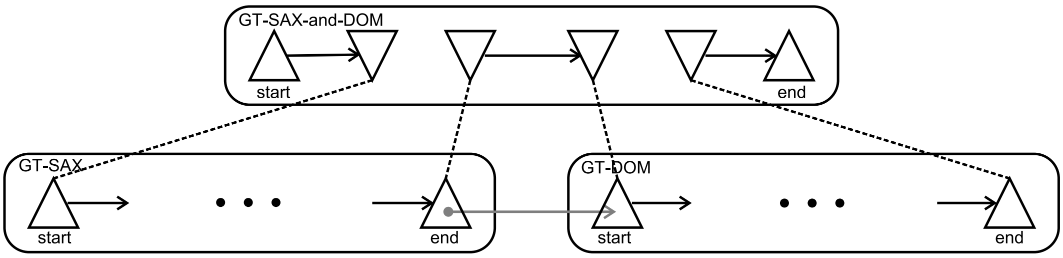
Figure 10: Using interface participants inside links

Such transparent nodes can also be useful inside links. Introducing a transparent node in the link from two resources to two other resources in figure 10(a) yields the equivalent, but maybe more concise, link of figure 10(b). The conciseness becomes more obvious when considering higher number of resources.

Figure 11(a) shows the composition of guided tours using special interface participants, depicted with triangles. Tip-up triangles indicate interface definitions, tip-down triangles indicate interface references. Figure 11(b) shows the situation from the perspective of the composition link: it only needs to know the interfaces of the extended links it composes.



(a) Composition, all participants shown



(b) Perspective of the composing link GT-SAX-and-DOM

Figure 11: Composition of guided tours with interfaces

5.3 Suggested Extensions of XLink

This work settles for the introduction of special interface participants. They seem preferable as they make a link structure's interfaces more explicit, can also be useful inside extended links as transparent nodes, and do not require handling of multiple names and URIs for the same resource in the link processor.

An interface is a link participant —just like other resources— that serves as an access point to the structure provided by the extended link. Other links can access and reuse the link's structure by linking to or from the interface with arcs. Unlike other resources, interfaces have only structuring purpose and are not displayed to the user; they are "transparent".

Two new link participants are suggested as an extension to XLink: interface definitions and interface references. This work uses a new namespace

```
http://www.pms.ifi.lmu.de/xlinkext/
```

for the extensions. It is assumed to be bound to the prefix `xlinkext:`.

Interface Definitions

Interface Definitions are link participants just like locators and resources. Their `xlink:type`-attribute is set to the value `other`, and the `xlinkext:type`-attribute signals their type with value `interfacedef`.

In order to be referred to, an interface definition needs a name. This name has to be a unique identifier within the document containing the interface definition. It is supplied in the `xml:id`-attribute of type ID ([MV04]). Any other attribute of type ID would also work. Note that the `xlink:label`-Attribute is not a good candidate since it is not an ID. Within an extended link a label may be used by more than one participants and within a document several links can use the same label.

Putting all together, an interface definition looks like this:

```
<xlink xlink:type="extended">

  <interfacedef xlink:type="other" xlinkext:type="interfacedef"
                xml:id="sax-start"
                xlink:label="mystart" />

  <!-- ... -->

</xlink>
```

Interface references

Within an extended link, arcs can simply identify an interface by its label, just like any other participant.

To refer to an interface from the outside, interface references are introduced. They are link participants having `xlink:type="other"` and `xlinkext:type="interfaceref"`.

Two things are necessary to identify the interface they refer to: the name given to the interface in its definition and a URI of the linkbase containing the definition. Both can be put together into one URI reference with the fragment identifier being the interface's name. The `xlink:href`-attribute takes this URI reference as value.

```
<xlink xlink:type="extended">

  <interfaceref xlink:type="other" xlinkext:type="interfaceref"
```

```
                xlink:href="http://www.example.com/linkbase.xml#sax-start"
                xlink:label="sax-start" />

  <!-- ... -->

</xlink>
```

Example: Composed Guided Tour's XML

Figure 12 shows the XML for the composition of guided tours using the interface definitions and references just described. A graphical representation was already shown in figure 11.

Processing Interfaces

A browser should treat interface resources the same as *documents* with the exception that they will not be displayed to the user. When a user traverses an arc inbound to an interface resource, instead of displaying it, an outbound arc will be traversed *automatically*.

If the outbound arc leads again to another interface resource automatic traversal continues. Of course, a browser should take care not run into cycles and might want to pose a limit on the maximum of automatic traversals. Note that this is not a new problem introduced by the interfaces: a set of documents with a cycle of arcs with `xlink:actuate="onLoad"` needs the same considerations.

In the overall traversal, several arcs can carry conflicting behavior information of the type `xlink:show`. In this case it seems most sensible to apply the behavior given by the first arc, that is the arc used to initiate the overall traversal.

If an interface has multiple outbound arcs, these are treated just like multiple outbound arcs for any other resource, e.g. by allowing the user to choose one arc for traversal.

Local Resources are not an Alternative

Interface definitions are very similar to local resource elements, interface references very similar to locator elements. In fact, it might seem that interfaces are redundant and a combination of local resources and locators with arcs having `xlink:actuate="onLoad"` and `xlink:show="replace"` as in figure 13 yields the same result.

This is not the case, the result is fundamentally different. When the inbound arc to the local resource is traversed, the *whole document* containing the local resource is loaded and processed. Thereby not only the arcs originating from the one local resource used as an interface can be traversed, but *all* `xlink:actuate="onLoad"`-arcs originating from *any fragment of the whole document* can be traversed.

It is not guaranteed that the one arc originating from the local resource used as interface is traversed at all: the document can contain several arcs with `xlink:actuate="onLoad"` and `xlink:show="replace"` as is the case in figure 13.

The `xlink:show="replace"` specifies to replace the current document with the destination document in the given presentation context (browser window etc.). No new presentation

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

  <locator xlink:href="http://www.example.com/sax1.xml" xlink:label="sax1" />
  <!-- ... -->
  <locator xlink:href="http://www.example.com/saxn.xml" xlink:label="saxn" />

  <interfacedef xlink:type="other" xlinkext:type="interfacedef"
                xml:id="gt-end"    xlink:label="end" />

  <arc xlink:type="arc" xlink:from="start" xlink:to="sax1" />
  <arc xlink:type="arc" xlink:from="sax1" xlink:to="sax2"
       xlink:actuate="onRequest" xlink:show="replace" />
  <!-- ... -->
  <arc xlink:type="arc" xlink:from="saxn" xlink:to="end" />
</xlink>
```

(a) Guided Tour about SAX, located at `http://www.example.com/saxtour.xml`

```
<xlink xlink:type="extended">
  <!-- interfaces of the combined gt -->
  <interfacedef xlink:type="other" xlinkext:type="interface"
                xml:id="gt-start" xlink:label="start" />
  <interfacedef xlink:type="other" xlinkext:type="interface"
                xml:id="gt-end" xlink:label="end" />

  <!-- references to the other interfaces -->
  <interfaceref xlink:type="other" xlinkext:type="interface"
                xlink:href="http://www.example.com/saxtour.xml#gt-start"
                xlink:label="sax-start" />
  <interfacedef xlink:type="other" xlinkext:type="interface"
                xlink:href="http://www.example.com/saxtour.xml#gt-end"
                xlink:label="sax-end" />
  <interfacedef xlink:type="other" xlinkext:type="interface"
                xlink:href="http://www.example.com/domtour.xml#gt-start"
                xlink:label="dom-start" />
  <interfacedef xlink:type="other" xlinkext:type="interface"
                xlink:href="http://www.example.com/domtour.xml#gt-end"
                xlink:label="dom-end" />

  <!-- arcs to set up the composition of the guided tours -->
  <arc xlink:type="arc" xlink:from="start" xlink:to="sax-start" />
  <arc xlink:type="arc" xlink:from="sax-end" xlink:to="dom-start" />
  <arc xlink:type="arc" xlink:from="dom-end" xlink:to="end" />
</xlink>
```

(b) Composed Guided Tour about SAX and DOM

Figure 12: Composition of guided tours using interfaces. The guided tour `domtour.xml` is analogous to `saxtour.xml`

```
<xlink xlink:type="extended">
  <res xlink:type="resource" xlink:label="start" xml:id="gt-start" />
  <!-- ... -->
  <res xlink:type="resource" xlink:label="end" xml:id="gt-end" />

  <arc xlink:type="arc" xlink:actuate="onLoad" xlink:show="replace"
       xlink:from="start" xlink:to="sax1" />
  <!-- ... -->
  <arc xlink:type="arc" xlink:actuate="onLoad" xlink:show="replace"
       xlink:from="saxn" xlink:to="end" />
</xlink>
```

(a) Attempt to express an interface definition as local resource

```
<xlink xlink:type="extended">
  <!-- ... -->

  <loc xlink:type="locator" xlink:label="sax-start"
       xlink:href="http://www.example.com/saxtour.xml#gt-start" />
  <!-- ... -->
  <loc xlink:type="locator" xlink:label="sax-end"
       xlink:href="http://www.example.com/saxtour.xml#gt-end" />
  <loc xlink:type="locator" xlink:label="dom-start"
       xlink:href="http://www.example.com/domtour.xml#gt-start" />
  <!-- ... -->

  <arc xlink:type="arc" xlink:actuate="onLoad" xlink:show="replace"
       xlink:from="start" xlink:to="sax-start" />
  <!-- ... -->
  <arc xlink:type="arc" xlink:actuate="onLoad" xlink:show="replace"
       xlink:from="sax-end" xlink:to="dom-start" />
  <!-- ... -->

</xlink>
```

(b) Attempt to express an interface reference as locator

Figure 13: Failing attempt to express interfaces with local resources and locators

context is to be opened. This means that only one of the `xlink:show="replace"` can be traversed. XLink places no constraints on the behavior when there are several arcs "competing" to be traversed. An application might decide to traverse no arc at all or choose *any* of the available arcs, e.g. the first arc in document order. Instead of the one arc outbound from the local resource used as interface, a different arc or no arc at all might be traversed.

This is why interface definitions and references are needed. They are treated like *whole documents*, not *subresources*. When an interface is accessed, only the arcs originating from the interface itself are traversed; other arcs originating from other locations of the document containing the interface definition are not considered for traversal.

6 Linkbase Management

The need for linkbase management derives from XLink's support for out-of-line links. Linkbase management takes care of locating linkbases containing these out-of-line links and deciding when their links will actually be displayed.

This section starts with a motivation for linkbase management that also outlines the basic requirements to any linkbase management mechanism. Next, a general characterization of possible solution attempts is made. This characterization also serves as a guideline for design decisions in settling for a linkbase management mechanism. Based on this ground work, extensions to XLink for an arc-based linkbase management mechanism are suggested. Finally usage examples scenarios illustrate the mechanisms capabilities.

6.1 Motivation

A first simple example will explain how the XLink standard treats linkbase management and introduce the notion of conservative and inflationary linkbase management from [BK02b] an [BK02a].

Revisiting the example of composing guided tours from the previous section will show that neither of these is sufficient. A discussion of the influence of linkbase management on bookmarks and annotations will conclude the motivation.

Linkbase Management in XLink

Figure 14(a) shows a simple situation with two documents D1 and D2 and a linkbase B1 that contains a link L1 from D1 to D2. To display the link L1 when a user is viewing D1, the browser needs to be able to locate the linkbase.

XLink provides the special arc role

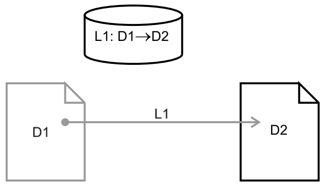
```
http://www.w3.org/xlink/properties/linkbase
```

to locate linkbases. If D1 contains an arc to the linkbase as depicted in figure 14(b), the browser is able to find it and display link L1.

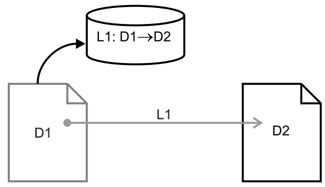
Now, let the user follow L1 to D2. As figure 14(c) suggests, D2 again has an arc to a linkbase B2 containing links L2 and L3. L2 links from D2 back to D1, L3 links D1 to a third document D3. The browser will display D2 and the link L2; L3 will not displayed since it starts from D1, not the currently displayed D2.

Conservative and Inflationary Linkbase Management

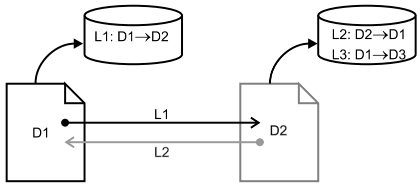
The interesting question now is, what happens if the user follows L2 arriving at D1 that he already saw before (Figure 14(d)). Obviously the browser will display D1 with link L1 since D1 has an arc to the linkbase containing L1. But what about L3? Did the browser forget about L3 when traversing L2, or not?



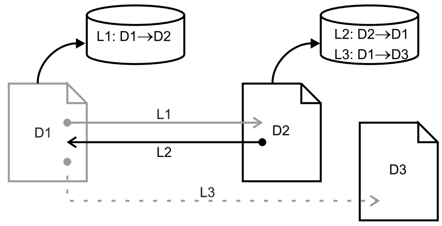
(a) Two documents D1 and D2 and a linkbase



(b) Locating the linkbase with a special arc



(c) Situation after traversal of L2



(d) Situation after traversal of L3

Figure 14: Motivating example for conservative and inflationary Linkbase-Management. Documents and Links displayed are highlighted

Asking this question leads to the distinction between a document's linkbase and a browser's linkbase. A document's linkbase contains all links contained in the document itself or in linkbases referred by the document directly or indirectly (by chaining of linkbases). A browser's linkbase contains all links the browser has knowledge of; this will usually include the currently displayed document's linkbase, but can also include links from documents previously viewed or manually loaded linkbases such a user annotations.

The situation when a browser's linkbase is always equal to the currently displayed document's linkbase shall be called *conservative linkbase management*. It is *conservative* as a document is always displayed with the same set of links.

In contrast the situation where a browser's linkbase contains all the links a browser can possibly have knowledge of, that is it is equal to the union of all document's linkbases seen so far, shall be called *inflationary linkbase management*. The number of links available to a document can increase as more and more links are collected

Both approaches have their strengths and weaknesses: conservative linkbase management 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 guided tour, its not possible to view it as a stand alone document without the guided tour links or use it in other guided tours.

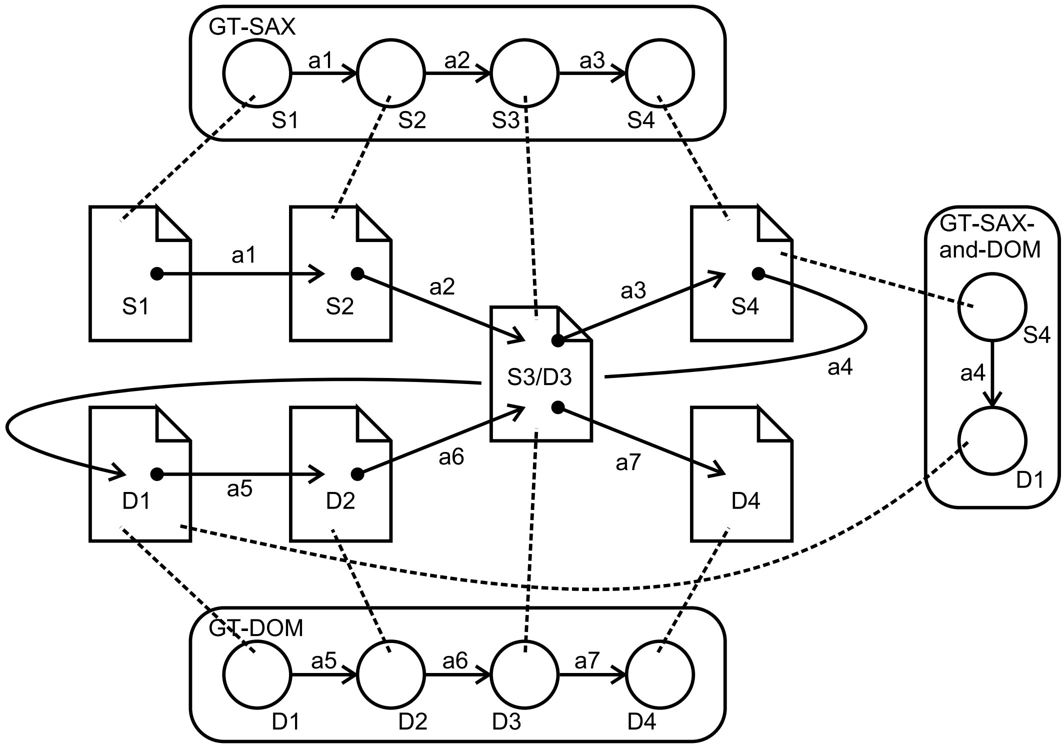


Figure 15: Composition of guided tours GT-SAX and GT-DOM with a shared resource S3/D3

Inflationary linkbase management allows a document to be presented with different links at different times. This on the other hand can confuse and disorient users. By collecting links inflationary, 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 demand much memory and computation time.

Composition of Guided Tours

Conservative and inflationary linkbase management are opposite extremes. Reconsidering the composition of guided tours from the previous section will give an example where neither is satisfactory and an in-between solution is necessary.

The two guided tours about SAX (GT-SAX) and DOM (GT-DOM) will probably contain a page displaying some input data to run their example programs on. In fact, this page might be common to both guided tours as in figure 15.

Under conservative linkbase management, the common input data page has to be aware of both guided tours. When a user arrives at S3/D3 for the first time (coming from S2), she is offered links to both S4 and D4. Going to D4 at this stage however is not in the intention of the guided tour.

Under inflationary linkbase management, all links for GT-SAX will be loaded on its first page, S1. When a user arrives at S3/D3 for the first time, she will only see the link to S4. Once she passes by D1, all links for GT-DOM will be added. Arriving at S3/D3 for the second time (coming from D2), the user will see links to both S4 and D4. Allowing a user a jump back to S4 however is not in the intention of the guided tour.

Neither of the extreme linkbase management approaches is satisfactory. What is needed is a linkbase management mechanism that can collect links, but also discard them. The goal of this section is to devise such a mechanism.

Influence on Bookmarks, History, and Annotations

A linkbase management mechanism should also take into consideration its influence on bookmarks, history, and annotations.

Suppose a user wants to bookmark a page in a guided tour as the one described above. Bookmarks, as they are common in today's browsers, are simple links having the page's URI as destination; they have no notion of the linkbases that supply a context for the page. Such a bookmark would loose the guided tour information.

With external linkbases, new "linkbase-aware" bookmarks are needed. They should be able to hold and restore linkbase information. Linkbase management mechanisms should take this into account. Preferably, it should be possible to express a linkbase-aware bookmark in XLink as an extended link.

History mechanisms in browsers have the same requirement, the history entries also need to be linkbase-aware. Since a history can be understood as a stack of bookmarks, supplying linkbase-aware bookmarks essentially solves any history-related problems.

Annotations pose a similar but different problem. Suppose a user places an annotation on a page in a guided tour. He might want this annotation to be visible no matter whether the page is viewed in the guided tour or not, or he might want the annotation to be visible only in the context of the guided tour.

Annotations are different from bookmarks. A linkbase-aware bookmark needs to *restore* the linkbases. A linkbase-aware annotation, however, needs to *check* what linkbases are loaded to decide whether they should be displayed or not.

6.2 A Characterization of Solution Attempts

Devising a linkbase management mechanism requires some design decisions. These will be outlined now.

Granularity of Linkbase Management

Linkbase management is concerned with the validity of links, that is when which links are displayed. A linkbase management mechanism has to decide on a level of granularity for the link validity information.

It could deal only with whole linkbases (XML documents containing links) displaying either all links or no links from that linkbase. Finer grained approaches would deal with fragments of linkbases, single links, or even single arcs.

Purely inflationary and purely conservative linkbase management as discussed above are very coarse-grained: the strategy (collecting or not collecting links) applies to *all* links and linkbases.

The solution proposed here deals with whole linkbases. The main advantage of this is that a linkbase has a URI to identify it; identification at a finer grained level can be hard. It is then in the link author's responsibility to split links reasonably amongst different linkbase files. Also is seems too demanding for an author to concern himself with single links or arcs without some higher grouping. Linkbases provide a simple grouping of links.

Linkbase Validity Determination

The next question is how to determine whether a link is displayed or not. Browser modi where a user can decide e.g. between conservative and inflationary linkbase management are a possibility, but they put weight on the user's shoulder.

Providing the authors with constructs in the documents to explicitly load and unload linkbases requires no user interaction. It might however be difficult to determine all places where unloading of a certain linkbase is wishful.

This work uses a solution where a "validity range", that is a set of documents the linkbase is active on, is declared by the authors. A linkbase stays active as long as the user views only documents from within this range; once he leaves the range, the linkbase becomes invalid and is automatically unloaded.

This validity range has a similarity to dynamic scope of variables in programming languages. In a dynamically scoped programming language, the scope of a variable depends on the execution order (or calling sequence) of program units. Similarly, the validity of a linkbase depends on the order documents are viewed in or arcs are traversed in.

In programming languages this temporal behavior is considered objectionable; lexical scoping, where a scope depends on the spatial layout of the program code, is preferred. In linkbase management however temporal behavior is exactly what is wanted. Documents should be able to be viewed with different linkbases depending on the traversal context, that is the documents viewed previously, as the example of the shared resource in a guided tour shows.

Unlike programming languages, hypertext has no block structures to determine a scope. The model is a flat model of nodes (documents and other resources) and links. So despite the discussed similarity with dynamic scoping, this work will use the name "range" rather than "scope".

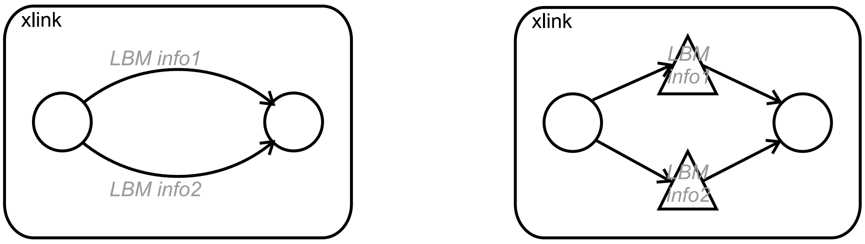
Triggers for Browser's Linkbase Changes

Another design issue is what browser actions can trigger changes in the browser's internal linkbase. Such triggers can be the loading of resources, e.g. when a new document is displayed, or the traversal of an arc.

Note that this is not the same: while traversing an arcs leads to loading a new resource, there might be several arcs leading to the same resource triggering different changes in the browser's internal linkbase.

It is also conceivable to use the locator and resource linking elements as triggers. This is midway between arcs and loading of resources. Not the arc that is traversed itself determines changes in the internal linkbase, but the locator or resource elements the arc connects do.

Manual loading or unloading of a linkbase by user interaction might be another trigger that could be offered additionally without much trouble.



(a) Linkbase management information on arcs

(b) Linkbase management information relocated to interfaces

Figure 16: Simulating arc traversal triggers by means of interfaces

Loading of a resource is the trigger closest to the XLink standard. XLink's linkbase arcs can either be traversed on request or upon loading of their source endpoint resource. The solution proposed later follows this. Note that interfaces as discussed in the previous section could be used to simulate arc traversal triggers as figure 16 suggests.

Retrieval of Linkbases

Resources on the web can be dynamic: their retrieval can yield different results at different times or under different conditions. Linkbases are just ordinary web resources and can also be dynamic.

It makes a difference whether a linkbase management mechanism retrieves a linkbase once, extracts its links, and stores them until they are no longer needed, or whether it retrieves a linkbase everytime a new document is displayed.

The latter behavior is preferable if a linkbase delivers context-dependent links dynamically. While possibly less memory intensive than storing links, it needs more computation time and adds network traffic. Retrieving linkbases everytime also poses a difficulty when a linkbase contains inbound arcs. This issue is discussed in appendix A.

In order to allow as much dynamics as possible this work's solution prefers retrieval with every new document viewed. Just like with ordinary HTML documents, caching of static linkbases is still possible to reduce computation needs and network traffic.

An alternative solution would be to allow the linkbase management information to indicate a retrieval modus for every linkbase. The linkbase management mechanism suggested here is open to such an extension.

Syntactic Constructs for Linkbase Management Information

XLink's construct to find linkbases is an arc with a special arcrole and a linkbase as destination. The solution proposed here will follow these lines and introduce new arcroles.

Other ways to provide the syntactic constructs for linkbase management information are, of course, also imaginable. A linkbase could contain a header that indicates how long it should be valid. Navigational arcs could carry attributes indicating whether all known links should be maintained or forgotten when traversed.

A solution based on linkbase arcs is very flexible: the arcs do not have to be inside the linkbase they point to (as a linkbase header would). They also allows very flexible "chaining" of linkbases, i.e. when a certain linkbase is loaded this will lead to loading other linkbases.

Note that the choice of syntactic constructs is very dependent on other design choices. The solution based on arcs makes most sense if loading of resources triggers changes. The linkbase management arc can then have the resource as starting point.

Data Structures to Store Links

Deciding on suitable data structures to store links is a question depending on all previous design choices. It seems natural to store link information as tables of source and destination points. However linkbases are themselves XML documents and thus also have the usual document tree structure; they can even be XML documents suitable for presentation such as XHTML documents.

After link information has been extracted from a linkbases document tree, it seems that this document tree could be disposed of and only a (relational) table of links maintained. This however poses a problem if a linkbase contains inbound arcs as the local resource the inbound arc points to is also lost.

A linkbase management mechanism should think of such problems. The solution proposed in this work assumes that a linkbase's DOM tree is retained if inbound arcs are present.

6.3 A Solution based on new Linkbase Arcs

The linkbase management mechanism suggested, replaces XLink's linkbase-arcrole

`http://www.w3.org/xlink/properties/linkbase`

with three new arcroles:

```
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
```

When a linkbase is loaded with an arc having one of these arcroles, the arcrole determines how long a linkbase will be kept in memory. Arcs with one of these three arcroles will be called `transient`-arcs, `temporary`-arcs, and `permanent`-arcs, respectively.

Loading Linkbases for the Current Document only: `transient`

Linkbases loaded via a `transient`-arc have the shortest validity. Their validity range is the current document, their links apply *only* to the current document. When a user leaves the document by traversing a link, the linkbase and its links are discarded. Transiently loaded linkbases have no control over documents following the current document. The `transient`-arc resembles conservative linkbase management (cf. section 6.1).

Loading Linkbases Permanently: `permanent`

The longest validity have linkbases loaded via a `permanent`-arc. Linkbases loaded this way stay in memory forever, their links apply to the current and all future documents. In other words, once they are loaded, their validity range is the whole Web. The `permanent`-arc resembles inflationary linkbase management (cf. section 6.1).

Loading Linkbases for a Limited Time: `temporary`

Between the extremes of living for only the current document or living forever is the interesting case: linkbases that are valid over a restricted range of documents and discarded when this range is left. These linkbases *temporarily* stay in memory while a user views only documents from their 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. Intuitively, when a user traverses a link to a new document, a linkbase loaded by a `temporary`-arc is discarded, *unless* there also is a `temporary`-arc from the new document to the linkbase. This intuition shall be refined in the next subsection "Processing of the new Linkbase Arcs".

Chaining of Linkbases

XLink allows linkbases to "be chained by virtue of serving as the starting resource of yet another linkbase arc" ([DMO01]). In XLink, support for chaining of linkbases is optional, a linkbase processor *may* ([Bra97]) support it. For the linkbase management solution proposed here, chaining is very beneficial and important. A linkbase processor thus *should* ([Bra97]) support it.

Chaining of linkbases is particularly useful to aid reuse. In the guided tour example from the beginning of this section, the composed guided tour GT-SAX-and-DOM makes use of two other guided tours, GT-SAX and GT-DOM. It is necessary for the composed guided tour (GT-SAX-and-DOM) to stay in memory while each of its parts (GT-SAX, GT-DOM) is viewed. This dependency can be expressed by chaining the guided tour's linkbases. Linkbase arcs *from* GT-SAX and GT-DOM *to* GT-SAX-and-DOM indicate that the composed guided tour linkbase is active whenever one of its parts is active.

Note the direction of this arc: it goes from the parts to the composite. The reason behind this is that the parts should not all be active at once but subsequently. Otherwise the parts, GT-SAX and GT-DOM, interfere at the shared resource as discussed before.

The usage scenarios at the end of this section provide more examples where chaining of linkbases is important.

6.4 Processing the new Linkbase Arcs

The intuitive descriptions of the new linkbase arcs will now be refined. Chaining of linkbases will be given special attention.

A browser's linkbase processor must maintain a set of the active linkbases. The document to be displayed is always a member of this set; initially, when there are no permanently loaded linkbases, it is the only member.

By traversing linkbase arcs, this set grows. The main concern here are arcs that are traversed automatically due to the behavior specification `actuate="onLoad"`. For a linkbase arc to be traversed, its source must be a member of the set of active linkbases while its destination 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.

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 leaves the current 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.

In principle, computing this set of active linkbases is the 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 static, adding a new linkbase to the set of active linkbases can add new linkbase arcs to the graph.

Processing `transient`-Arcs

Processing `transient`-arcs is straight forward. Linkbases loaded by means of a `transient`-arc are valid for 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

Processing `permanent`-arcs is similarly straight forward. Linkbases loaded by means of a `permanent`-arc are valid permanently, that is for the current document and all future documents. When a user leaves the current document, the linkbase stays active. This implies that permanently loaded linkbases become initial members of the set of active linkbases together with the new document.

Processing `temporary`-Arcs

Recall the intuition behind `temporary`-arcs: A linkbase that is not permanently loaded is discarded when the current document is left, *unless* there is a `temporary`-arc from the new document to the linkbase. Since chaining of linkbase is explicitly wanted, a `temporary`-

arc should also be allowed to originate from any linkbase that becomes active for the new document.

Determining whether a linkbase has to be discarded or kept bears some similarity to garbage collection (see e.g. [Wil92]): linkbases are similar to memory blocks, linkbase arcs similar to references of memory blocks. There is however an important difference. Memory blocks contain only outgoing references to other memory blocks. Linkbases, on the other hand, can contain third-party linkbase arcs that connect other linkbases than the linkbase they are physically located in. Removing a linkbase thus can remove a linkbase arc between two other linkbases.

The approach suggested here is based on growing a set of active linkbases and does not exploit similarity to garbage collection. An approach based on growing the set of active linkbases is very open to dynamic linkbases. A linkbase is not really "kept" but rather "reobtained"; that is, its content can be retrieved newly from the web.

Technically, `temporary`-arcs are given a slightly longer lifetime than normal arcs. When a user leaves the current document, all information from non-permanently loaded linkbases is discarded. The `temporary`-arcs, however, 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.

Putting it all together

Now that all new linkbase arcs have been described, figure 17 shows a possible pseudo-code algorithm to process them. The growing set of active linkbases is maintained in *activeLinkbases*; it is initialized with the new document and all permanently loaded linkbases. A set of linkbase arcs that have not been traversed, yet, is maintained in *linkbaseArcs*; it is initialized with all `temporary`-arcs and all linkbase arcs from linkbases loaded permanently.

The algorithm then grows the set of active linkbases until there is no linkbase arc a left that has its origin ($a.from$)in the set of active linkbases. Observe that the algorithm deals with cycles: arcs that have their destination ($a.to$) in the set of active linkbases are removed from *linkbaseArcs*. The only way for the algorithm not to terminate is a chain of linkbases of infinite length. Chains of infinite length are impossible with static resources, but they are a theoretical possibility with dynamic resources: picture for example 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*.

If m denotes the number of active linkbases after execution and n the number of all linkbase arcs the algorithm is or becomes aware of, its time complexity is $O(mn^2)$: The WHILE-loop is executed at most n times, each run has to check at most n linkbase arcs ($\exists a \in \mathit{linkbaseArcs}$) if their origin is in the set of at most m active linkbases ($a.from \in \mathit{activeLinkbases}$). Together this gives mn^2. Note that the CHOOSE-operation requires no additional time as it is just the WHILE-condition and the check for $a.to \in \mathit{activeLinkbases}$ in the IF-condition can be done in parallel with the $a.from$ check in the WHILE-condition.

Linkbase arcs that were not traversed when the algorithm finishes are discarded. Effectively they are lost, unless they are reinstalled by one of the active linkbases. The linkbase man-

On leaving of the current document:

$linkbaseArcs \quad := \{$all `temporary`-linkbase-arcs$\}$
DISCARD all non-permanently loaded linkbases
RETRIEVE $newDocument$
$activeLinkbases := \{newDocument\} \cup permanentlyLoadedLinkbases$
$linkbaseArcs \quad := linkbaseArcs \cup \{$all linkbase-arcs from $newDocument\}$
$\quad\quad \cup\{$all linkbase-arcs from $permanentlyLoadedLinkbases\}$

WHILE $\exists a \in linkbaseArcs. \quad a.from \in activeLinkbases$ DO

  CHOOSE $a \in linkbaseArcs$ WITH $a.from \in activeLinkbases$
  IF $a.to \in activeLinkbases$
    $linkbaseArcs := linkbaseArcs \backslash \{a\}$
    IF $a.arcrole =$ `permanent`
      $permanentlyLoadedLinkbases := permanentlyLoadedLinkbases \cup a.to$
    ENDIF
  ELSE
    RETRIEVE a.to
    $linkbaseArcs := (linkbaseArcs \cup \{$all linkbase-arcs from$a.to\} \backslash \{a\}$
    $activeLinkbases := activeLinkbases \cup \{a.to\}$
    IF $a.arcrole =$ `permanent`
      $permanentlyLoadedLinkbases := permanentlyLoadedLinkbases \cup a.to$
    ENDIF
  ENDIF

END

Figure 17: Algorithm for processing the new linkbase arcs

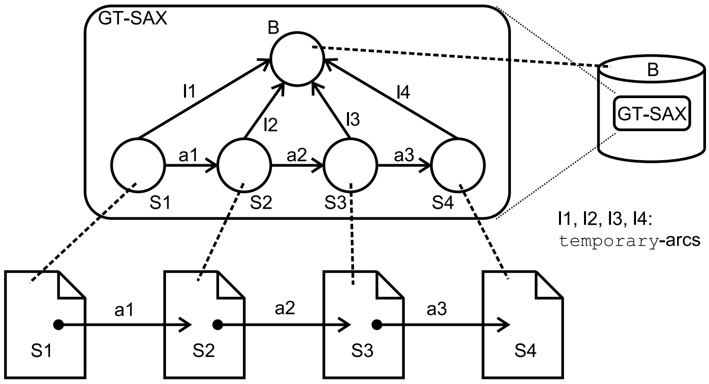


Figure 18: Using `temporary`-linkbase arcs (l1–l4) in a guided tour

agement information, i.e. the linkbase arcs, are subject to linkbase management themselves.

In practice, an implementation might want to put an upper bound on the size of the set of active linkbases. If this upper bound is reached, the order in which linkbases are retrieved is important. Criteria to choose a good order are outside the scope of this research.

6.5 Usage Scenarios

To demonstrate the potential of the three new linkbase arcs, their usage will now be demonstrated in a couple of scenarios. The examples put an emphasis on the `temporary`-arc, as this arc is the most interesting.

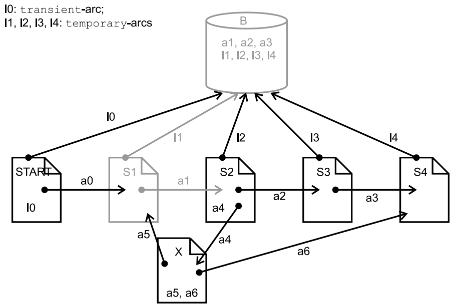
Simple Guided Tour

As discussed in the previous section, a guided tour can be formulated as one extended link (figure 6). Additionally to the arcs that interconnect the documents participating in the guided tour, linkbase arcs are needed. These linkbase arcs have to make sure that the linkbase containing the guided tour extended link stays active throughout the tour.

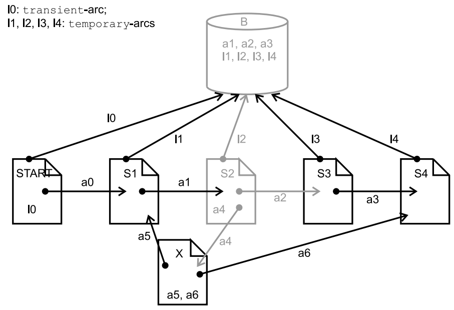
This is achieved by putting `temporary`-arcs from all the tour participants to the linkbase containing the guided tour extended link. Figure 18 shows this situation: a new locator element pointing to the linkbase is added and `temporary`-arcs go from all other participants to this locator.

It would also be possible to realize the guided tour by loading its linkbase once at the beginning using a `permanent`-arc. This however has two drawbacks: When the user decides to leave the guided tour (e.g. by following a link that is available in one of the documents and not part of the tour), the tour's linkbase stays in memory, even though it is not needed anymore. Secondly, if the guided tour should be fit for composition, permanent loading does not go well with resources shared by composed guided tour as discussed in the beginning of this section.

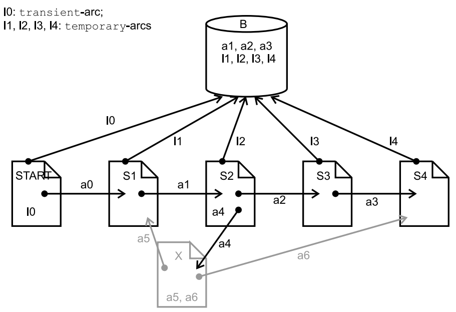
Leaving a guided tour built with `temporary`-arcs is an interesting and maybe puzzling case. Figure 19 illustrates such a case. The user accesses a document START that sends her on the guided tour GT-SAX from figure 18. She reads document S1 while the tour's linkbase B is active (figure 19(a)). When she traverses a1 to the next document S2, B stays active



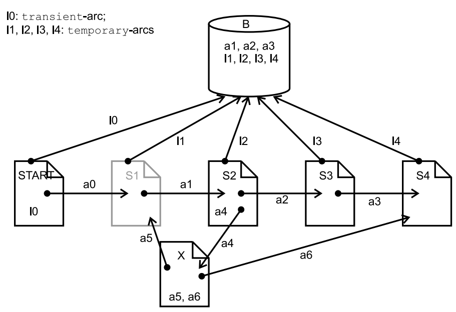
(a) User views document S1 in the guided tour



(b) User traversed a1 to S2 in the guided tour



(c) User traversed a4 to X and has left the guided tour



(d) User traversed a5 and views document S1 now without the guided tour context

Figure 19: Effect of the `temporary`-arcs (l1–l4) when a guided tour is left. The active documents and linkbases are highlighted

(figure 19(b)). Document S2 now shows not only arc a2 (from B) to the next document in the guided tour, but also an arc a3 that is contained in S2 itself. When the user leaves the guided tour by traversing a4 to X, the linkbase B is discarded (figure 19(c)).

Note what happens now when she traverses the arc a5 back to S1 (figure 19(d)). S1 now does not appear in the context of the guided tour anymore, B is not active. The reason behind this is that the linkbase management information is in the form of arcs and thus subject to linkbase management itself. When B is discarded, the linkbase arcs leading to B are discarded since they are in B.

The effect this has might seem puzzling at first, but is intended. When a user leaves a guided tour, she leaves it for good. The guided tour cannot be reentered in the middle; the only way to enter it is to access START again. Note that this also avoids the effect that a user accidentally skips documents in a guided tour with an arc like a6 in figure 19(c).

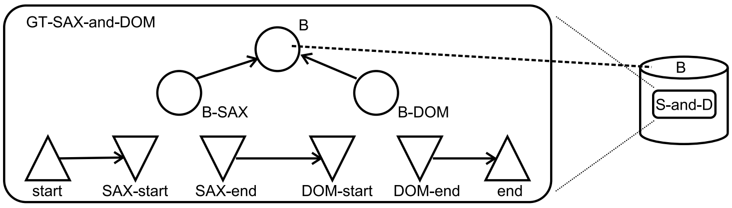


Figure 20: Chaining linkbases when composing guided tours

Composed Guided Tour

In the beginning of this section, an example of a composed guided where a resource is shared by the parts was given. This example illustrates the need for chaining of linkbases.

The parts, GT-SAX and GT-DOM, sit in linkbases B1 and B2, respectively. They use `temporary`-arcs from their documents to the linkbase as just discussed.

The composite link, GT-SAX-and-DOM, sits in a linkbase B3. Linkbase B3 has to be active the whole time a user views documents from either GT-SAX or GT-DOM. Putting linkbase arcs from all the documents to B3 breaks encapsulation: GT-SAX-and-DOM would have to know all documents from GT-SAX and GT-DOM and any change in either of them would possibly break GT-SAX-and-DOM. Here, chaining of linkbases is very useful. GT-SAX-and-DOM simply uses `temporary`-arcs from B1 and B2 to its linkbase B3. Now, whenever B1 or B2 are active, so is B3.

Figure 20 shows the composition link; note that is a variant on the composition link with interfaces from the previous section (figure 11). Using the interfaces makes sure that linkbases B1 and B2 are found when S1 and D1 are first viewed. Without interfaces, B1 would have to contain additional linkbase arcs from S1 to B1 and D1 to B2 in order to find B1 and B2. This again breaks encapsulation. Linkbase arcs from B3 to B1 and B2 are not an option — then both B1 and B2 would be active at the same time causing problems at the shared resource.

Enriching Documents with Links

Consider a set of teaching material on XML. It primarily consists of some lecture notes but also includes links to the W3C standards and other external material, maybe even the guided tours on SAX and DOM discussed so often.

In order to make the lecture notes and the external material easier to read for students, the author of the lecture notes provides a glossary of important terms and annotations. Assuming the availability of generic links, whenever the word "query" appears in the external material, a link to the lecture note's glossary entry for "query" is provided. From a link perspective, annotations are the same as a glossary. They only differ in their rendering: an annotation is usually displayed automatically in the current document while a glossary entry is display on request in a new window.

The external material can also be enriched with other links, for example links back from a W3C standard back into the corresponding chapters in the lecture notes or links from occurrences of a person's name to the person's homepage or publication list.

Since the authors of the lecture notes do not have write-access to the external material, their enriching links need to be stored in separate linkbases. The task now is to establish the connection between the documents and the linkbases.

All these enriching links are not restricted to a fixed set of documents, they are generally useful. Thus it makes sense to load the linkbases permanently. In order to do so, the teaching material's entry page would contain a `permanent`-arc to each linkbase.

A linkbase that is only useful for the lecture notes, however, can use `temporary`-arcs as described before in the guided tours examples. The `temporary`-arcs would sit in the linkbase and go from all documents of the lecture notes to the linkbase. Additionally, the entry document to the teaching material has to contain a `transient`- or `temporary`-arc in order to locate the linkbase first.

Sometimes a linkbase will be useful for only one or a few documents. In this case, `transient`-arcs from the document to the linkbase do the job. The `transient`-arcs can sit either in the documents themselves or in a linkbase that is used by all documents.

Chaining of linkbases is useful to allow sharing of annotations or other link enrichments. For example, a student can have a linkbase with her own private annotations on the W3C standards. Linkbase arcs from her private linkbase to linkbases from the lecture notes effectively *add* the teaching material annotations. Now the student can access the W3C standards directly without the detour over the the teaching material.

Such chaining of linkbases is allowed to be cyclic. Students can decide to *share* their annotations by putting links to the other students' linkbases, respectively.

Linkbase-aware Bookmarks

Figure 21 shows how a linkbase-aware bookmark can be formulated as an extended link. A local resource provides the bookmark's title and a locator points to the bookmarked page's URI; they are connected by an outbound arc.

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

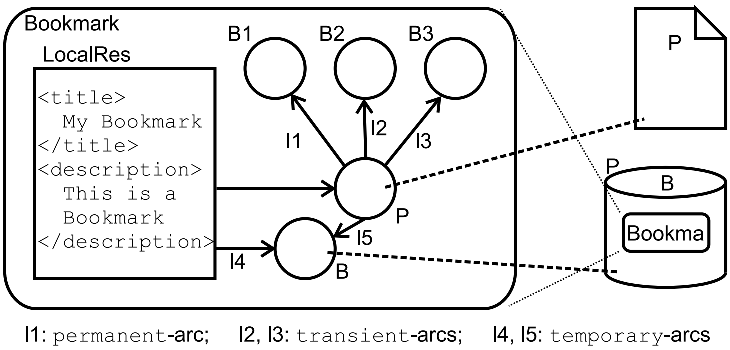


Figure 21: Realizing a linkbase-aware bookmark as an extended link

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

Linkbase-aware bookmarks also illustrate a weakness of the arc-based linkbase management mechanism put forward here. The origins of linkbase arcs are resources, identified by URIs, not locator elements. Bookmarking a page twice with different active linkbases in the same bookmark linkbase causes trouble. Activating one of the bookmarks and loading of the bookmarked page triggers not only the linkbase arcs from the activated bookmark, but also the linkbase arcs from other bookmarks of the page. In effect more linkbases are loaded than should.

While a browser can certainly find a way around this problem for bookmarks, the linkbase management mechanism proposed offers no general solution. The problem is inherent to the design choice of using resources as the triggers for linkbase changes discussed in section 6.2. Devising linkbase management mechanisms that allow to use other triggers, such as arcs or locator elements, is left open for future research.

Parallel Running Guided Tours

This final example shows that guided tours can even be "paralleled". Figure 22 depicts a situation where two guided tours are available: one could cover basic material (arcs bn in B1) while the other could cover more advanced material (arcs an in B2) of the same subject. The guided tours share some documents. Paralleling the guided tours makes the arcs from both guided tours available to the user and he can jump freely between the basic and advanced guided tour.

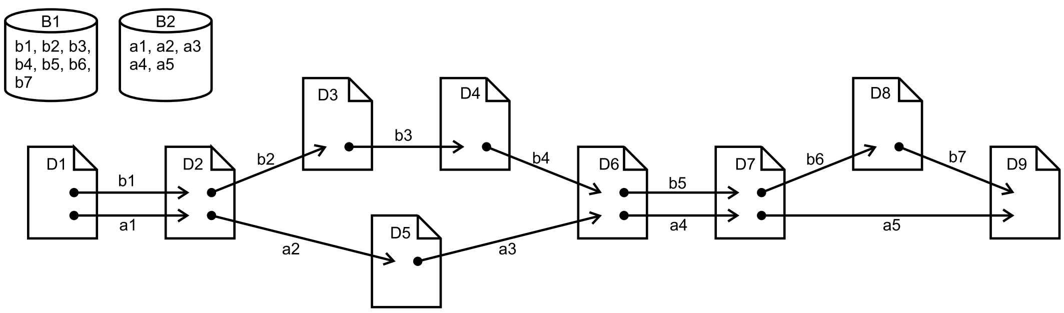


Figure 22: Paralleling two guided tours from linkbase B1 and B2

This parallelization can be expressed by putting `temporary`-arcs from B1 to B2 and from B2 to B1 into a third linkbase B3. 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.

While such a parallelization of guided tours might have only limited applicability and only be useful in rare cases, it illustrates that the the linkbase management mechanism suggested here is fairly general and powerful.

7 Discussion and Future Research Directions

This work explored XLink as a linking mechanism for hypertext on the Web. XLink's capabilities go far beyond those known from HTML. Those capabilities, especially the possibility to define links *out-of-line* in external linkbases, have been known for some time in hypertext systems other than the Web. Unlike most of these however, the Web is an open and decentral world. The Web's architectural properties make linking on the Web harder than in classical hypertext systems that have central document or link repositories.

XLink is very powerful, but it and its related standards still leave many issues open. With the discussion of these issues, this work tried to provide a roadmap to improvements in XLink and outline the difficulties solution attempts are likely to face.

As an extension to XLink, an interface mechanism was suggested. Interfaces are specially designated link participants that serve as well-defined access points to a link structure. They provide a kind of symbolic address for resources and can be used to ease reuse and combination of link structures. By decoupling a resource from its physical location, they can also be used to ease authoring and maintainance, especially if the hypertext structure changes.

This work's other contribution to XLink was a linkbase management mechanism. Linkbase management determines how external linkbases are found and when their links are displayed. It derives its motivation from the Web's decentral and open world: there are not central link servers, instead linkbases have to be discovered starting from documents a user accesses.

The mechanism suggested here tried to stay close to XLink's ideas and introduced three new linkbase arcs that are traversed when their source resources are loaded. The arcs allow a linkbase to be loaded for the current document only, for a limited range of documents, or for all documents. The power of this mechanism was shown with usage examples.

Exploring linkbase management mechanisms that are not based on loading of resources, but traversing of navigational arcs, and linkbase management mechanisms that allow a finer grained link validity below whole linkbases, seem an interesting subject for further research. Those mechanism should be interoperable with the suggested mechanism and together they could overcome the shortcomings of the suggested mechanism outlined in the bookmark usage example.

While the concrete extensions in this work concentrated on XLink, the general problems and solution ideas should apply to any linking mechanism in the Web that allows *out-of-line* links. In the Web's progress towards a separation of concerns into layers, such links become more and more important: a content's presentation information is specified in external stlyesheets, link structure in external linkbases, and meta-data such as ontologies in external resources. Links are an important backbone in establishing the connections between these layers, and allow different content or data to be viewed with different presentations, different link structure or under different meta-data information. This again motivates linkbase management to find links and to decide whether a link should be applied or not.

A Retrieval of Linkbases and Inbound Arcs

This appendix takes on the discussion of an issue with retrieval of linkbases and inbound arcs mentioned section 6.2.

The Problem

Consider the following setting: A document D is displayed in the browser to the user, and a linkbase B is active. Linkbase B contains an the specification of an extended link L. It consists of a locator to D, a local Resource R and an inbound arc A from D to R. The situation is depicted in figure 23.

Note that the linkbase B is an arbitrary XML document that does not necessarily contain only links. It can also contain data suitable for presentation, e.g. XHTML. Now, let the user traverse the inbound arc A from D to B. According to the XLink standard, the browser has to display the *whole* document B containing the local resource R with some kind of focus on R (e.g. scroll to R or highlight R), not only the content of the local resource.

As discussed in section 6.2, it is generally desirable to retrieve linkbase B freshly and not use the possibly old version present in the browser's memory. This is especially the case for resources with time dynamics, e.g. a train time table that always links to the next train leaving a station.

If the old version of B is to be used there is no problem: the browser's internal representation of arc a can have a reference to the local resource's node in the B's document tree kept in memory. The browser simply renders B from the document tree kept in memory with a focus on the local resource it holds a reference to.

However, if a new version of B is to be retrieved, the browser cannot use the reference to the local resource's node in the document tree kept in memory. This document tree is replaced with the new version of B and the reference to the local resource is lost. The browser has to find the local resource in the *new* document tree. This can be difficult.

If B is static, finding the local resource in the new document tree is not a big problem: the browser could simply identify it by its position in the document tree. This corresponds to synthesizing a simple XPointer expression (as suggested in [Dan00]) that uses element-wise navigation down the document tree to locate an element.

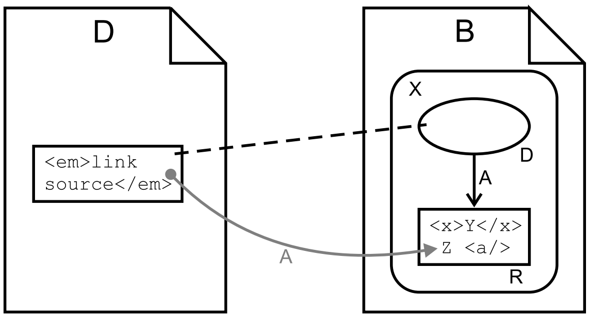


Figure 23: Situation with an external linkbase containing an inbound arc

To give an example for such a synthesized XPointer expression,

```
xpointer(/*[3]/*[2]/*[4])]
```

locates the fourth child of the second child of the third child of the document root. Such a synthesized expression can also make use of ID-attributes present. For example,

```
xpointer(id('myid')/*[3])
```

locates the third child of the element with ID "`myid`". The XPointer's element()-scheme ([GMMW03a]) offers a short-hand notation for this kind of expressions. In the element()-scheme, the examples from above can be written as `element(/3/2/4)` and `element(myid/3)`.

If B has only limited dynamics, this approach can still be applicable. For example if new elements are only added at the end of existing elements, a synthesized XPointer expression as just described will still locate the right local resource.

If B is very dynamic, locating the local resource proves to be difficult. The local resource can be moved around in the new document tree, can have changed its content, or even have disappeared completely.

Alternatively to locating the local resource in the newly retrieved document it would also suffice to locate the specification of the arc that is being traversed. This is however, in essence, the same problem. In both cases the task is to locate an element from the old document tree in the new document tree.

Solution: No Retrieval when Inbound Arcs are Traversed

The simplest approach to the described problem is simply not to retrieve destination documents of inbound arcs anew, but use the old version in present in memory.

This requires the browser to actually keep the document trees of all linkbases containing inbound arcs in memory. The browser cannot just parse a linkbase, extract link information and discard all other content. The solution thus can be rather memory demanding.

Solution: Retrieval Specification

It might seem a little unsatisfying that retrieval of documents differs depending on the type of arc that is traversed. In case of an inbound arc, the old version is used, in all other cases a new version is retrieved.

To make retrieval behavior more consistent, one could ask arcs to indicate the behavior in an `xlinkext:retrieve`-attribute. When the arc is traversed, the value `yes` indicates that the arc's destination should be retrieved newly, the value `no` indicates that the destination's old version kept memory should be used.

The solution then demands the constraint that inbound arcs are only allowed to set the `xlinkext:retrieve` to `no`. The value `yes` would be a violation of the constraint and could be indicated accordingly to linkbase authors with a warning or an error message.

Solution: Use Synthesized XPointers or IDs

A third solution is to use always retrieve, even for inbound arcs, and use synthesized XPointer-expressions as described before to locate the local resources.

In this case, it is in the responsibility of the author or the program generating a linkbase with inbound arcs to maintain the position of the local resources, so they can be found with the synthesized XPointer-expression.

If IDs are present, such a synthesized expression should make use of them and start its navigation with the nearest ancestor carrying an ID. In the best case, the local resource itself will carry an ID. Then the linkbase generating author or program has only to maintain the IDs of the local resources and can move them around freely.

Summary

This section raised an issue concerning retrieval of linkbases containing inbound arcs to local resources. If such an arc is traversed, retrieving the linkbase anew entails the danger of losing track of the local resource. This becomes especially obvious when considering dynamic linkbases where the local resource can move around in the document tree, change its content or disappear completely.

The solution ideas given here, either suppress the retrieval when an inbound arc is traversed or try their best to locate to local resource in the newly retrieved document. The latter solution is more dynamic, but requires effort from the author or program generating a linkbase and is more error-prone.

References

[ABC$^+$99] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. XSL Transformations (XSLT) Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116/`.

[ABC$^+$01] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. Extensible Stylesheet Language (XSL) Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), October 2001. `http://www.w3.org/TR/2001/REC-xsl-20011015/`.

[BCF$^+$03] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. W3C working draft, World Wide Web Consortium (W3C), November 2003. `http://www.w3.org/TR/2003/WD-xquery-20031112/`.

[BÇHL04] Bert Bos, Tantek Çelik, Ian Hickson, and Hâkon Wium Lie. Cascading Style Sheets, level 2 revision 1 (CSS 2.1) Specification. W3C candidate recommendation, World Wide Web Consortium (W3C), Febuary 2004. `http://www.w3.org/TR/2004/CR-CSS21-20040225/`.

[BK01] François Bry and Michael Kraus. Advanced modeling and browsing of technical documents. In *Track on Electronic Books for Teaching and Learning, Proceedings of 17th ACM Symposium on Applied Computing*, 2001. `http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2001-11`.

[BK02a] François Bry and Michael Kraus. Perspectives for electronic books in the World Wide Web age. *The Electronic Library*, 20(4), 2002. `http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2002-6`.

[BK02b] François Bry and Michael Kraus. Position paper: Style sheets for context adaptation. In *Proceedings of W3C Workshop on Delivery Context*. W3C/INRIA, March 2002. `http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2002-3`.

[BL] Tim Berners-Lee. Design Issues: Architectural and philosophical points. Personal notes. Available online `http://www.w3.org/DesignIssues/`.

[BL02] Tim Berners-Lee. When should I use XLink? Personal notes, 2002. Available online `http://www.w3.org/DesignIssues/XLink.html`.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic syntax, August 1998. `ftp://ftp.rfc-editor.org/in-notes/rfc2396.txt`.

[Bra97] S. Bradner. RFC 2119: Key words for use in RFCs to indicate requirement levels, March 1997.
`ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt`.

[Bus45] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.
`http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm`.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), November 1999.
`http://www.w3.org/TR/1999/REC-xpath-19991116`.

[CFMR03] Don Chamberlin, Peter Fankhauser, Massimo Marchiori, and Jonathan Robie. XML Query (XQuery) Requirements. W3C working draft, World Wide Web Consortium (W3C), November 2003.
`http://www.w3.org/TR/2003/WD-xquery-requirements-20031112`.

[CH02] Bent Guldbjerg Christensen and Frank Allan Hansen. XLink — bridging Open Hypermedia and the World Wide Web. Master's thesis, Departement of Computer Science, University of Aarhus, 2002.
`http://fahbentor.daimi.au.dk/`.

[CRF00] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In *The World Wide Web and Databases, Third International Workshop WebDB*, pages 1–25, 2000.
`http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf`.

[Dan00] Ron Daniel Jr. Harvesting RDF statements from XLinks. W3C note, World Wide Web Consortium (W3C), September 2000.
`http://www.w3.org/TR/xlink2rdf/`.

[DeR99] Steven J. DeRose. XML Linking. *ACM Computing Surveys*, 31(4es):21, 1999.
`http://www.cs.brown.edu/memex/ACM_HypertextTestbed/papers/47.html`.

[DMD02] Steve DeRose, Eve Maler, and Ron Daniel Jr. XPointer xpointer() scheme. W3C working draft, World Wide Web Consortium (W3C), December 2002.
`http://www.w3.org/TR/2002/WD-xptr-xpointer-20021219/`.

[DMO01] Steve DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), June 2001.
`http://www.w3.org/TR/2001/REC-xlink-20010627/`.

[DS04] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs); draft-duerst-iri-06. Technical report, Internet Engineering Task Force (IETF), Feburary 2004.
`http://www.w3.org/International/iri-edit/draft-duerst-iri-06.txt`.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol (HTTP/1.1), June

1999.
ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt.

[FHHD90] Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh Davis. MICROCOSM: An open model for hypermedia with dynamic linking. In *European Conference on Hypertext*, pages 298–311, 1990.
http://citeseer.ist.psu.edu/fountain90microcosm.html.

[FNP00] Gilles Falquet, Luka Nerima, and Seongbin Park. The hypertext view update problem. Technical report, Centre Universitaire d'Informatique, University of Geneva, March 2000.
http://cui.unige.ch/isi/reports/hvu.pdf.

[GMMW03a] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. XPointer element() scheme. W3C recommendation, World Wide Web Consortium (W3C), March 2003.
http://www.w3.org/TR/2003/REC-xptr-element-20030325/.

[GMMW03b] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. XPointer framework. W3C recommendation, World Wide Web Consortium (W3C), March 2003.
http://www.w3.org/TR/2003/REC-xptr-framework-20030325/.

[GSO99] Kaj Grønbæk, Lennert Sloth, and Peter Orbæk. Webvise: Browser and proxy support for Open Hypermedia structuring mechanisms on the WWW. In *Proceedings of the Eighth International World Wide Web Conference*, pages 253–268, 1999.
http://www8.org/w8-papers/3a-search-query/webvise/webvise.html.

[Hal94] Frank Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 37(2):30–39, 1994.

[HS94] Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model. *Commun. ACM*, 37(2):30–39, 1994.

[HY97] International Standard ISO/IEC 10744: Information technology — Hypermedia/Time-based Structuring Language (HyTime). Second edition 1997-08-01, 1997.
http://www.y12.doe.gov/sgml/wg8/docs/n1920/html/n1920.html.

[Kra00] Michael Kraus. A toolkit for advanced XML browsing functionalities. Master's thesis, Institute of Computer Science, LMU, Munich, November 2000.
http://www.pms.ifi.lmu.de/publikationen/#DA_Michael.Kraus.

[KRC+00] Joe Kesselman, Jonathan Robie, Mike Champion, Peter Sharpe, Vidur Apparao, and Lauren Wood. Document Object Model (DOM) Level 2 Traversal and Range Specification Version 1.0. W3C recommendation, World Wide Web Consortium (W3C), November 2000.
http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113.

[Mac02] Brett MacLaughlin. *Java & XML*. O'Reilly, April 2002.

[Mal] Eve Maler. XML linking: State of the art. Article in XML AT SUN Developer Connection. Available online.
`http://wwws.sun.com/software/xml/developers/xlink/`.

[Mar01] Jonathan Marsh. XML Base. W3C recommendation, World Wide Web Consortium (W3C), June 2001.
`http://www.w3.org/TR/2001/REC-xmlbase-20010627/`.

[MM03] Wolfgang May and Dimitrio Malheiro. A logical, transparent model for querying linked XML documents. In *GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW 2003)*, Febuary 2003.
`http://www.btw2003.de/proceedings/paper/71.pdf`.

[Moa97] R. Moats. RFC2141: URN syntax, May 1997.
`ftp://ftp.rfc-editor.org/in-notes/rfc2141.txt`.

[MV04] Jonathan Marsh and Daniel Veillard. xml:id Version 1.0. W3C working draft, World Wide Web Consortium (W3C), April 2004.
`http://www.w3.org/TR/2004/WD-xml-id-20040407/`.

[MW] Merriam-Webster online dictionary. `http://www.m-w.com/`.

[P+02] Steven Pemberton et al. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). W3C recommendation, World Wide Web Consortium (W3C), August 2002.
`http://www.w3.org/TR/2002/REC-xhtml1-20020801`.

[Pea89] A. Pearl. Sun's Link Service: A protocol for open linking. In *Proceedings of the second annual ACM conference on Hypertext*, pages 137–146. ACM Press, 1989.

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 specification. W3C recommendation, World Wide Web Consortium (W3C), December 1999.
`http://www.w3.org/TR/1999/REC-html401-19991224/`.

[UPIG01] W3C/IETF URI Planning Interest Group. URIs, URLs, and URNs: Clarifications and recommendations 1.0. W3C note, World Wide Web Consortium (W3C) / Internet Engineering Task Force (IETF), September 2001.
`http://www.w3.org/TR/uri-clarification/`.

[Wal01] Norman Walsh. XML linking and style. W3C note, World Wide Web Consortium (W3C), June 2001.
`http://www.w3.org/TR/2001/NOTE-xml-link-style-20010605/`.

[Wil92] Paul R. Wilson. Uniprocessor garbage collection techniques. In *Proceedings of the International Workshop on Memory Management*, pages 1–42. Springer-Verlag, 1992.

[WL96] Uffe Kock Wiil and John J. Leggett. The HyperDisco approach to Open Hypermedia systems. In *Proceedings of the the seventh ACM conference on Hypertext*, pages 140–148. ACM Press, 1996.

[WOL01] Harald Weinreich, Hartmut Obendorf, and Winfried Lamersdorf. The look of the link — concepts for the user interface of extended hyperlinks. In *Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia*, pages 19–28. ACM Press, August 2001.
http://citeseer.nj.nec.com/weinreich01look.html.

[YHMD88] Nichole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. Intermedia: The concept and the construction of a seamless information environment. *Computer*, 21(1):81–83, 90–96, 1988.

[Zel89] P. T. Zellweger. Scripted documents: a hypermedia path mechanism. In *Proceedings of the second annual ACM conference on Hypertext*, pages 1–14. ACM Press, 1989.