Methods for accessing Linkbases

Diploma Thesis DA-2002.03

Tutor: Dr. Erik Wilde, ETH Zurich
Supervisor: Prof. Dr. Bernhard Plattner, ETH Zurich

Christian Stillhard
christian@stillhard.net

4 March 2002

ETH Swiss Federal Institute of Technology Zurich
TIK Computer Engineering and Networks Laboratory

Abstract

Recent advancements of Web technologies, like XML, XLink and XPointer, brought us one step closer to the vision of using the Web as an *Open Hypermedia System (OHS)*. OHSes allow for openly accessing and managing linked resources. *Linkbases* are one concept of OHSes and facilitate the storage of the resources separated from the hyperlinks connecting them.

In our work we try to motivate the usage of linkbases *on the Web* by looking at different scenarios. They will allow us to analyze the role of linkbases in the Web infrastructure. We will outline the current state of research, explore the missing pieces, and show how the experiences made with OHS can be applied to the Web.

One particular aspect of OHSes is the *access* of linkbases. We analyze XLink as the Web's link interchange format and derive the implications from the accompanying linking model on access mechanisms. By including existing Web technologies and other hypermedia concepts into the discussion we work out a set of requirements for linkbase access. A main focus lies on simplicity and a quick and easy application of the concepts to the Web.

Out of the analysis we propose a *Linkbase Access Protocol*, which is called *LBAP*. It is a simple, extensible protocol that allows to access linkbases in an open infrastructure. To retrieve only relevant links from a linkbase LBAP defines a query mechanism that operates on the XLink linking model. A binding to HTTP 1.1 provides the glue to the existing web infrastructure. LBAP focusses on the necessity and a clear limitation of it's scope. It defines a common set of operations that guarantee interoperability between clients and linkbases. Extension mechanisms allow the independent evolution of the infrastructure and therefore aid the development of hypermedia concepts in the future.

The definition of a sample web service that uses LBAP and a prototype implementation of the protocol are provided.

Zusammenfassung

Die jüngsten Weiterentwicklungen der Web Technologien, wie XML, XLink und XPointer sind ein wichtiger Schritt in Richtung Nutzung des Webs als ein *Open Hypermedia System (OHS)*. OHS ermöglichen das frei Zugreifen und Verwalten von verlinkten Ressourcen. *Linkbases* sind ein Konzept von OHS, das die Speicherung von Resourcen und die Speicherung von Hyperlinks zwischen den Resourcen trennt.

In dieser Arbeit motivieren wir den Einsatz von Linkbases *auf dem Web* mit verschiedenen Szenarien. Diese erlauben uns die Rolle von Linkbases in der Web Infrastruktur zu analysieren. Wir werden den aktuellen Stand der Forschung aufzeigen, Lücken darin suchen und skizzieren, wie die Erfahrungen mit OHS auf dem Web angewendet werden können.

Ein Aspekt von OHS ist der *Zugriff* auf Linkbases. Wir analysieren XLink als das Austauschformat auf dem Web für Links und leiten die Auswirkungen des Link Modells auf die Zugriffsmechanismen her. Wir erarbeiten unter Berücksichtigung der existierenden Web Technologien und weitern Hypermedia Konzepten eine Liste von Anforderungen an ein Protokoll. Schwerpunkte sind dabei Schlichtheit und ein einfaches Anwenden der Konzepte im Web.

Unter Berücksichtigung der Analyse schlagen wir ein *Linkbase Zugriffsprotokoll (LBAP)* vor. Es ist ein schlankes, erweiterbares Protokoll das den Zugriff auf Linkbases in einer offenen Infrastruktur ermöglicht. Um die relevanten Links aus einer Linkbase zu extrahieren definiert LBAP einen Query Mechanismus der auf dem Link Modell operiert. Ein Binding zu HTTP 1.1 bindet das Protokoll in die existierende Web Infrastruktur ein. LBAP beschränkt sich auf das Nötigste und definiert eine gemeinsame Menge von Operationen die Interoperabilität zwischen Clients und Linkbases garantieren. Erweiterungsmechanismen erlauben die unabhängige Evolution der Infrastruktur und unterstützt damit die zukünftige Entwicklung von Hypermedia Konzepten.

Ein Web Service, der LBAP benutzt, und ein Prototyp des Protokolls sind Teil dieser Arbeit.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web (WWW)[1] has gone a furious path of success since its release in 1991. The result was exponential growth. According to the Internet Software Consortium[1] the number of hosts advertised in the Domain Name System (DNS) has crossed the 100 million mark sometimes in the late months of 2000. In October 2001 Netcraft [2] received responses from more than 33 million hosts providing an HTTP service.

More important than the huge number of hosts (they only provide the infrastructure), is the amount of information that is provided on the WWW. Because of the chaotic growth, the information lacks structure and organization. Web pages range from a few words to long texts, contain more or less advertising multimedia, and are written in different languages and styles. They are provided by publishers with different background, culture, interest, motivation, and varying intentions. Consequently, the quality of the content in terms of truth, relevance, and sense is unpredictable and can hardly be handled.

However, the today's significance of the WWW for economy, society, culture, and communication in general is a direct consequence of the fast development. Exactly because the Web's technology allowed an evolution without structuring- and organizational overhead, the internet can be considered a technological revolution - with it's assets and drawbacks.

The dissatisfying quality - which inherently comes with rushing change - has caused that much effort is put into finding solutions dealing with the flood of information. One approach is to improve search engine algorithms. They analyze the information structure of the Web in order to produce better ranking of the links in search results [4, 5]. Another path on the way to higher quality of information is the description of the content with metadata. As an example one can imagine metadata about a research paper containing the author, title and date of publishing. Technology like the Resource Description Framework (RDF) [6], developed by the World-Wide Web consortium (W3C), standardize models to specify information about information in a machine readable format. This information helps enriching the Web and facilitates easier and more precise search for relevant content for both - human users and machines.

The structure and organization of Web content as it is today was driven - but also is restricted - by the potential of using networked hypertext. Driven, because the simple-to-use linking functionality in HTML [7] has prompted many users to create and explore countless innovative applications on the Internet. Restricted, because the very simplistic linking model of the traditional Web lacks important functionality [9]: An HTML link is a static, directional,

[1] Internet Software Consortium, http://www.isc.org/

[2] Netcraft Web Server Survey, http://www.netcraft.com

single-source, single-destination link that is embedded into the source document. This fact affects the chaotic order of the WWW. Examples of problems caused by a restricted linking model are missing control over broken links, trial-and-error link traversal because of unlabelled links, or restriction to a single destination for a link.

More sophisticated linking models provide an approach to enhance the structure and quality of content in the traditional WWW, hence increasing the value of linking information. They provide typing of links and resources, human readable labels for links, and generally more flexible link structures. A notable feature of extended linking models is the separation of links from the content into link databases.

1.1 Motivation

The idea of richer linking models than the one found in the WWW is not new - actually they existed before the WWW was introduced. The *Xanadu* system [10] described by Ted Nelson in the 1970's was conceptually more powerful (though Xanadu was never actually implemented). Today, the emergence of XML, and in particular the linking model that accompanies it - materialized through XLink [12] - will enable much more sophisticated linking to be natively supported on the Web. With the standardization of the linking model by the W3C there is a good chance that this technology will be adopted by the leading software companies and find it's way into the implementations of web-clients and -servers.

XLink provides a new way of looking at information publishing by cleanly separating the content and the links associated with the content. In the XLink linking model, documents containing linking information about separated, external documents are called linkbases. The concept of external links and linkbases opens many possibilities for better structuring of information, for enriching the value of content with context information, and easier navigation for the user. *Open Hypermedia Systems (OHS)* [16, 17, 18] have a strong focus on the concept of linkbases. *Open Link Services* are one category of OHS's (as described by Wiil and Leggett[15]). They concern themselves primarily with the provision of hypermedia functionality to third party applications.

Currently (November 2001), the XLink linking model and the syntax to describe links are standardized by the W3C. The structure and syntax of linkbases is covered with this standard. However, it was part of the design principles underlying the XLink specification that XLink shall represent the *abstract structure and significance* of links and *not* specify precise link formatting and behavior. This decision is wise because it allows to use XLink with existing markup languages and leaves formatting decisions to the markup community. The decision is explained in the same note with another design principle:

> "XLink Shall Be Usable by a Wide Variety of Link Usage Domains and of Classes of Linking Application Software."

From the openness of XLink follows that a lot of questions concerning link formatting, behavior remain open.

The presentation of XLinks to the user is being actively discussed [13]. We hope that in spite of the complexity of the subject 'XLink-enabled' browsers will be available in the near future. However we are not aware of publicly available research covering the accessing of linkbases.

Because the success of a new linking model depends on whether it can become a part of the Web infrastructure it is essential to analyze the methods for accessing linkbases. It is the first step to implement it and make it applicable.

1.2 Goals

Within the scope of this work we will show why it is important to explore the mechanisms in accessing linkbases and propose a solution in the form of a protocol specification.

The goals in detail:

- Explore the concept of linkbases and motivate the need for sophisticated access methods with scenarios.
- Develop a mechanism to extract relevant information from a linkbase under the assumption that linkbases can be very big.
- Describe the linkbase access processing model.
- Develop a protocol specification that incorporates the query mechanism with the linkbase access processing model.
- All work is based on the XLink linking model and complies with other W3C standards.
- Provide a proof-of-concept implementation of the protocol.

The analysis of the consequences of an extended linking model to the structure and organization of the information on the Web is not subject of this work.

1.3 Outline

In chapter 2, an introduction to the concepts of the XLink linking model and an explanation of the role of linkbases are presented. It builds a basis for the discussion of scenarios of linkbases access in chapter 3. The three scenarios focus on the user point of view, they outline the participants of an access, and analyze the involved processes. Chapter 4 will address concepts of linkbase access. Based on the scenarios and the concepts a list of requirements for an access protocol is extracted in chapter 5. The specification of the protocol, covering the data and processing model, and describing the binding to an underlying protocol, is done in chapter 6. The protocol is verified in a proof-of-concept prototype which is described in detail in chapter 8. Finally, Chapter 9 and 10 conclude this work and provide an outlook on future activities.

Chapter 2

The Linking Model

The solutions presented in this work are based on the XLink linking model. The following sections outline the concepts behind XLink. Special consideration is given to the model behind the syntax - in particular to the role of linkbases in the model - and to a lesser extent to the formal specification.

2.1 XLink Terminology

> Definition: An XLink link is an explicit relationship between resources or portions of resources. It is made explicit by an XLink linking element, which is an XLink-conforming XML element that asserts the existence of a link.[1]

The term *link* describes a logical construct stating a relationship between resources. A link can combine two or more resources allowing multi-ended links. XLink markup makes a link explicit - or in other words, there is an XML element that can be identified as a link. XLink provides two types of links: *simple links* and *extended links*.
A link states the relationship between resources. Therefore it needs a construct to describe participating resources. A *locator* identifies a *remote resource* by addressing it with a URI reference. With the concept of *local resources*, resources can be directly embedded in a link in contrast to addressing it by a URI reference. A simple link is a link with exactly two participating resources - a local starting resource and a remote ending resource. It is a restricted subset of an extended link with a slightly different syntax. An extended link offers full XLink functionality, whereas simple Links offer less functionality, have no internal structure, and are oriented on the model of HTML.

With links, local, and remote resources XLink can only describe *that* resources or resource portions are in relationship. It does not express *how* these resources are related in terms of their semantic properties and also in terms of traversal. It is important to understand that XLink separates the concept of linking and traversal. An example to illustrate this separation: Imagine a news article about Josh Stone, the windsurfing freestyle world champion. In a linking element references to all resources related to Josh and windsurfing are included: NeilPride, his sponsor and sails manufacturer, Maui, his home, pictures of Josh in action, and so on. The Name "Josh Stone" in the news article is a local resource. At this stage the link only says *that* a person, a company, an island, and some pictures are in relationship but not *how* they are related. It is not yet clear how these resources are connected. Does the image show Josh in action, a crater on Maui or the sewing of a sail? Therefore the author declares a connection originating in the

[1] Section 2.1 of the W3C XLink recommendation

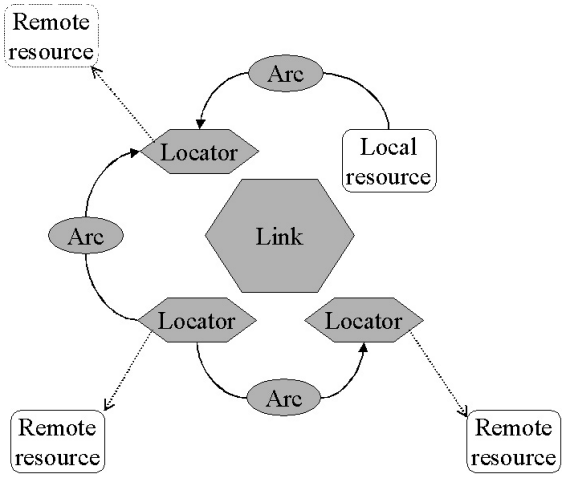

Figure 2.1: XLink linking model

name "Josh Stone" (the *starting resource* and leading to a picture of him - the *ending resource*. In Link terminology connections representing directed traversal from a starting to an ending resource are called *arcs*. One link may define a set of traversable arcs between some resources. A connection can start at the name "Josh Stone" (the local resource) and end in a remote resource like the website of Maui. In this case XLink speaks of an arc going *outbound*. An arc goes *inbound* if the local resource is the ending resource and traversal comes from a remote starting resource. A *third-party* arc doesn't involve any local resource.
Figure 2.1 shows four resources - one local, three remote - that are put into relation by a link. An outbound arc defines traversal from the local to a remote resource and two third-party arcs connect the remote resources. The resulting situation for our windsurfing example is shown in Figure 2.2

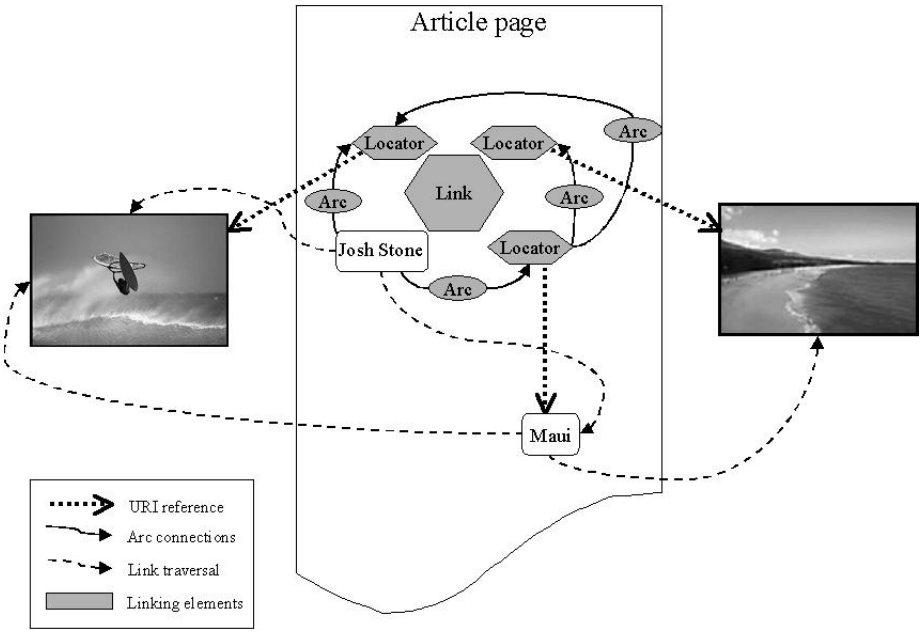

Figure 2.2: Example of an extended link

Note the difference between the two resources "Josh Stone" and "Maui": "Josh Stone" is a local resource of the link whereas "Maui" is a remote resource - referenced by a locator - although it resides in the same document.

One of the requirements for the development of XLink was: *XLink must be straightforwardly usable over the Internet*[2]. When looking at the resource "Maui" in Figure 2.2 the question arises how these two arcs can be used over the Internet when browsing page two of the article. When should the traversal to the two images be done and where will they be displayed? Additionally to the direction of traversal of an arc from one resource to another XLink defines the behavior of the traversal. For simple links and arcs, the *show* behavior property defines the presentation context in which the ending resource should be displayed. See Table 2.1 for possible values for the show property. The last column shows how equivalent behavior is achieved in HTML.

value	**treatment**	**HTML**
new	load the ending resource in a new window, frame, pane, or other relevant presentation context.	anchor element with target= "*_blank*"
replace	load the ending resource in the same window, frame, pane, or other relevant presentation context in which the starting resource was loaded.	anchor element with target= "*_self*"
embed	load the ending resource in place of the starting resource and therefore embedding it in the presentation context of the document that contains the starting resource	*IMG* element
other	the behavior is not specified by XLink. Proprietary markup in the link might specify the treatment	N/A
none	the behavior is not specified by XLink. There is no proprietary markup that specifies the treatment	e.g. the LINK element

Table 2.1: Values of the XLink show attribute.

The table shows that the control over the behavior is only possible in noticeably restricted context. The main difference of XLink to HTML lies in the *embed* value for the show attribute. This is implicitly already supported by HTML for images and some other objects like java applets. With XLink it is possible to embed arbitrary resources into a document.
The second behavior property is *actuate*. It defines for an arc or a simple link *when* traversal is initiated. The value *onLoad* implies that the traversal is triggered immediately on loading the starting resource. This is similar with the HTML IMG element with the usually chosen browser preferences. A post-loading event like a mouse click on the starting resource can trigger the traversal if the actuate property has the value *onRequest*.

Table 2.2 shows the commonly used behavior of HTML 4.0 elements that contain elements with attributes that have URI references as their value. Note that this table is not precise since behavior can usually be influenced by the user through settings in the browser. It's goal is rather to illustrate behavior in a familiar technology like HTML.
It can be seen that the concepts that XLink links introduce are not new at all. For example

[2]See http://www.w3.org/TR/NOTE-xlink-req/

Element Name	Description	Attribute with URI value	Show	Actuate
FORM	interactive form	action	variable	onRequest
BODY	document body	background	other	onLoad
BLOCKQUOTE, Q	quotation	cite	new	onRequest
DEL, INS	mark changes	cite	new	onRequest
OBJECT	generic embedded object	classid, codebase, data	embed, other	onLoad
APPLET	Java applet	codebase	embed	onLoad
A, AREA, LINK	links	href	variable	onRequest
IMG	image	longdesc	other	onLoad
FRAME, IFRAME	subwindow	longdesc	other	onLoad
HEAD	document head	profile	other	onLoad
SCRIPT	script statements	src	other	onLoad
INPUT	form control	src	embed	onLoad
FRAME, IFRAME	subwindow	src	embed	onLoad
IMG	image	src	embed	onLoad
IMG, INPUT, OBJECT	images	usemap	other	onLoad

Table 2.2: HTML 4 outgoing links and their behavior.

the *IMG* element has usually embed/onLoad behavior or the *A* element leaves the choice of the show behavior to the user (with the exception of *embed* behavior) and traversal is initiated onRequest. Many elements define other show behavior as for instance for the resource addressed by the *longdesc* attribute in an *IMG* element: In the popular Web browsers the description will be presented in a popup that occurs when the mouse hovers over the image. Some resources are not intended for presentation but rather for processing like the resource addressed by the *usemap* attribute of the *IMG* element. The table also shows that multi-ended links are already used in HTML since many of the elements contain multiple %URI-type attributes. However the difference in XLink is that multiple endpoints will be presented to the user.

A third major difference to the HTML linking model are link databases. Link databases - or shorter: linkbases - are documents containing collections of inbound and third-party links. This means that the document only contains linking information and therefore separates it from the content (with the exception of local resources in inbound links).
Figure 2.3 shows the link of the windsurfing example swapped out to a linkbase.
The extended link in the linkbase contains only third-party arcs. To display the links in the document 'article page', it needs to be aware of the linkbase, meaning that the if the content and the links are separated, the document ultimately still needs to be merged with the links and therefore a mechanism is necessary to locate the linkbase. This can easily be achieved in XLink because a linkbase or portions of a linkbase are resources itself. An outbound arc in the document with the linkbase as its ending resource and 'onLoad' as the value for actuation will load the linkbase. This simple concept and its stumbling blocks will be explained in more detail in the following chapters.
This scenario makes sense if e.g. the article page is write protected and links cannot be inserted. In the next chapters we will look at more sophisticated possible scenarios for linkbases in more detail.

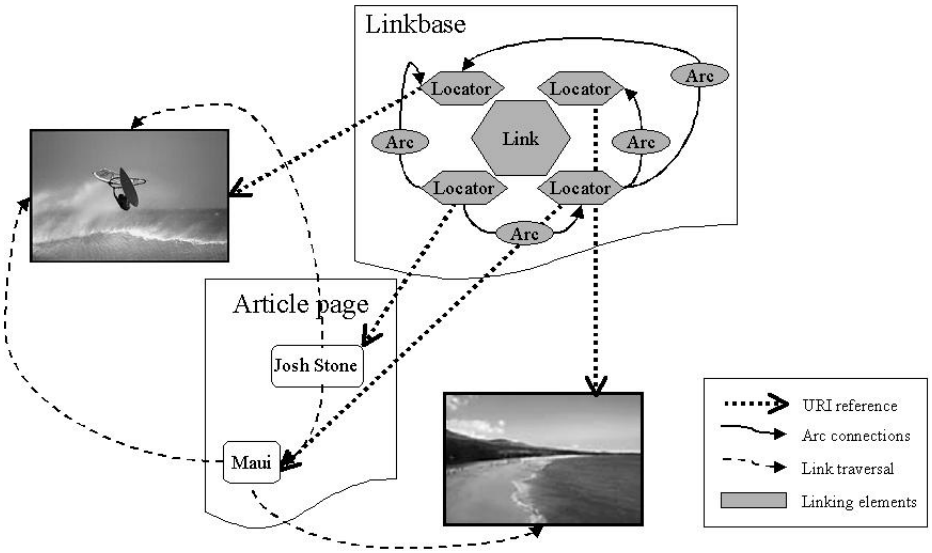

Figure 2.3: Example of an extended link

2.2 XLink Markup: Linking Language Syntax

The XLink markup vocabulary consists of a set of global attributes defined in XLink's namespace. All XLink information is defined in attributes which are used to identify elements as linking constructs and provide semantic information. Formally, it is therefore only correct to speak of locator-type elements and not just locator elements or locators. For simplicities sake we will loosen this in the future writing and stay precise only where it is necessary for the understanding.
In the subsequent lines of text the XLink attributes will be introduced and presented by the means of the windsurfing example in its linkbase variation.
The ten global attributes of XLink are listed in Table 2.3

Let's start by defining the locators needed in the windsurfing example. First we need to declare the XML elements:

```
<!ELEMENT picture EMPTY>
<!ATTLIST picture
  xmlns:xlink     CDATA           #FIXED "http://www.w3.org/1999/xlink"
  xlink:type      (locator)       #FIXED "locator"
  xlink:href      CDATA           #REQUIRED
  xlink:role      CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED
  xlink:label     NMTOKEN         #IMPLIED>
```

The Xlink attributes are attached to the *picture* element. The type attribute states that this element will serve as a locator. The respective XML for the picture with Josh in action:

```
<picture
  xlink:href="pictures/joshinwave.jpg"
  xlink:role="http://www.example.com/linkprops/peoplepictures"
  xlink:title="Josh Stone surfing"
  xlink:label="joshimg42"/>
```

Note that the type attribute defaults to 'locator' for the picture element. The href attribute holds the URI reference to the resource - the actual picture. The label joshimg42 will be used to identify this locator in an arc. The role attribute assigns a type to the resource and gives this picture a semantic meaning. According to the XLink specification this needs to be a URI

attribute	meaning
type	The type of the linking element. Can be one of the following: *simple, extended, locator, arc, resource, title*
href	required attribute for a locator- and optional attribute for a simple-type element. Contains a URI referencing a resource or portions of it.
role	semantic attribute used in linking elements and locator- and resource-type elements. It allows a typing of links and resources.
arcrole	semantic attribute for simple links and arcs.
title	provides human-readable description of an entire link, resources or arcs.
show	indicates the desired presentation of the ending resource on traversal from the starting resource. Possible values are: new, replace, embed, other and none.
actuate	indicates the desired timing of traversal from the starting resource to the ending resource. Possible values are: onLoad, onRequest, other, none.
label	identifies resources (resource- and locator-type elements).
from	defines resources in an arc from which traversal may be initiated. The value is the same as in the label attribute in the corresponding resource.
to	defines resources in an arc that may be traversed to. The value is the same as in the label attribute in the corresponding resource.

Table 2.3: The ten global XLink attributes.

reference. In this case it states that this locator references a resource which represents pictures of peoples. The title attribute could be used by a browser to display a meaningful description of the resource to the browser.

Here the declaration for and definition of the resource "Josh Stone" in the article page:

```
<!ELEMENT person EMPTY>
<!ATTLIST person
  xmlns:xlink     CDATA          #FIXED "http://www.w3.org/1999/xlink"
  xlink:type      (locator)      #FIXED "locator"
  xlink:href      CDATA          #REQUIRED
  xlink:role      CDATA          #IMPLIED
  xlink:title     CDATA          #IMPLIED
  xlink:label     NMTOKEN        #IMPLIED>

<person
  xlink:href="article/1202/thelegend.xml#xpointer(string-range(//*,'Josh Stone'))"
  xlink:role="http://www.example.com/linkprops/nameoccurence"
  xlink:title="Josh Stone"
  xlink:label="joshphrase"/>
```

The interesting part of the person locator is the URI reference in the href attribute. It contains an XPointer fragment identifier which points to every occurrence of the string "Josh Stone" in the document thelengend.xml. This is called a generic link[3] . The role of this locator is an

[3]For an explanation of the concept of generic links refer to section 2.3.5

occurrence of a name.

Lets now define the arc to define traversal to the picture:

```
<!ELEMENT go EMPTY>
<!ATTLIST go
  xmlns:xlink     CDATA           #FIXED "http://www.w3.org/1999/xlink"
  xlink:type      (arc)           #FIXED "arc"
  xlink:arcrole   CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED
  xlink:show      (new
                  |replace
                  |embed
                  |other
                  |none)          #IMPLIED
  xlink:actuate   (onLoad
                  |onRequest
                  |other
                  |none)          #IMPLIED
  xlink:from      NMTOKEN         #IMPLIED
  xlink:to        NMTOKEN         #IMPLIED>

  <go
    xlink:from="joshphrase"
    xlink:to="joshimg42"
    xlink:show="new"
    xlink:actuate="onRequest"
    xlink:arcrole="http://www.example.com/arcprops/pictureofaperson"
    xlink:title="Josh in wave action" />
```

The *from* and *to* attributes in this arc contain the labels that identify the two resources. Traversal will be triggered by the user (actuate= "*onRequest*") and the image - the ending resource - will be displayed in a new presentation context (show= "*new*").

The three elements defined until now - the two locators and the arc - are the building blocks of an extended link. Putting it all together results in an extended link:

```
<!ELEMENT joshstone ((person|picture|go)*)>
<!ATTLIST joshstone
  xmlns:xlink     CDATA           #FIXED "http://www.w3.org/1999/xlink"
  xlink:type      (extended)      #FIXED "extended"
  xlink:role      CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED>


<joshstone>

    <!-- resources -->
    <person
    xlink:href="article/1202/thelegend.xml#xpointer(string-range(//*,'Josh Stone'))"
    xlink:label="joshphrase"
    xlink:role="http://www.example.com/linkprops/nameoccurence"
    xlink:title="Josh Stone" />

    <picture
    xlink:href="pictures/joshinwave.jpg"
    xlink:label="joshimg42"
    xlink:role="http://www.example.com/linkprops/peoplepictures"
    xlink:title="Josh Stone surfing" />

    <!-- more pictures would go here -->

    <!-- arcs -->
    <go
        xlink:from="joshphrase"
        xlink:to="joshimg42"
```

```
        xlink:show="new"
        xlink:actuate="onRequest"
        xlink:arcrole="http://www.example.com/arcprops/pictureofaperson"
        xlink:title="Josh in wave action" />

    <!-- more arcs would go here -->

</joshstone>
```

This example should have given an overview of how XLink is used in XML and what the attributes in XLink stand for. The next section outlines linking concepts as they can be realized with the XLink linking model.

2.3 Linking Concepts

This section gives a quick, informal overview over the possibilities introduced by XLink in comparison with the linking in the traditional web. These concepts are described in more detail in [29].

2.3.1 Finer grained Resource Definition

The XLink linking language makes use of the XML Pointer language (XPointer) to reference subresources in XML resources. Practically, the fragment identifier within the URI reference of a href attribute (if any is used) is specified by XPointer. XPointer allows to address sub resources in a document at a character level. In our example we addressed a string range in the document. The developer has a tool at hand that allows him to be very precise. Unfortunately this fine grained addressing into a document also imposes some tough problems. For instance XPointer allows to address sub resources that are not balanced XML. In other words, a resource could contain an opening element without the corresponding closing element. The styling of such resources becomes very complicated, especially if you consider that the styling mechanisms like CSS and XSLT only address on a node level (facilitated by XPath).

In contrast to XLink, HTML usually addresses the whole resource (a document, an image, etc.). The fragment identifier in a URI reference refers to an id on an element and therefore is less precise than XPointer. But more important than the lower precision is that in HTML the mentioned *id* needs to exist in the target resource for a link. This means that the creator of a document specifies what the sub resources are (by adding an id to the elements). With XPointer an arbitrary sub resource in a document can be addressed without depending on the document creators will to allow a sub resource to be addressed. In face of this noticeable improvement of precisely referencing into an arbitrary XML resource it also has it's drawbacks. In particular the persistence of the URI references needs to be carefully controlled to avoid invalid links.

2.3.2 Multiended Links

As already seen in the windsurfing example (Figure 2.2) a link can include multiple ending resources for one starting resource. Or in other words, multiple arcs with the same starting resource can be included in an extended link. The link on "Josh Stone" in the article presents an arc to the picture showing the surfer in action and a link to a picture of his home Maui.
The consequences of this on the web are multifaceted: First of all, the presentation to the user needs to be carefully thought over. One could imagine a context menu popping up on a link presenting all arcs with their title. Additionally, the offered choice of a set of links brings up

questions about how this links should be ordered, how the relevance of a link can be expressed and presented to the user or how the links can be filtered in order to restrict the choice if the number of ending resources for a starting resource is very high. These issues are discussed in the context of work about how links can be styled[4].

2.3.3 Typed Links

The introduction of multiended links requires that the multiple resources need to be described and typed in order to allow the user to judge which one of the outgoing arcs is interesting. With XLink they can be given a title - a description of the link that is intended for presentation to the user - and a role. The role defines the semantic meaning of the link.
These concepts are an important improvement to the linking mechanisms in HTML because information is provided to the user that supports him in the decision whether he is interested in the resource behind a link or not. With an appropriate description of a link the traversal of an arc in order to see whether the ending resource is of interest or not can be reduced. We believe that this is a concept that has the potential to substantially improve the quality of the web.

2.3.4 Dynamic Links

With linkbases content and linking information can be separated. Separation means that one of them can change without changing the other one. With dynamic links the linking information is not predefined, but changes based on parameters. The parameters could be specified by the user. This is one of the major subjects of this work and the concepts will be explained in more detail later.

2.3.5 Generic Links

We have already used a generic link in the windsurfing example:

```
href="article/1202/thelegend.xml#xpointer(string-range(//*,'Josh Stone'))"
```

The sub resource is not defined absolutely but rather as a condition on the resource.
The scenario in section 3.2 motivates a more sophisticated usage of generic links.

The generic nature of a link can be increased by combining the concept of dynamic links with a link that has as its starting resource any occurrence of a word. If the resource part of the URI reference is dropped, the link references *any* occurrence of a word in *any* resource:

```
xpointer(string-range(//*,'Josh Stone'))
```

To use such a link it is necessary to resolve it for a resource before it can be used in an XLink linkbase. This is a task of a linkbase server.

We refer to both kind of generic links as such and only differentiate between them where the context requires it.

[4]See XML Linking and Style,W3C NOTE 5 June 2001, http://www.w3.org/TR/xml-link-style/

2.3.6 Transclusion

Transclusion means that parts of the content being presented are reused from an other source. The *transcluded* content is not copied or included, but rather referenced. The consequence is that no content is duplicated. Every presentation of it uses the original version. This concept, coined by Ted Nelson, a hypertext pioneer, allows to compose documents from different sources, while still keeping references to the original content.
With XLink transclusion can be modelled with show="embed" together with actuate="onLoad" behavior. The starting resource referenced by the link will be replaced by the ending resource. Replacing means that the ending resource is loaded and presented in the context of the starting resource.

2.4 Extension of Link Semantics

The standardized features to describe link semantics are limited in XLink. However, the XLink recommendation leaves space for extension of these semantics in several places. This section shows the possible extensions. They will be of importance again in the considerations concerning the openness of the Linkbase access protocol in chapter 6.

2.4.1 Proprietary Attributes

All XLink elements can have any other attribute in addition to the XLink specific ones. Consider the following example:

```
<joshstone>

    <!-- resources -->

    <!-- definition of remote resources for Josh's homepage, an picture of Josh and Josh's sponsor -->

    <!-- arcs -->
    <go
        xlink:from="joshphrase"
        xlink:to="joshhomepage"
        xlink:show="new"
        xlink:actuate="onRequest"
        xlink:arcrole="http://www.example.com/arcprops/homepage"
        xlink:title="Josh's personal homepage"
        relevance="0.9"/>

    <go
        xlink:from="joshphrase"
        xlink:to="joshimg42"
        xlink:show="new"
        xlink:actuate="onRequest"
        xlink:arcrole="http://www.example.com/arcprops/pictureofaperson"
        xlink:title="Josh in wave action"
        relevance="0.4"/>

    <go
        xlink:from="joshphrase"
        xlink:to="neilpryde"
        xlink:show="new"
        xlink:actuate="onRequest"
        xlink:arcrole="http://www.example.com/arcprops/sponsor"
        xlink:title="Josh's sponsor"
        relevance="0.2"/>

</joshstone>
```

The arcs are supplemented with the attribute *relevance* that specifies the weight of the arc. This information could for instance be used to calculate an order in which the arcs are presented to the user.
Attributes can be attached to any XLink element - linking elements, resources, arcs, etc. The possible enhancements are manifold and could range from copyright notice for resources or a price for arc traversal to presentational information like which icon should be displayed to identify an embedded resource.
As described in Section 2.2 the show and actuate attributes can take the value *other*. According to the XLink recommendation, in this case the XLink application should look for other markup that defines the link behavior. This markup will usually be an additional, non-XLink attribute in the arc-type element.
The proprietary attributes described here could of course also become standardized in future versions of XLink if they are widely used and conceptually can be seen as part of the linking model. However, this scenario should not be expected for a wide variety of additional attributes since it is a design principle of XLink that it does not favor particular usage domains or applications. More probable is therefore that particular extensions will be described in separate, application specific standards. While XLink provides the generic part of the linking semantics, modules of extensions can be added where needed.

2.4.2 Role and Arcrole Resources

The values of the role and arcrole attributes are URI references. They identify some resource that describes the intended property. The format of this resource is not standardized by XLink and therefore is an open field for proprietary extensions of semantics. One could imagine that this resource is an XML document containing a set of property descriptor elements. An element in this document is selected by the URI reference containing an XPointer fragment identifier:

```
xlink:arcrole="http://www.example.com/arcprops.xml#xpointer(//propertydescriptor[@type='homepage'])"
```

According to the definition of URIs (IETF RFC 2396), it would also be possible to specify a query in the role or arcrole attribute:

```
xlink:arcrole="http://www.example.com/arcprops.cgi?type=homepage"
```

These two examples show that the variety of role and arcrole attributes is huge and left very open by the XLink recommendation. If applications will make use of these possibilities at all it is likely that a mechanism to describe metadata about a link is chosen for better interoperability. This could for instance be RDF[6].
For most situations this goes too far and the content of the resource that is referenced by role or arcrole attributes is not of importance.

2.5 XLink Information Set

The first thing of XML to be standardized was the syntax. Later, when it became apparent that several other standards should be based on XML's data model rather than its syntax, the XML Information Set (XML Infoset)[19] was standardized, describing the abstract data model of XML documents. XLink's development is pretty similar. So far, only the XLink syntax has been formally standardized (the underlying data model is described by the specification's prose, but there is no formal model of it). However, it has already become apparent that a formal XLink Infoset, for example in the form of XLink Infoset contributions, would be useful, and Walsh[13] has published a first (but limited) approach towards an XLink Information Set.

At the time of writing, it is not clear if and when W3C will work on the XLink Infoset, but we believe that in the same way as the XML Infoset has become essential for a number of XML-related standards (eg, XQuery), the XLink Infoset will be very useful for different link-related areas, such as the presentation of links, or the linkbase access discussed in this work. We therefore assume that an XLink Information Set will become available in the near future, and as a first approach we base our considerations on the model presented by Walsh[13].

Walsh proposes a new data structure, the Link Set, and three new properties to access it. The Link Set holds an interpretation of the XLink data model: the links, arcs, and participants that are known as well as any additional metadata about them. The Link Set is constructed using Element Information items[5]. There are three types of Element information Items in the Link Set: Link Information Items, which represent links, Arc Information Items, which represent single arcs, and two Participant Information Items which represent the start and end of an arc. An example Information Set for an extended link:

```
<?xml version="1.0" encoding="UTF-8"?>
    <link role="http://www.stillhard.net/roles/family">
        <arc role="http://www.stillhard.net/roles/father" show="replace" actuate="onRequest">
            <startParticipant resource="#a" title="Jane"/>
            <endParticipant resource="#b" title="Tom"/>
        </arc>
        <arc role="http://www.stillhard.net/roles/mother" show="replace" actuate="onRequest">
            <startParticipant resource="#a" title="Jane"/>
            <endParticipant resource="#c" title="Tanja"/>
        </arc>
    </link>
```

The Information Set proposal from Walsh is not complete and does not cover the whole linking model. One major glitch is that resources can only be start and end participants in an arc. In the above example you can see that a link consists of arcs which contains participants. However, the XLink linking model allows to have link participants for which no traversal is defined. The shortcoming of Walsh's model is understandable since the proposal was developed in the context of processing XLink in order to style links. The Infoset contributions are needed to make the styling engine - whether this is the CSS or XSL or some other engine - XLink aware. Because the background is displaying links to a user the focus of Walsh's work is on link traversal. It's the arcs that need to be styled for the user.
From the XLink linking model view a link is an explicit relationship between resources. These resources are called participants. Additionally a link provides information about how to traverse a pair of resources in the form of an arc. The Infoset as proposed by Walsh sees a link as a definition of traversal between resources. This is acceptable but the resulting Infoset does not cover the full linking model. Therefore we change the Infoset to express the link participant concept explicitly.

First of all there is an element *participant* of type *participantType*, which represents a resource participating in the arc:

```
<xsd:complexType name="participantType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="link:title"/>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="role" type="xsd:anyURI"/>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="resource" type="xsd:string"/>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
```

[5]Refer to [13] for an explanation of this choice.

```
        <xsd:attribute name="traveralDefined" type="xsd:boolean" use="required"/>
        <xsd:anyAttribute namespace="##other" processContents="strict"/>
    </xsd:complexType>
```

Note the id property that assigns a unique identifier to every participant in the link. The *linkType* is extended with a list of participant information items:

```
<xsd:complexType name="linkType">
    <xsd:sequence>
        <xsd:element ref="link:participant" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="link:arc" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="title"/>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="type">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="simple"/>
                <xsd:enumeration value="extended"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="role" type="xsd:anyURI"/>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:anyAttribute namespace="##other" processContents="strict"/>
</xsd:complexType>
```

The *start and endParticipants* changed in that they don't contain the information about the resource anymore, but reference a participant through their idref property.

```
<xsd:element name="startParticipant">
    <xsd:complexType>
        <xsd:attribute name="idref" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:IDREF"/>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
<xsd:element name="endParticipant">
    <xsd:complexType>
        <xsd:attribute name="idref" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:IDREF"/>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
```

Refer to appendix A.2 for the full definition of the Infoset schema.

In contrast to Walsh's proposal the Infoset does not replicate the resources in every arc anymore. Even though replication is in this case not a question of memory consumption since the Infoset is only a conceptual model, this models an XLink link correctly; the information which resources participate in a link is available.
The usage of id's in the Infoset should not be considered to be misplaced here, again since the Infoset is only a way to express the semantics of a link.
The preceding example in our Infoset model:

```
<?xml version="1.0" encoding="UTF-8"?>
    <link role="http://www.stillhard.net/roles/family">
```

```
    <participant id="p0" traveralsDefined="true" resource="#a" title="Jane"/>
    <participant id="p1" traveralsDefined="true" resource="#b" title="Tom"/>
    <participant id="p2" traveralsDefined="true" resource="#c" title="Tanja"/>
    <arc role="http://www.stillhard.net/roles/father" show="replace" actuate="onRequest">
        <startParticipant idref="p0"/>
        <endParticipant idref="p1"/>
    </arc>
    <arc role="http://www.stillhard.net/roles/mother" show="replace" actuate="onRequest">
        <startParticipant idref="p0"/>
        <endParticipant idref="p2"/>
    </arc>
</link>
```

Chapter 3

Linkbase Scenarios

Throughout this chapter we present scenarios of linkbase access. Since XLink linkbases are not yet used on the web, the examples introduce a number of futuristic services. They show applications of the linkbase concept as it is possible to realize with XLink. They are chosen to cover a broad range of different requirements to linkbase access.

3.1 The Example

Jim Maze is reading a whitepaper about the concepts used in the latest release of the market-leading linkbase server. Jim is new to linking models and the associated XML standards. The subject came to his attention because of the manifold articles in the technology news where headlines ranged from "the new way to see the web" or "what hypermedia always was about" to "the value of links".

Jim is in a rife situation: He wants to get an overview over a new subject which involves gathering a lot of information and making it fit into a clear and structured picture in the shortest possible time. The web is full of useful information but for Jim to find the subset that suits his technical know-how, that is representative, and is not bigger than necessary is very hard. See Figure 3.1 for examples of involved resources and information concepts. The danger to loose track in the flood of new, exciting information is big. Who doesn't know this from his own experience?

To introduce the example let's have a look at the process of working into a new subject with a whitepaper as the starting point. The next few lines show the process how it could be seen in the traditional web and the following sections outline it in an infrastructure with extended linking and XLink linkbases.

> While reading Jim encounters terms he doesn't understand. He copies the phrase to the clipboard, contacts a WWW search engine, pastes the phrase into the query field and tries some of the many hits from the result. After some less successful sites - they contained the term in a completely different context - he finds the answer and returns to the paper. Later he notices that the site provides a glossary that explains some of the other terms he doesn't understand and he sticks to this aid for future lookups of unknown terms. If they are not mentioned in the glossary he queries a technical almanac or an online encyclopedia for a definition and explanation.
> He found a site that seems useful because it contains a lot of resources and link collections about XML and the accompanying standards. This reference site helps him a lot because it guides him to further reading if he wants to go deeper into a

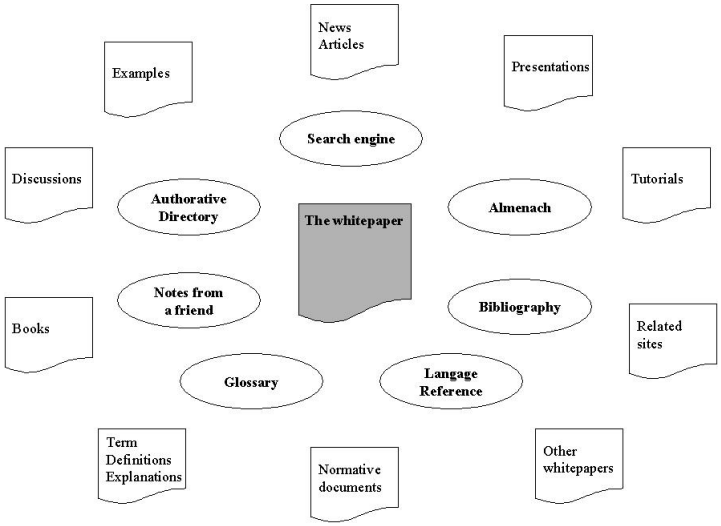

Figure 3.1: Information concepts and resources involved when reading a whitepaper

certain area.
Jim's attention for linkbases is attracted. He thinks about buying a book that is cited in the withepaper. He accesses an online book store and uses the search facility to find the book in question.
While talking to a friend they discover that this friend has written some comments and notes about the whitepaper that could very well help Jim to understand it even better. Jim receives the notes by Email.

This example showed only some possible information sources and how they are usually used and accessed. The spectrum of resources involved with the reading of a single paper is huge if we consider all information available on the Web. Linkbases can help to facilitate orientation and structure in the sometimes seemingly chaotic web. As we will see they can reduce 'copy-and-paste' actions, help to omit unnecessary link traversal, bring the information closer to the user and into the current context.

3.2 Extended Glossary

The glossary and the whitepaper are now linked by XLink links that are stored in a linkbase.

Jim comes across the term *XSLT* which he does not understand. He clicks on the term and a menu appears that offers him to view an explanation of XSLT. He clicks the menu entry and a new window opens that contains a detailed description of XSL transformations with drawings, links to further readings, etc. He closes the window and inspects the other entries that are shown in the context-menu for the term XSLT: One entry offers him to open a complete index to the glossary in case he is interested in descriptions of other terms. Another one links to a resource providing information about the author and provider of the glossary. Jim chooses the last entry which states *Enable live glossary*. Now, for all occurrences of terms that are covered in the glossary, a bubble occurs that contains a short definition of the term (Figure 3.2). Jim is delighted because he anyway would need to lookup every second term. Now he saves the two mouse clicks that led him to definition before.

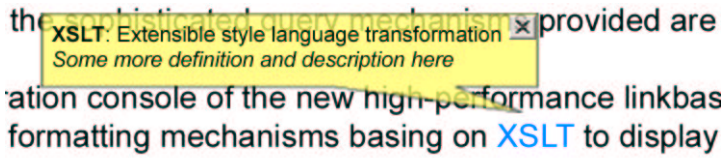

Figure 3.2: Glossary embedded into the document

3.2.1 Architecture

In the described scenario the glossary data is stored in an XML document (glossary.xml). It contains definition and description elements, one of each for every term. These elements refer to the term by an attribute called *term*. Another document contains an index for the glossary (glossindex.xml). All links are stored in a linkbase (glosslb.xml). Figure 3.3 shows one link out of the linkbase. This extended link connects the occurrence of the term *XSLT* with the according definition and description. To address the term in whitepaper.xml the concept of a generic link is used. The XPointer expression in the href attribute of the locator selects all occurrences of *XSLT* and *XSL Transformations* in whitepaper.xml. The other locator-type elements address the definition and description elements in glossary.xml.
Two arcs define traversal from the term. They differ in their behavior. The first one is triggered on request and opens a new window to display the description. The second one replaces the term with the definition resource. The definition resource could be the string *XSLT* and DHTML markup defining a layer that contains the definition in a bubble as it is shown in Figure 3.2.
The next arc defines traversal to the glossary index which opens in a new window on request. The arc enabling the live glossary is explained in the next section.
The glossary linkbase would contain an extended link of the structure just described for every term. If a new document wants to use the glossary, locators pointing to the document are added to every link.

3.2.2 Linkbase Access Aspects

The glossary linkbase is referenced in whitepaper.xml by a simple link with the special linkbase arcrole (Figure 3.4). When the document is loaded in the user-agent, it will follow the link to the linkbase and load the complete glossarylb.xml. The glossary, depending on the number of terms it contains, can be very big. All locators are transferred to the client - even if they reference a document different from whitepaper.xml and even if some or most of the terms in the linkbase do not appear in the document. The access mechanism to the linkbase needs to be finer grained than it is described in Figure 3.4 to avoid the unnecessary loading of irrelevant data. Therefore a linkbase access protocol needs to provide filtering for links starting from a specified resource.
When Jim looks at the whitepaper for the first time he does not see the bubbles with the definitions. This implies that the arcs to the definition are not available after loading. They have been filtered out when the linkbase was loaded. Or in other words only arcs with show=”new” behavior were loaded in the first place. The last arc in Figure 3.3 allows to enable the ’live glossary’ by providing a link to reload the linkbase without a filter criteria that disallows arcs with behavior show=”embed”.

```
<extendedlink xlink:type="extended"
    xlink:title="XSLT">

    <!-- resources -->
    <loc xlink:type="locator"
        xlink:href="whitepaper.xml#xpointer(string-range(//*,'XSLT'))"
        xlink:label="term"/>

    <loc xlink:type="locator"
        xlink:href="whitepaper.xml#xpointer(string-range(//*,'XSL Transformation'))"
        xlink:label="term"/>

    <!-- add locator type elements for other documents containing the term XSLT here -->

    <loc xlink:type="locator"
        xlink:href="glossary.xml#xpointer(//definition[@term='XSLT'])"
        xlink:label="definition"
        xlink:title="Definition of XSLT"/>

    <loc xlink:type="locator"
        xlink:href="glossary.xml#xpointer(//description[@term='XSLT'])"
        xlink:label="description"
        xlink:title="Description of XSLT"/>

    <loc xlink:type="locator"
        xlink:href="glosslb.xml"
        xlink:label="liveglossary"
        xlink:title="Description of XSLT"/>

    <loc xlink:type="locator"
        xlink:href="glossindex.xml"
        xlink:label="index"
        xlink:title="Glossary index"/>

    <!-- arcs -->
    <!-- Description of a Phrase-->
    <arc xlink:type="arc"
        xlink:title="View Description"
        xlink:from="term"
        xlink:to="description"
        xlink:show="new"
        xlink:actuate="onRequest"/>

    <!-- Definition of a Phrase -->
    <arc xlink:type="arc"
        xlink:from="term"
        xlink:to="definition"
        xlink:show="embed"
        xlink:actuate="onLoad"/>

    <!-- Show glossary index -->
    <arc xlink:type="arc"
        xlink:title="View glossary index"
        xlink:from="term"
        xlink:to="index"
        xlink:show="new"
        xlink:actuate="onRequest"/>

    <!-- Enable live glossary -->
    <arc xlink:type="arc"
        xlink:title="Enable live glossary"
        xlink:role="http://www.w3.org/1999/xlink/properties/linkbase"
        xlink:from="term"
        xlink:to="liveglossary"
        xlink:actuate="onRequest"/>

</extendedlink>
```

Figure 3.3: Glossary linkbase (glosslb.xml) excerpt.

```
<loadlinkbase xlink:type="simple"
    xlink:href="glossarylb.xml"
    xlink:arcrole="www.w3.org/1999/xlink/properties/linkbase"
    xlink:actuate="onLoad">
```

Figure 3.4: Link to load glossary linkbase

3.2.3 Summary

The glossary linkbase offers multiple advantages over traditional linking:

- The separation of the links to a linkbase enhances readability of the referencing document because heavy linking constructs in the text can be avoided
- Generic links allow to link multiple subresources with only one locator
- Changes in the structure of the glossary only affects the linkbase and not multiple referencing documents
- Arcs can be added and removed - or better: activated and deactivated - on demand

To make linkbase access efficient and to allow the partial delivery of links, features from a linkbase access protocol are needed. This scenario motivated the filtering for the href attribute and for the behavior attributes.

3.3 Moving the Hub to your Fingertips

Jim opens the item *Linkbases* in the menu of his Browser. He sees a list of activated linkbases and an *add* button which he clicks. In the now displayed dialog box he enters the URI of a linkbase: http://www.xmlexperts.net/linkbase. He confirms that he wants to connect to this linkbase and returns to the whitepaper. He clicks on the term *XSLT* again and now - besides the link to the description in the glossary - he sees various XSLT related links in the context menu (Figure 3.5). The links are categorized. The first category is *Publications* with links to the W3C XSLT Requirements document and the XSLT Design Principles. The next category presents *Normative Documents* and contains links to the XSLT Recommendation V 1.0 and the XSL Recommendation V 1.0. Further categories are *Articles*, *Discussions*, *Samples*, etc. He follows one of the links and a new window opens with the requested document.

Because Jim is currently only interested in Articles and Discussions and the other items in the menu only distract him he goes back to the browsers linkbase selection dialog box. By clicking on the entry of the xmlexperts.com linkbase he opens a window with configuration options. He deselects the items *Publications*, *Normative Documents* and *Samples*. Back viewing the whitepaper the menus content now only displays the requested Articles and Discussion links.

3.3.1 Architecture

The site www.xmlexperts.com offers a directory for XML related information [1]. This hierarchical directory is organized in topics around the XML standard. The site doesn't offer content for the

[1] A running site offering this kind of information is 'The XML Cover Pages' (http://www.oasis.org/coverpages)

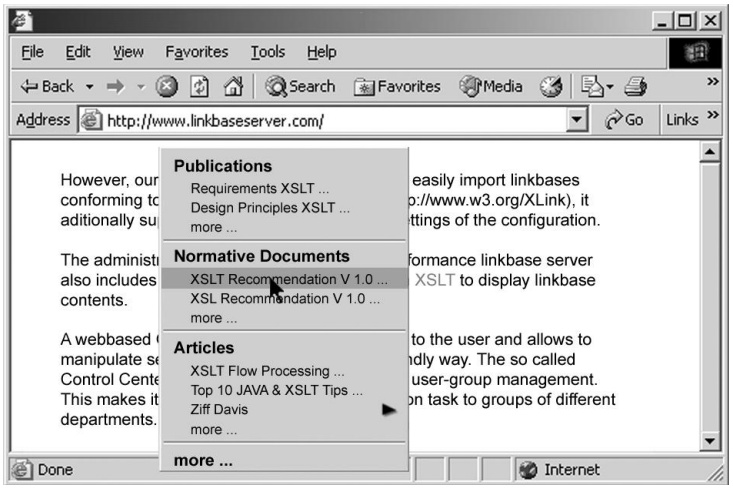

Figure 3.5: Presentation of the xmlexperts.com linkbase

topics itself but rather links to third party documents. The directory can be browsed on their site, but can also be accessed through a linkbase access mechanism. The linkbase contains links from occurrences of certain phrases on indexed sites (similar to the glossary scenario in section 3.2) to the relevant resources. The links are categorized by definitions of semantic attributes for link-, locator- and arc-type elements.
Figure 3.6 shows the structure of a particular link in the linkbase. The labels in the figure refer to the XLink *title* attribute. The XLink recommendation states in section 5.5 that

> "The *title* attribute is used to describe the meaning of a link or resource in human-readable fashion, along the same lines as the *role* or *arcrole* attribute[12].

In our case the title attribute is used to present the menu as it was outlined in the previous section. The figure shows the mapping of two levels in the directory's hierarchy of topics to an XLink. The top level results in the title attribute of the extended link, the subtopics (Publications, Normative Documents, Articles, etc.) in arc-titles. The locators refer to the simple links in the directory. Its title describes the target resource. The title attribute is therefore used to describe the semantics of the linking elements. The role attribute of the linking elements expresses the same but intended for interpretation by machines rather than human beings.
Note that the value of the title attribute could contain information about how to present the link and its participating elements. A suitable presentation could also be specified for devices or applications which don't support context menus in a XLink title-type element. But this falls in the responsibility of XLink style processing and is out of the scope of this work.
More levels of hierarchy can be modelled as a locator referencing a page that shows the sub-sub-topic and its links. A more complex variation to model multiple levels of the directory is to let the locator for the sub-sub-topic point to another link-type element in the linkbase. In this case the arcrole would be http://www.w3.org/1999/xlink/properties/linkbase indicating that traversal of the link results in loading a linkbase (or at least parts of it). The loaded links can be displayed as another menu. The menus form a multilevel context menu as it is known from user interfaces of modern desktop environments or applications. The *Articles - Ziff Davis* entry is an example of a link to a sub-sub-topic. Note that the XLink presentation software realized the different treatment of links resulting in loading a linkbase. It displays a ▷ symbol and otherwise formats it the same way as locator-type elements.
The arcs in the links all define a role attribute identifying them as different topics. The role is

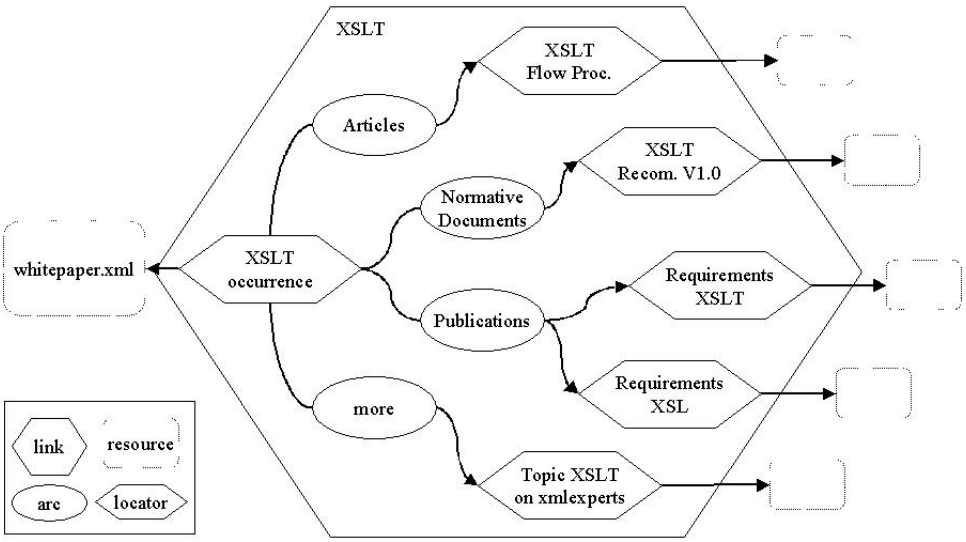

Figure 3.6: Link in the xmlexperts.com linkbase showing title attributes

the machine readable equivalent of the title attribute value. When Jim chooses which topics should be included in the menu he in fact changed a filter that filters arcs based on their arcrole.

3.3.2 Linkbase Access Aspects

As in the previously described scenarios it is again necessary to filter for the href attribute of the starting resource when loading the linkbase. It is even more obvious here since the XSLT link can contain starting resources (like the locator referencing the whitepaper) for hundreds or thousands of documents. Furthermore, the directory may contain many links (topics in the directory) which should not be loaded if they don't contain locators referencing the document in question.
In addition to the other scenarios we see the need to specify the size of the result that a linkbase access results in. The menu can't display all links that are stored for one topic. The *more...* item in Figure 3.5 indicates that not all locators are displayed. A click on the *more* icon would result in loading a separate document with more entries or open a sub-menu. It should be possible that the user-agent can indicate how many links should be returned that match the query in the request. In our example this could be a preference in the linkbase configuration.
The filtering for certain topics introduced another possible application of a query mechanism in a linkbase access protocol: the filtering for arcrole attribute values. This feature could also facilitate the suppression of loading arcs with the special linkbase arcrole. This is by the way one of the few statements about linkbase processing in the XLink recommendation; It must be possible for a user to suspend traversal of arcs to other linkbase. Realizing this in the access protocol is one possibility but it could also very well be implemented in the application that presents the links to the user - the browser or user-agent.
The decision which party in the linkbase-enabled infrastructure should fulfill a certain task is often hard to make and will be treated further in the discussion of linkbase access concepts in chapter 4.

When we are talking about the filtering of semantic attributes we should also have a look at the semantic extensions that can be made to XLink as outlined in chapter 2.4. Because the XLink standard allows to extend the ability to express semantic relationships it is legal to

think about a more complex link design for the xmlexperts directory. Imagine that after years of collecting links to articles the authors realize that they need to categorize them in *recent articles* and *older articles*. If this split should be available to the linkbase user it needs to be expressed in terms of XLink attributes. The problem is that the arcrole attribute provided for this reason is already used to mark the arc as one that defines traversal to an article resource. The authors can now 1) add another, proprietary attribute out of a namespace different than the XLink namespace or 2) introduce a special resource that is referenced by the URI in the arcrole. Both solutions require additional functionality in the query mechanism of the linkbase access protocol that allows to filter for proprietary semantic constructs.

3.3.3 Summary

The idea to provide link collections in the form of linkbases instead of a set of documents is very attractive. It brings the links right to the users fingertips by embedding it into the document he is currently viewing. The links appear in a user-defined context and it can be argued that their value is therefore higher. The browsing through the directory hierarchy or the searching for the term is not necessary anymore. The user saves time and there is no danger for him of being distracted by other, less relevant information.

From a conceptual point of view this is a good scenario for linkbase usage. The directory's intention is to provide hyperlinks and XLink provides the linking model that allows to provide them in a new way. This mainly because of the possibility to express multiended links and specify semantics. Besides the raw links, the structure of the directory contains semantics and the answer to the question whether XLink is the appropriate model to describe them needs to be left open here because we would leave the scope of this work. A vague indication that it might not be the right choice to use only XLink's linking model is the proprietary extensions we need relatively quick and the fact that XLink's linking model could be expressed in RDF[6] which would indeed cover these extensions.

It was mentioned at the beginning of this chapter that the documents using the linkbase need to be indexed with the linkbase. The linkbase then adds generic links to every topic that occurs in the document. Occurs means that the document contains a phrase that is related to the topic, like the term *XSLT* or also *XSL Transformations*. The indexing is necessary because generic links are always bound to a certain document. The part of the URI reference in front of the generic XPointer cannot be made generic as well if used in XLink. And for obvious reasons it is not possible to add a generic link to every document that is available on the web. Once because of the huge number of locators to reference them and also because they cannot be found automatically; there is no easy way to find out about all documents that are contained in http://www.example.com if the owner of the domain doesn't want to make them publicly available.

An alternative to indexing is delivering the source document in the linkbase request. The linkbase server can then dynamically resolve the generic links for this document by finding occurrences of the words it provides generic links and by resolving the URI to the address of the source document. Section 7.2.2 looks deeper into the subject of generic links and their role in accessing linkbases and presents a solution to access a generic link service.

3.4 Comments from a Friend

Jim mentions to a friend that he has read the whitepaper about the new linkbase server and that he thinks it's a great innovation. His friend, very familiar with XML technology and specialized in linking concepts, reminds him of some issues that are not yet solved in the linkbase server. He has written some notes about the whitepaper and offers Jim to read them.
Jim adds a new linkbase reference in his browser to an annotation linkbase. The friend has authorized Jim to view the annotations and after reloading the whitepaper it is enriched with comments and notes from his friend. Some shorter ones are directly embedded in the document and where there is more text or multimedia in a comment Jim clicks on an icon and the note opens in a new window.
The annotation service is free, however some annoying advertisement window opens when Jim views the whitepaper with enabled annotation functionality.

3.4.1 Architecture

This third scenario is kept short because most of the concepts should now be familiar and already appeared in the previous sections. It shows one of the main advantages of the XLink linking model: Jim's friend can change and enrich a document for which he doesn't have write access because he is not the author. Let's have a look at the annotation infrastructure and the process of annotating a document.
Figure 3.7 shows the involved components. The annotation application that Jim's friend is using

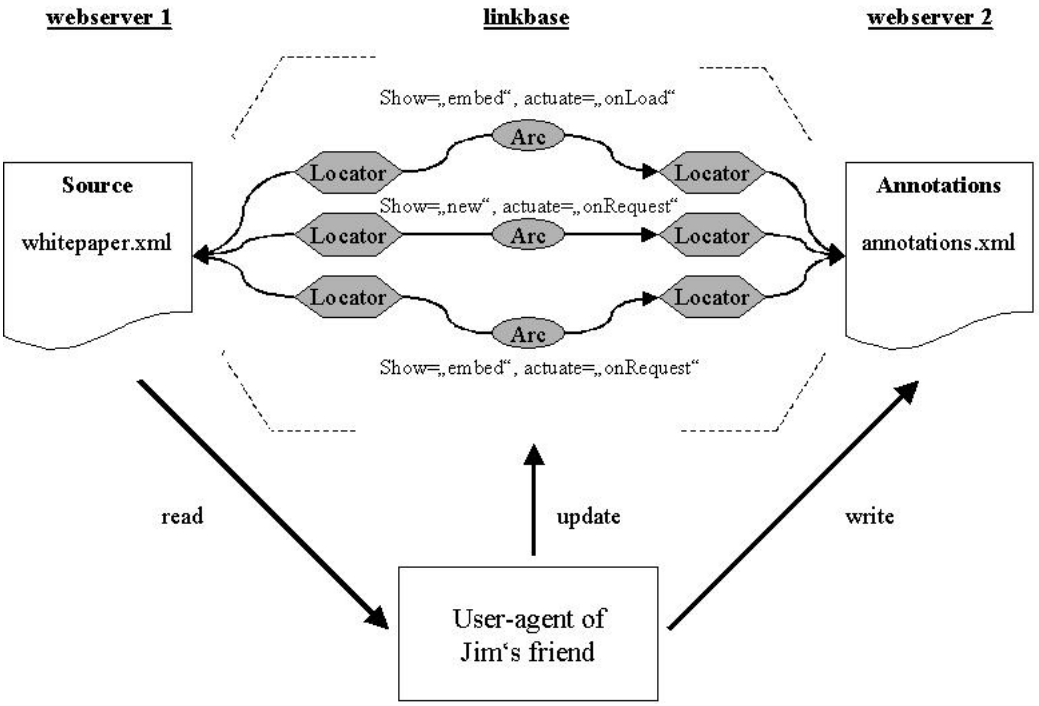

Figure 3.7: Annotation linkbase infrastructure

communicates with three servers: the webserver hosting whitepaper.xml, a webserver that stores the markup of his notes and comments, and the annotation linkbase. He reads whitepaper.xml, marks a position in the document, and writes a new annotation. Writing means he creates an

annotation element and specifies style properties, e.g by selecting an appropriate cascading style sheet (CSS[2]). The annotation document is uploaded to webserver2 and a link from the position (a subresource) in whitepaper.xml to the annotation element is added to the linkbase. He can define the behavior of the link. Larger notes which contain images or long descriptions are connected to the document with traversal behavior *actuate="onRequest"* and *show="new"*. Some smaller notes he marks as *embed note in document and load immediately* which results in adding an arc with XLink behavior attributes *actuate="onLoad"* and *show="embed"*. Even though the direct embedding of a note in a document can be very useful because it is immediately visible and opening and closing again of notes can be eliminated, this is not always straight forward to realize with XLink.
Embedding a target resource means that the presentation of the source resource is replaced with it. In the case of adding a note as we discuss it here it is not the goal to replace subresources with other resources, but they should be added or inserted. The source document needs to provide anchors to achieve this otherwise content will be replaced with a note and this is inappropriate and can confuse the reader if the note is loaded immediately. An existing innovation that presents a solution for adding anchors is pageseeder[3]. An anchor in the annotated document is called a seed and they can be added in various different ways. To come back to linkbase access, a possible solution for "seeding" a document would be that the linkbase provides an arc that triggers the automatic loading of an XSLT document. The transformation adds the seeds (anchor elements that are referenced later) and therefore prepares the document to correctly display embedded annotations.

3.4.2 Linkbase Access Aspects

As shown in the previous section with the loading of an XSLT document, linkbase access can trigger the start of a complex process. The seeding XSLT is an application of the linking model and does not pose additional requirements for linkbase access. In this scenario the additional requirement lies in the authorization of Jim with the linkbase. In order to view the annotations from his friend he needs to provide user information. Indeed this is again not directly an issue that a linkbase access protocol needs to care about. But it it has consequences on the embedding of the protocol in the existing web infrastructure. More precise - the authorization should be covered by the protocol that underlies the linkbase access protocol. For instance HTTP includes authorization mechanisms and therefore is a good candidate. A more detailed discussion about the embedding of linkbase access in the web infrastructure will follow in the following chapters.

3.4.3 Summary

The goal of the annotation scenario was, in addition to presenting the ability of XLink to link read-only documents, to make clear that linkbases very often need to be used for purposes other than just adding hyperlinking functionality in order to exploit the full power of XLink. The embedding of resources, the recursive loading of linkbases and the usage of rich semantic attributes pose inherently difficult problems. When the styling of links and embedded content is taken into account the finding of solutions becomes even more challenging. For this reason we will extract the aspects that are relevant for linkbase access in chapter 4 and logically separate them from questions about styling or the presentation of embedded content.

[2]see http://www.w3.org/Style/CSS/
[3]http://www.pageseeder.com

The World Wide Web Consortium implemented an annotation service (The Annotea Project[4]) that provides the features introduced in this scenario. It is worth noting that W3C decided for RDF to describe annotations as metadata. A member of the W3C notes[5] that XLink is not used because of it's lack of extensibility of semantic properties. He argues that the decision is reasonable because the design of Annotea is ambitious and goes far beyond only presenting human navigation links by modelling sophisticated semantics.

3.5 Conclusions

The scenarios made clear that there are attractive applications for linkbases. The separation of links from the content allows insertion of links into read-only documents and dynamic control over the links that are presented to the user. The value of hypertext can be augmented by enriching content with links from third parties. Typed and labelled links increase the quality of linking information. Additionally, we have shown that complex hierarchic link structures can be modelled in linkbases and presented to the user.

However, the linkbases can contain a great many of linking elements when the full benefits of extended linking are explored. Generally used linkbase services that are targeted towards the linking of a numerousness of documents - as the one provided by xmlexperts.com in the second scenario - need to facilitate the filtering of links. The scenarios showed that usually only traversal information that originates in a certain document is relevant for a linkbase user - namely, the document he has currently loaded. All other linking elements should be filtered out and not delivered to the user. In all the scenarios we saw that the query mechanism needs to provide access to various properties of a link, especially the semantics, to allow precise filtering.

To conclude, linkbases hold great potential to rectify the quality of hypertext. But to explore it, an access mechanism is needed. Otherwise either linkbases can only be used to provide links for a small number of documents - which would reduce their value - or the linking information will be as chaotically presented as the content of the web today - which is against the goal to improve quality of hypertext. A certain irony of this situation cannot be overseen.

[4] `http://www.w3.org/2001/Annotea/`

[5] Ralph R. Swick on the xml-dev mailing list on the 17th of November 2001

Chapter 4

Linkbase Access Concepts

4.1 Entities of Open Hypermedia Systems

Linking in hypermedia is a wide field. A link can connect an entry in a document's table of contents with a chapter, or it can put a pattern in an audio stream of a speech in relation with a position in the textual presentation of the speech. It can link from the name "Nelson" to the Battle of Trafalgar where Admiral Nelson defeated Napoleon's navy or to the concepts created by the hypertext pioneer Ted Nelson. A link connecting the phrase "Bentley" with a document about noble cars can be created by the author of the document containing the phrase or it can be created automatically because a knowledge base states that a "Bentley" is a noble car.
As can be seen from these examples linking involves various media, a link is more than just a physical connection - it has semantics, and creation of a link can be achieved in many different ways. These are just a few facets of linking that make the subject very fascinating, but also very complex. It is therefore important for this work to outline some linking concepts and specially the concepts that are involved in extracting links from a linkbase.

In a paper about the semantic web at the 10th World Wide Web Conference 2001, Wendy Hall et Al., members of the Intelligence, Agents, Multimedia (IAM) Research Group at the Southhampton University, introduce three entities that are the base for the semantic web: Ontologies, the hypertext-, and the web-infrastructure[1]. Ontologies describe semantic knowledge (e.g. a "Bentley" is a car). Colloquially spoken, the goal of ontologies is to describe the world by logically combining assertions about it. The semantic information can then be used to link concepts that appear in documents. From a linkbase point of view, the hypertext infrastructure consists of the linking model and the linking language (as for instance defined by XLink), applications to display and present the links in resources, and services that store and provide the links. The web infrastructure is the well known, standardized, well scaling, open network that enables the delivery of information - whether this is semantic knowledge or hypertext information. These three entities are necessary to build a basic infrastructure for an open hypermedia system. The semantic information from the ontologies is used to create links in documents. The links are then stored and presented by applications from the hypertext infrastructure. The delivery of links is supported by the web infrastructure.

To explore linkbase access concepts three questions are discussed in this chapter: What is the role of linkbase access in the semantic web, what is the functionality that a linkbase service needs to provide and what needs to be considered in terms of the general framework of the current web infrastructure.

[1] Slides on http://semanticweb2001.aifb.uni-karlsruhe.de/slides/goble.pdf

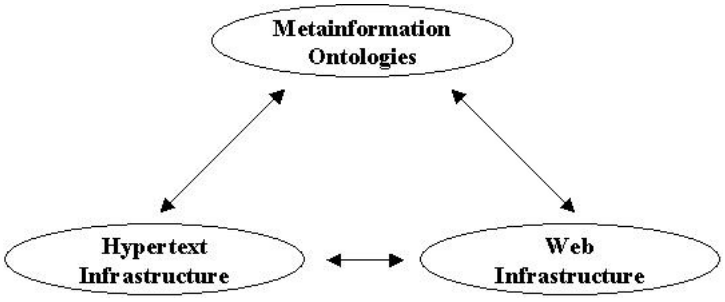

Figure 4.1: Entities of Open Hypermedia

The first step is to isolate the linkbase access from higher level concepts like ontologies, etc. A clear separation from linkbase access as part of the hypertext infrastructure from link creation - or better from the processing of semantic information - is important as will be shown in section 4.2. Even though it appears obvious here, these two different things can easily be mixed up if taken into account that the linking model of XLink allows to describe semantic information. An example that mixes these two things should make this point clear: A linkbase query could state: "Return all links for this resource that involve the concept of cars" (A concept describes a node in an ontology). Such a query breaks out of the hypertext infrastructure and involves ontologies. However it is very close to a query that states: "Return all links for this resource that define traversal to resources whose role is 'http://www.example.com/roles/car'". We would need to consider queries of the same kind as in the former example if there was an agreement on the representation of semantic information by one widely used standard - from which we are far away.

With a clear idea of what the role of linkbase access is, section 4.3 discusses infrastructural aspects of linkbase access. We will have a closer look at characteristics of linkbase servers, XLink enabled user-agents, topologies and the mapping of the communication with linkbases to existing protocols.

The last section (4.4) provides an insight into the actual processing of linking information by a linkbase server. The results of this section are the foundation for the design of requests that can be sent to a linkbase server (as described in chapter 7).

4.2 The Role of Linkbase Access

4.2.1 Static and Dynamic Context

The distinction between semantic knowledge and the hypertext infrastructure can be highlighted from an other point of view by focusing on the linkbase access itself. We assume that all links are created and are stored in the linkbase. They include semantic information - for XLink in the form of role attributes. The main goal of the linkbase access is to get *relevant* links. The *relevance* of a link is a question of definition and can be influenced by many factors. Conceptually it depends on the context in which the linkbase access is triggered. One part of the context is - in the case of browsing the web - the document that is currently active. The active document is the one being loaded or displayed, depending on the actuation of the linkbase access. Links that don't originate in the active document are not relevant (even though this is inherently true for browsing the web, it is not for e.g. robots that analyze the contents of a linkbase). This context is static. Dynamic context is information about the user. That is

the users interests. If he is interested in cars only, then a link that connects "Bentley" with the Bentley College is not relevant in this context.
So, again conceptually, the situation before a linkbase access is that on the client side there is a certain context (static and dynamic) and on the server side is a linkbase. The linkbase access should result in delivery of relevant links with respect to the context. We will now discuss what steps are involved to get the result.

4.2.2 Layers of Abstraction

Figure 4.2 shows four layers of abstraction. If we start from the view of a linkbase as implicated by the XLink recommendation, then a linkbase is an XML document that distinguishes itself from any other XML document by using attributes from the XLink namespace. Consequently the lowest possible level of abstraction for a linkbase access is the XLink syntax. All operations are performed on the XML information set. The second level abstracts the representation of a link with the linking model. The highest two levels are exemplary for higher level abstractions. An elaborating analysis of them would go beyond the scope of this work but they are actively discussed in research about hypermedia (e.g. by the OHS Working Group[2] or the semantic web community[3]) and from a more general view in research about knowledge representation and reasoning[4]. The *Link Semantics* layer accommodates the modelling of link semantics - the semantics of the semantics. The *Link & context* layer builds on the semantic web (modelled by the layer below) and provides an abstraction of user interests. Note that a model of user interests not necessarily operates on the semantic web. But for our illustrative purposes we assume that it does.
Linkbase access can potentially be performed on every layer. However, as indicated in the picture, the result of a query is returned in form of a XLink linkbase, encoded in the XML Linking language. This is the only requirement that ultimately needs to be fulfilled. It is the bases for the hypertext infrastructure for the web. The XML Linking language and the accompanying linking model are also the only part in the infrastructure that are standardized. The handling of links at the client- and server side is not fixed. We also assume that the linking language is the input for the link presentation software. With these assumptions this discussions differs from other hypermedia research. Our starting point is the linking model and its encoding. Other work puts the main focus on the infrastructure and leaves the linking model open (e.g. OHS [16, 17, 18]). But to really bring extended linking to the web it is important to have the "lowest layer" fixed.

Let's discuss the implications of linkbase access on the different layers. In general the layer on which the query is exchanged defines what functionality the linkbase server needs to provide. If a query is performed on layer one, then a linkbase server does not need to know anything about the linking model, semantics or even about the concept of user interests. It simply applies the query to an XML document. If in contrast the linkbase access is modelled in layer four, then the linkbase server needs to be able to interpret a user interest and select relevant links based on it.
The requirements on intelligence that a client side application needs to provide is higher when the linkbase access is performed on a lower layer. The client application is responsible to describe the dynamic and static context of an access. If a query is performed on layer four than the encoding of the context into a query is all that needs to be done. If the query is performed

[2]OHS Working Group website: http://www.csdl.tamu.edu/ohs/
[3]W3C semantic web activity: http://www.semanticweb.org/
[4]KR Inc. organizes the yearly Knowledge Representation and Reasoning Conference: http://www.kr.org/kr/

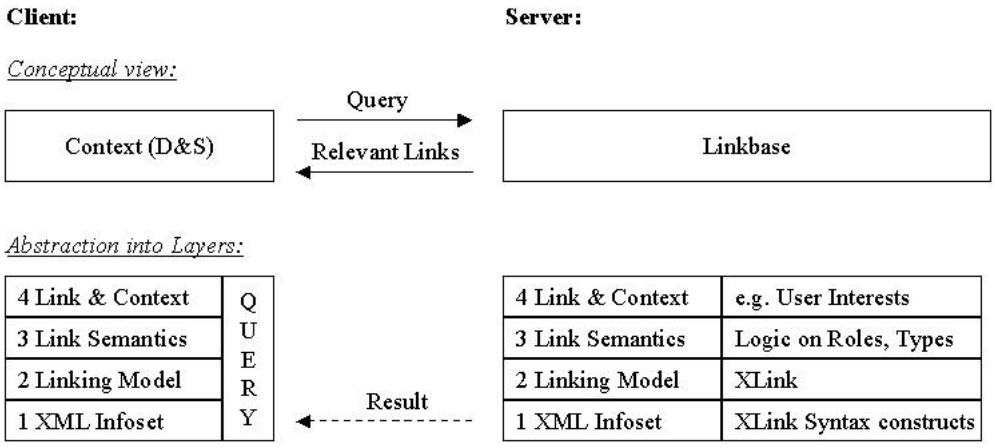

Figure 4.2: Abstract Layers involved in OHS

on a lower layer then the context needs to be transformed under consideration of the link semantics, the linking model and maybe even the linking language.
For the design of our linkbase access protocol we need to agree on the layer on which it works. This decision implies the requirements to the intelligence of both, the linkbase access client and the linkbase server. While it is desirable to free the client of interpretation of context information it is not appropriate to require the server to understand a defined set of user interests. This decision is of course also influenced by the agreement of the encoding of e.g. user interests.

To highlight the implications we discuss now queries on every layer. The tasks to compile the query on the client side, the application of it in the linkbase server are considered. Additionally, for an open hypertext architecture it is necessary to provide mechanisms to query a linkbase for metadata. For a client to perform a query it might be necessary to know e.g. the roles that are used in the linkbase.

4.2.3 Layer 1, Linking Language

On layer one, the linking model is not taken into consideration. The basis is the markup as defined by XLink, in particular the XLink attributes and their values. A linkbase is simply an XML document with global attributes from the XLink namespace.
The filters and conditions in a query are applied on the XML syntax elements (elements, attributes, entities, etc.) without considering their meaning in terms of the linking model.

Query

A query could be written as an XQuery SELECT statement or an XSL stylesheet that transforms the full linkbase into a subset of it.

Linkbase server

The part of the linkbase server that handles queries is not aware of the linking model. It executes the query on the linkbase document and delivers the result. The storage of the linkbase is not necessarily an XML document. The query could also be transformed into a SQL statement that

is executed against a RDBMS holding the link information. The intelligence of the linkbase in terms of creating links can of course be more complex, but it is not required to process a query.

Client side

The clients task is to build up the query. This involves two major tasks: (1) interpret the context and express it in terms of the linking model and (2) build a query that contains the structure of the linking model. The former will be discussed later. The consideration of the linking model in a query that only acts on the linking language is necessary to avoid corruption of the information that is contained in the linkbase. Consider the following case:
A very simple linkbase consists of one extended link that contains two locators. One of the locators points to an image resource showing a person, the other references the persons homepage. The extended link contains one arc-type element that defines traversal from the picture to the homepage. If a linkbase client requests traversal information the originates in the homepage, then the only arc in the link would be filtered out. The resulting linkbase is valid XLink even with no arcs defined and the XLink recommendation states in section 5.1.3 that the link is to be interpreted as defining traversal from every resource to every other. This example shows that the filtering resulted in a linkbase defining more traversal information then it originally contained. A query would therefore need to consider such situations and become quite complex.

To summarize: After interpreting the context, the client comes up with a query that operates on the linking model. For example, give me all arcs that define traversal from a certain resource. This query needs to be transformed to a query operating on the linking language. That step involves encoding the linking model in the query. For example: Give me all arcs which have a from attribute that matches the from attribute of locators that have the URI reference of the starting resource as their href attribute. And also give me all arcs with no from attribute specified, if there are locators in the link that have a matching href attribute, and so on.

Metadata

In an open environment the client accesses linkbases which are not known at the stage of creation of the document to be linked. An example is the scenario "Moving the hub to your fingertips" in chapter 3.3. To create a query the client needs to know e.g. what roles are used in the linkbase. On layer one, this metadata constitutes of a list of values of XLink attributes.
Because the linking language is fixed, the basic elements of the query are defined and don't need to be described with metadata. More concrete: the Query defines conditions and filters for the attributes of the XLink namespace. In contrast, to build a query on a higher level, operating on e.g. user interests, the client application needs to gather information about the form of encoding of user interests. This is not inherently true, but comes from the fact that the description and encoding of user interests is not standardized and can take various forms.

Evaluation

Advantages: The main advantage of performing a linkbase access on layer one is that the requirements on the linkbase server are very small. In fact the only reason to call it a linkbase server is because it handles XML documents that contain linkbases. It "only" needs to provide a engine that executes queries on XML documents. An XQuery service would be enough. Another advantage is that the query can be fully expressed by standardized languages - whether this is XQuery, a subset of XQuery, an XPath expression or a XSL stylesheet.

Extensions of XLink can be handled by the protocol very easily because it is based on XML and not bound to the linking model.
Disadvantages: The encoding of the linking model in the query is a complex task. Writing a correct query by hand is nearly not possible. Queries will not be human readable.

The linkbase access needs also be highlighted under consideration of architectural aspects. Performing the linkbase access on the linking language layer also means that for all higher level abstractions there is no restriction. E.g. the way how link semantics are evaluated and applied is not bound by the access protocol. Other services will be part of the infrastructure to facilitate the management of this abstractions. For example there can be services that manage user interests.

4.2.4 Layer 2, Linking Model

On layer two, mechanisms like the defaulting of traversal attributes don't need to be considered anymore. Operating on the linking model means to operate on links, arcs and resources and not XML elements with a type attribute that has a value of "extended", etc.
The linking model of XLink can be described and made explicit as contributions to the XML information set (as described in chapter 2.5). A linkbase could therefore also be stored as information set encoded as XML elements. Note that the information set is only one way to express the linking model. It is the way to make it explicit.

Query

One way would be to assume the XML information set to be the representation of a linkbase. A query could then be expressed as an XQuery statement or an XPath expression that operates on the information set. Another way would be to introduce a new query language that is designed to operate on the linking model.

Linkbase server

The linkbase server needs to interpret the XML linking language and provide operations on the model. Whether it internally stores the links in the XML linking language or in some other way is the matter of the server implementation. As we have seen for the queries it would be possible to have an XQuery supporting service that executes these queries on the information set items of the linkbase.

Client side

The client is no longer forced to construct complex queries that operate on XML. It can focus on higher abstractions.

Metadata

From a content point of view, the metadata that needs to be delivered is the same as if we operate on layer one. However, it would be described differently. Namely in terms of XLink model items instead of attribute values.

Evaluation

Advantages: The client side compilation of complex queries is not necessary.

Disadvantages: A linkbase server needs to be XLink aware.

4.2.5 Layer 3, Link Semantics

The real power of links comes with their semantics. By interpreting the semantic information the value of a network of linked resources is fully exploited. Figure 4.3 shows a link with six resources. For some of them traversal is defined with explicit arcs (indicated by full lines). The arcs have semantics indicated by the arcrole (labels of the arcs in the Figure). When the semantic

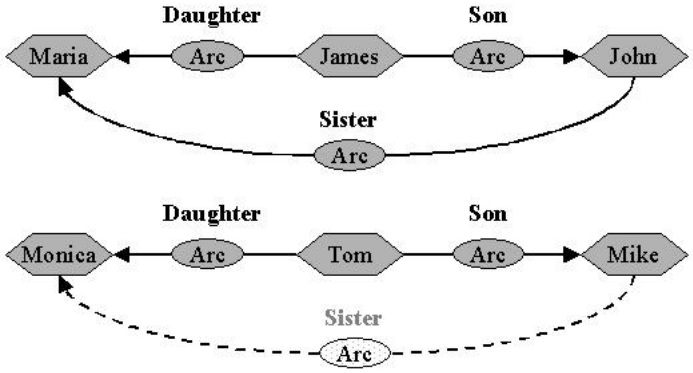

Figure 4.3: Semantic logic. Implicit Arc.

information, here the daughter, son and sister roles are interpreted it follows implicitly (dotted lines) that Monica is the sister of Mike.
The interpretation of semantic information is part of the *Semantic Web* initiative. DAML+OIL is proclaimed as a language to describe these semantics. The previously mentioned ontologies can be used to augment a linkbases semantics further. For example there could be the ontologies "FATHER" which is defined as the inverse of "DAUGHTER".

Query

On layer three the semantic relations are assumed to be evaluated. A query could say: Give me all links that associate the resource "Mike". The result would contain an arc with arcrole "SISTER" to "Monica". The degree of evaluation of course depends on the implementation of the linkbase server. One that doesn't do any interpretation on semantic information would also respond "correctly" on the query just mentioned even though it would generally deliver less arcs. From this point of view queries are not really different then the ones operating on layer two. The difference comes with more complex queries that also would need to be supported. For example give me all links that involve resource x, but restrict the calculation of links to less then two indirections. The set of supported statements is consequently different (and bigger) for queries on layer three then on layer two.

Linkbase server

Obviously, the linkbase server is required to have much more intelligence than one responding to layer two queries.

Client side

The interpretation of link semantics brings higher value for the client. He is presented with links that have never been explicitly defined. However the additional work to be done is the specification of filters to avoid overloading with links. Imagine a "full" semantic web. If no very intelligent filtering is done the number of presented links would be enormous.

Metadata

A good filtering can only be performed with metadata about the provided links in the linkbase. The metadata - in contrast to layer two - does not only consist of a list of role and arcrole information items, but would build a web of semantic metadata. The representation would be more complex.

Evaluation

Advantages: The value of linking information would be much higher.
Disadvantages: The main consequence of positioning linkbase access on this layer is that technologies are required that are not yet fully developed. The semantic web is in development and the directions can still change massively in the next few years.
Also is the complexity of queries greater again then it was for queries on the linking model.

4.2.6 Layer 4, Linking and Context

This last, most abstract, layer is only mentioned for completeness. It is no option to create a XLink based linkbase access protocol on this layer because to many open issues exist until now. In all the previously discussed linkbase access mechanisms the client application has transformed the interests of the user to semantic and other attributes. The client application was responsible to calculate the dynamic context. Another approach - early proposed by the OHS working group - is to make user interests explicit. If the representation of user interests was standardized then a linkbase access query would simply contain the user interest and the static context (in whatever terms this is described). The linkbase would then calculate relevant links and return them. This is definitely a great improvement for client software since it is freed of the task to interpret the dynamic context.
Note that the management of user interests doesn't necessarily need to be performed by a linkbase server. Recent approaches propose agent based infrastructures that allow linking in context[20].

Note again that the upper two layers are exemplary for higher abstraction layers. The calculation of relevant links, other operations on links and the management of additional information like the user context can take arbitrary complex dimensions. Additionally, as we have seen with the linking in context there are various ways to realize the infrastructure, whether it is based on agent technology, or the linkbase server performs the task, or other services are involved in performing the management of context. The decision on the infrastructure should not be influenced by the linkbase access mechanism.

4.2.7 Summary: Where to settle Linkbase Access

The main idea behind proposing a standardized linkbase access mechanism is to facilitate and speed up the development of a hypertext infrastructure on the web. A communication protocol

between web clients and linkbase servers on which everybody agrees will encourage the development of hypertext functionality. The quality of the proposal therefore depends highly on its potential to get the agreement of as many involved parties as possible.
To decide on which layer the linkbase access should take place we consider the following criterias:

1. *Fitting into the existing infrastructure.*
2. *Simplicity.*
3. *Extensibility.*
4. *Openness.*

Fitting into the existing infrastructure

First, linkbase access should harmonize with existing infrastructure components, and second it should minimize the requirements for additional infrastructure components. With infrastructure we refer to existing services, applications and standards. Figure 4.4 shows some involved infrastructure components. This list is not complete. All the involved parties cannot be estimated in advance. But the most important ones can be taken into account.
Linkbase access and storage is intentionally positioned above the resource access and storage infrastructure because a linkbase *is* a resource and should be handled as one. Fitting into the existing infrastructure means first of all evaluating which existing technologies can be used to support linkbase access. Linkbase access as we discuss it here is targeted on the world wide

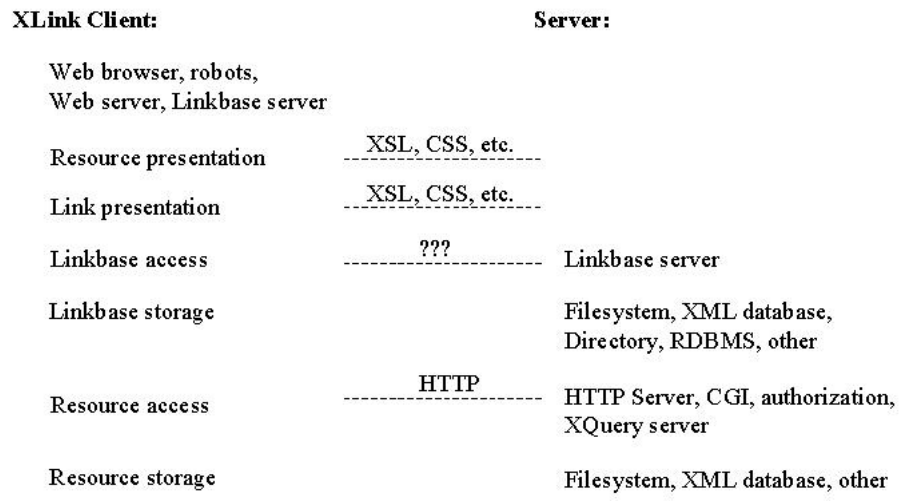

Figure 4.4: Fit to the existing infrastructure

web. The main and most important usage scenario is a web browser that displays web resources (HTML, XML). Involved is HTTP as the transport protocol and HTTP servers to service requests for resources. Other, non-webbased applications that use links are not considered important.
The decision to build the linkbase access on the W3C XLink standard was the first important step to fit the linkbase access into the web infrastructure. The linking model is fixed and an existing standard. The big question is still whether XLink will really be the technology to represent link information. More powerful and complex concepts like RDF could take this role in the future. We don't discuss this question here and assume that XLink is the linking

language of choice.

To come back to the discussion on which level to settle the linkbase access, Figure 4.4 does not show any higher-layer service or application that is yet widely used. Even though they exist (examples like semantic-web services), they are not widely used and no agreement has been made to use them. Linkbase access should not be positioned above layer two because of the lack of agreement on these higher-level services.

Simplicity

As already mentioned in the discussion of every layer, the queries are complex to compile on layer one, are less complex on layer two and get more difficult on the upper layers.
Simplicity also means that the access should be easily understandable for it's users. For an XLink client the most easy understandable linkbase access would be if he only needed to deliver user interests. However the understanding of user interests and their implications on the selection of relevant links is certainly not that easy. But this discussion is anyway dispensable here because the description of user interests is a field to be explored first.
It is not enough to define simplicity of linkbase access in terms of operations that need to be performed on the client side to calculate a query. The linkbase server functionality is more complex on a higher layer.
Transparency of the linkbase access can only be achieved if it does not presume more than the linking model. Consequently the choice falls onto layer two.

Extensibility

All higher level services have been discarded as too complex, intransparent and not yet matured in the above discussions. Still they are very important for the development of the hypertext infrastructure. It is therefore important that good extensibility is provided in order to facilitate these services. The higher we settle the linkbase access in the layers of abstraction the more we fix the possible services. Because of their diversity this is not appropriate.

Openness

Openness is a term that can have many different meanings. It is widely used and often not really defined. Here we mean that an open set of clients of the linkbase services is allowed (analogous to the definition by the OHSWG[18]). To bring in a quality measure for openness, this means that the set of potential clients should be maximized. This can be achieved by reducing the requirements on functionality that facilitates hyperlinking: Ideally the existing functionality of client applications should suffice to perform linkbase accesses. This cannot be expected to work, because besides the linkbase access, extended linking already has an influence on the way content is presented to the user.
On the server side the requirements on functionality can substantially be reduced by settling linkbase access on a lower level.

To summarize, the main aspect for a decision to settle linkbase access on layer two is the lack of standardized services with higher abstraction.

4.3 Linkbase Access Infrastructure

The infrastructure is a major concern in the design of a linkbase access service because it influences the acceptance of the service. Of concern are the smoothness of integration into the existing web infrastructure and the quality aspects that allow a long lasting, flexible and scalable linkbase framework. The maximization of reuse of existing technologies should be aspired and therefore this section discusses the participants in a linkbase access - the client and the server. To give an overview of the infrastructure it also discusses topologies, distribution of linkbases and related aspects.

4.3.1 Linkbase Topologies

We start from the discussion from a client view. Figure 4.5 shows two ways a client can profit from linkbase information. A) shows a linkbase un-aware client that accesses a content server. The content server extracts the links from the linkbase and enriches the documents with this information. The client doesn't handle linkbases and is only concerned with the presentation of the links. With XLink this setup doesn't exist alone. Because linkbase access is part of the linking language every client will need to be able to access linkbases.
This setup is implemented in many of today's content servers. They store the links in a separate store, however usually not in XLink. The difference is that they interpret the links and map them into HTML anchors because of the lacking support of XLink in today's user agents.
As soon as the client applications support XLink, case B) in Figure 4.5 becomes available. Note that the linkbase does not necessarily have any relation to the content server anymore. They can be maintained by different parties and the creator of a document doesn't need to be aware of the links that are added to his document. In the scenario in section 3.3 the user selected linkbases as a setting of his browser. It is likely that both approaches shown in Figure 4.5 will coexist. While the first approach requires no change on the client application (important for the migration phase to an XLink enabled web infrastructure), the second approach is more flexible for the user.

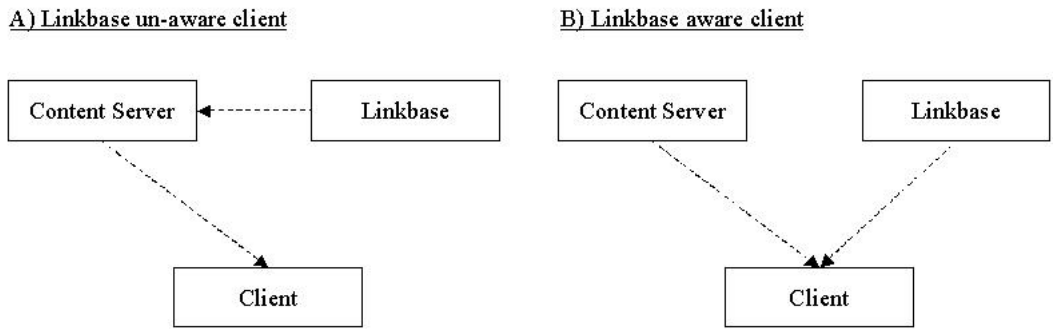

Figure 4.5: Linkbase Server, Content Server and Client

Distribution of XLink linkbases is an aspect to which not much attention has been paid in research yet (in contrast to OHS, where this always was an important issue). The protocols between linkbases and content servers to maintain the integrity of links, to add, delete and update links is out of the scope of this work - and with this also the tough questions of linkbase distribution. To mention is that XLink already contains mechanisms to access multiple linkbases because it handles a link to a linkbase as a regular arc. A linkbase can contain

traversal information to other linkbases - we call this chaining of linkbases. In Figure 4.6, picture A) shows this scenario. If a linkbase server interprets the links and traverses arcs with the special linkbase role, we have the setup as show in B).
In both approaches the server relies on its local knowledge to identify other linkbases.

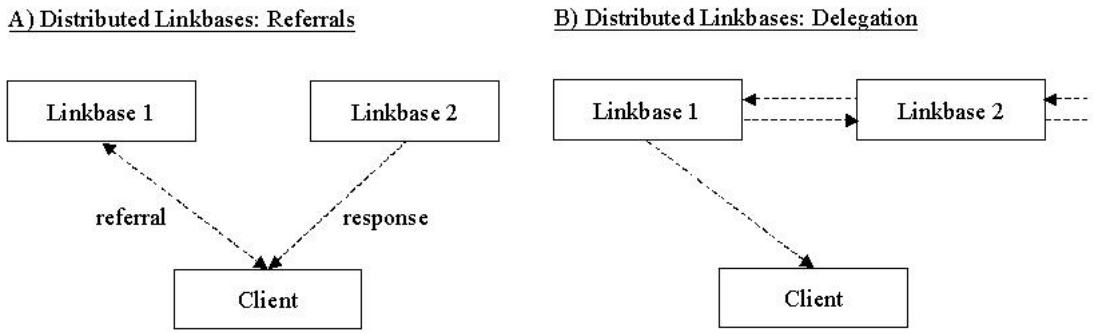

Figure 4.6: Distributed Linkbases

De Roure, Walker and Carr discuss further infrastructure concepts in [21]: Optimization of Linkbase distribution, Linkbases whose sole purpose is to carry knowledge about other linkbases, requests routing and relationships to directory services are subject of discussion.

4.3.2 Underlying Protocol

A protocol is a contract on how connected parties exchange information. As it can exemplarily be seen in the OSI Network Model, a protocol belongs to a layer in a stack of protocols. Figure 4.7 shows LBAP as a layer above the underlying protocol to which a binding needs to be defined. Defining protocols for the web is a critical design issue because it influences interoperability which

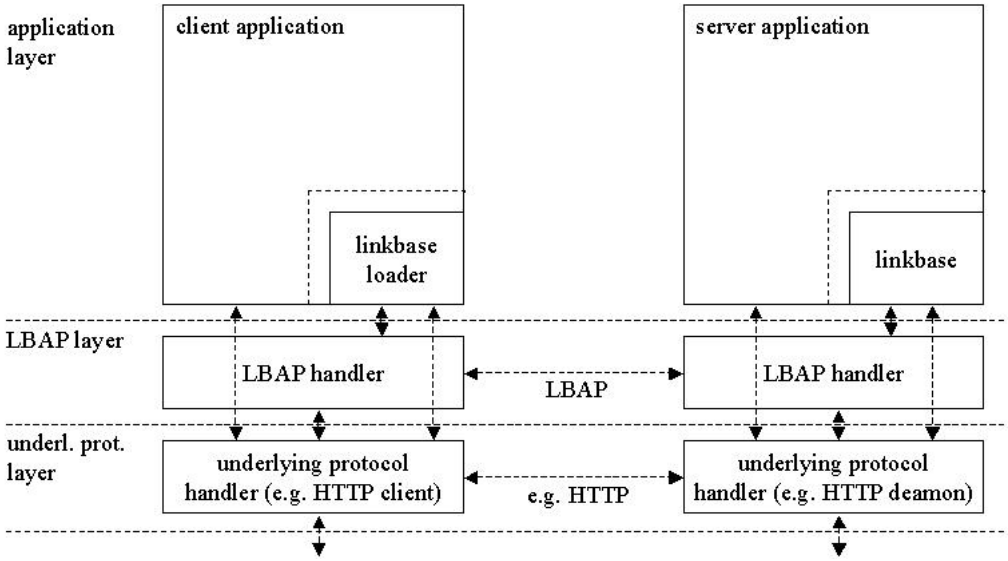

Figure 4.7: The Linkbase Access Protocol (LBAP) in the Protocol Stack

is one of the ultimate goals of the web. Therefore it is not surprising that there are activities that try to formalize protocol specifications. The W3C XML protocol working group[5] is specifying

[5]http://www.w3.org/2000/xp/Group/

SOAP which provides a standardized way to exchange structured and typed information between peers. SOAP also provides a framework to specify bindings to the underlying protocol. It would be possible to use SOAP to define the binding for a linkbase access protocol.
Figure 4.8 shows alternatives for underlying protocols that are all commonly used. Note that dotted lines indicate that this layer is optional (e.g. LBAP can use the HTTP entity without using SOAP).

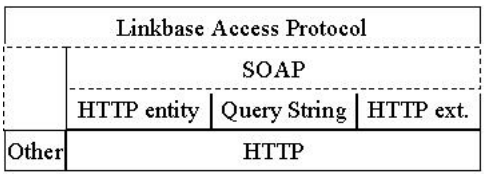

Figure 4.8: LBAP and Possible Underlying Protocols

HTTP is the lingua franca of the web and a founding block of its infrastructure. Our analysis of linkbase access showed that it also suits very well as an underlying protocol for LBAP. Request-response operation semantic is supported very well. Additionally, HTTP has proven features like authentication, referral, cache control (through extensions) which LBAP requires (the scenarios have shown the use of the authentication feature). Further more, HTTP is extensible through the HTTP extension framework. Let's have a look of some implications that the usage of HTTP as the underlying protocol would have and how it could be used.

A request can be encoded in a query string: The URI scheme (RFC2396[25]) allows you to include a query string that is to be passed to the designated URI reference. This is indicated by placing a question mark at the end of the URI reference, followed by the desired query string:

```
http://www.stillhard.net/service?query_string_data
```

The query string must be specially encoded, using what is known as *URI encoding*.

The query string is widely used in todays web to send small amounts to the server, as for instance parameters to a search engine. Another wide usage of query strings is to transport *HTML form* name-value pairs[7]. The processing of form data can be chosen by the user to result in submitting the data by either HTTP method GET or POST. In the former case a query string is constructed by (1) building pairs of control-names and values, separated by the = character, (2) encoding the query string as described in [25], and (3) appending the query string to the target URI reference, separating them with the ? character. An example for the resulting URI reference:

```
http://www.stillhard.net/service?name1=value1&name2=value2&name3=value3
```

Because the URI query string is widely used and can be processed with widely used server-extensions as CGI[23] this would be a good way to encode a linkbase request. The protocol handlers in linkbase servers could be implemented with well understood technology. However, query strings have two major disadvantages, namely that they are restricted in size and that the encoding of the values must comply to the URI encoding scheme. The length of the URI is not limited by HTTP, but practically lengths of more than 255 bytes should not be used. RFC2616[3] says:

> The HTTP protocol does not place any a priori limit on the length of a URI. Servers MUST be able to handle the URI reference of any resource they serve, and

> SHOULD be able to handle URIs of unbounded length if they provide GET-based forms that could generate such URIs. A server SHOULD return 414 (Request-URI Too Long) status if a URI reference is longer than the server can handle [...].
>
> Note: Servers ought to be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations might not properly support these lengths.

Because linkbase requests are not limited in their length and can easily exceed 255 bytes query strings are not the appropriate mechanism to transport LBAP requests. Additionally the mandatory escaping of characters should be omitted by using a mechanism that allows xml content types.

A LBAP request must consequently be transported in an HTTP entity with a POST method. The definition of LBAP shows how this can be achieved (in section 6.3).

4.3.3 XLink User-Agent - Issuing a Query

The processing of XLink type elements is not (yet) defined by a normative reference. The XLink recommendation contains processing related charters, but leaves a lot of space for interpretation to the application developers. This certainly is part of the reason for the slow adaption of the standard. XLink processing questions are also left untouched in most XLink related documents an exception being the *XML Linking and Style* Working Draft [13].
To develop the LBAP it is not necessary to make assumptions or proposal about how to process XLink links in general. The traversal of a linkbase arc is only a piece in the puzzle and can be separated. Linkbase arc traversal has a different semantic as indicated by the arcs special arcrole. However, the only difference is that it entails loading the ending resource rather than to present it to the user (and that the user must have control over the traversal by being able to suppress it, but this is not relevant here). Besides this, a linkbase arc traversal should have the same handling as a normal arc.

4.3.4 Linkbase Server - Querying a Linkbase

Request handling

The minimal configuration of a linkbase server handles a request for a resource referenced by a URI reference (to be precise by the *path* component of a URI reference). It locates and delivers an XML document that is stored in the filesystem in the response. This is the functionality a traditional HTTP- or web server provides. In other words, traversal of a linkbase arc in a document consists of a HTTP GET request to the server hosting the linkbase. The web server returns the XML document containing the XLinks in the HTTP response.
In a more sophisticated linkbase server configuration the linkbase resource is not a document but a serverside application that is accessed by the HTTP server on request for the linkbase. This conforms to the definition of a resource as discussed in IETF RFC 2396[25]: A resource is any addressable unit of information or *service*. In contrast to the preceding configuration this can be seen as an *active* linkbase server. Because it is active it can handle LBAP requests. The *query* component, according to RFC 2396 defining URI's, is interpreted by the resource - the linkbase. It can be used to parameterize the request and can contain instructions about what the linkbase server should return in the response. Assumed that HTTP is the underlying protocol

for LBAP, the HTTP POST method can also be used to transfer additional information to the linkbase server. That information is the data load of the LBAP.
The response from an active linkbase server must be an XML document. Before having a look at how this document can result from a filtering and calculation process let's have a closer look at why the response needs to be an XML Document.

Response

In the first place the response to a request for a linkbase should be an XML Document and not just an XML fragment for compatibility reasons. If an active linkbase server responds differently from a passive one, this imposes serious dependencies. Only clients fully supporting LBAP could communicate with the linkbase server. A second reason is simply that the response is well-formed XML. The use of the delivered linkbase document can not be predicted. A client might very well want to validate it. Note that the client of a linkbase can be a browser but also a noninteractive application, such as a robot. Third, the XLink recommendation states in section 5.1.5 a constraint: "Linkbases Must Be XML" and continues: "Any linkbase specified as the ending resource of an arc with this special value must be an XML document." A clear statement which is made a bit less clear by a sentence two paragraphs later: "[...] the application should access the linkbase and extract any extended link found inside it. In the case that the extracted resource is a portion of a complete XML document, such as a range or a string range, only those extended links completely contained in the extracted portion should be made available." The confusion results from the statement that the linkbase should be *accessed* and that the extracted resource can be a portion of a complete XML document. This could be interpreted that after a linkbase access just an XML fragment is returned. It is more likely that *extraction* means the processing of the fragment identifier in the URI reference that references the linkbase. In this case *access the linkbase* means receiving the linkbase XML document. Enough reasons to charter: The response to a linkbase access must be an XML Document.

Result generation

Four examples to get an impression on the various ways a linkbase server could come up with an XML document far a LBAP request:

1. From the query in the LBAP request an XSLT style sheet is generated. This can be a simple mapping from query components to conditions in XSLT templates (an XML data representation of the query components would certainly ease this task). The style sheet is then used to transform the original linkbase document to a document that only contains relevant links. This document is the response to the LBAP request.

2. The linkbase server can be an XQuery enabled application. The LBAP request is transformed into a String conforming to the XML Query Language. The result of the processing of this query is the response on the LBAP request. The question is allowed why transform the LBAP request into an XQuery expression. The answer is that it cannot be expected that every linkbase server is XQuery enabled and even more important: XQuery does not know the XLink information set and therefore is not suitable without extending it. This would collide with the simplicity requirement. The complexity of XQuery is not needed to filter a linkbase. This will be outlined in the next section. A reasonable solution however would be to subset XQuery and allow only the necessary constructs in an LBAP query. This would guarantee compatibility with XQuery enabled applications while still keeping the complexity as low as possible. Chapter 6 will discuss data representation in more detail.

3. The storage format of the linkbase is not necessarily an XML document. The links might be stored in a relational database or a set of files in non-XML format. Consider the case of an existing link repository in a RDBMS that was formerly used to enrich HTML documents with links. These links should now be provided in a linkbase. The linkbase server could construct the response document from the result of some SQL statements against the database.

4. In a more dynamic scenario the links are created for the request. If the linkbase server has knowledge about the content of the document that referred to the linkbase it can calculate relevant links for the document. The knowledge can be a set of keywords, an RDF resource or also the whole document. The calculated links are generic links if only a description of the document is available or full links if the whole document is available. An example for link calculation is a glossary linkbase that is not restricted to a known set of documents "using" the service.

The processing of the query can be arbitrarily complex depending on the number of features the linkbase offers. As already mentioned in the linkbase scenario *Comments from a friend* in chapter 3.4, links can require certain access rights to be retrieved. Or the LBAP request might result in loading the referring document to parse its content and then calculate links for it. This mechanism might be not efficient enough in most cases and therefore not a very good example. A similar situation which is much closer to reality is that the linkbase interprets references to further linkbases. An appropriate handling of chained linkbases is necessary to avoid endless processing if there are loops in the chain and to optionally limit the number of steps. The XLink recommendation mentions that "The application interpreting an initial linkbase arc may choose to limit the number of steps processed in the chain". If the linkbase server processes the chain this information needs to transferred in the LBAP request.

LBAP should not make any assumptions about how a query is evaluated and how much complexity and dynamics lie in the evaluation process. The next section formalizes the querying of a linkbase in order to allow its separation from the processing of the query.

4.4 Linkbase Processing

4.4.1 Query an XLink Document

The modelling of an XLink link might seem simple and straightforward at first sight. All that is provided are only six different element types which can be combined in a well defined way. Informally said: A link element contains resources and locators, and arcs connecting them. Additionally there are semantic attributes, but they don't influence the structure of a linkbase by definition. Querying a linkbase can seem straight forward at the first sight: The query arguments are mapped to conditions for selection of an XLink-type element. All selected elements build the response - the filtered linkbase. The problem - and the reason why querying is more complex - is the few constraints on XLink markup and the defaulting for missing attributes and elements. For instance, removing arcs and locators from a link can change its meaning; Consider an extended link with two locators. The first one acting as the starting resource, the second one as the ending resource. An arc in the link specifies traversal from the starting to the ending resource. Assume a query on this link that selects only arcs with a certain role. The arc in our example does not have this role. The result of the query would be the link containing both locators, but not the arc. According to XLink, if an arc is missing in a

link, then the missing *from* and *to* values are interpreted as standing for all the labels in that link. In other words, the link would define bidirectional traversal between the two resources. Removing an element from the link resulted in a different linking of the resources. This semantic relationships and dependencies are explored in this section.

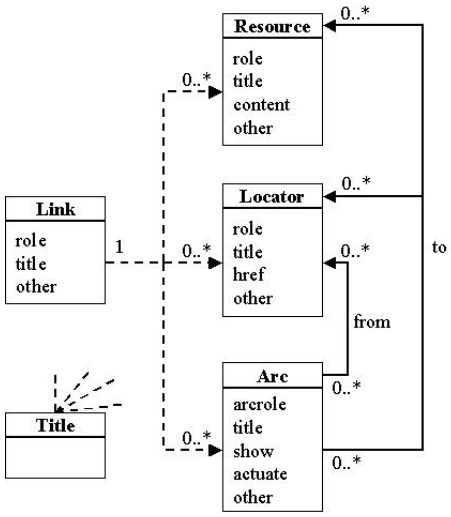

Figure 4.9: UML-like diagram for the XLink linking model

Figure 4.9 illustrates the XLink elements in the form of a UML-like diagram. Dotted lines indicate hierarchical, XML element/child dependency, full lines indicate relationships that are defined by the linking model. The diagram shows the significant child types of elements with XLink specific meaning. The elements can contain other, proprietary children which might be interpreted by XLink applications but don't have an XLink specific meaning.

Following a list of markup constraints and -defaulting in XLink as specified in the XLink specification. The list is not complete but considers all aspects relevant for subsetting a linkbase. In particular simple links are not taken into account.

Defaulting of from and to attributes

The two traversal attributes *from* and *to* are not mandatory for arc-type elements. If no value is supplied for either of them, the missing value is interpreted as standing for all the labels supplied on locator-type elements in that extended-type element. A missing traversal attribute is logically an abbreviation for multiple arcs, one for each different value of a label attribute. The consequence for query processing is that it is not sufficient to compare *label* attributes of locators and resources with *from* or *to* attributes, respectively.

Defaulting of missing arc-type elements

A missing arc-type element is equivalent to an arc not specifying the from and to traversal attributes. Note that this is only true if no arc-type element is provided by a link. To express that, there is no traversal between two resources arc traversal rules need to leave out this pair. The important part is that there need to be traversal rules to express that there is no traversal between two resources. As mentioned, if there are no arc traversal rules this defaults to traversal from every resource to every other. XLink does not provide a way to exclude any traversal. This

is acceptable since the purpose of XLink is to describe traversal rather than solely semantic connection of resources. This field is left to other mechanisms (like the Resource Description Framework (RDF)[6], Dublin Core[26], Topic Maps[27, 28], etc.).
The first paragraph in this section showed that it is not appropriate to simply strip an arc from an extended link because it can result in defining additional traversal instead of reducing it.

Multiple title-type elements

Title type-elements can occur in extended-, locator- and arc-type elements. Appropriate treatment of multiple title-type elements is left to higher-level applications built on XLink. Filtering out of title-type elements is not problematic with respect to the semantic meaning of a link. It would have been if they had a relationship to the title attribute which is explicitly denied in the XLink specification. A query can contain a condition and only matching titles are returned.

Extended-type element with fewer than two resources

The consequence is solely that the link is not traversable. Still it may be useful, for example, to associate properties with a single resource by means of XLink attributes. See section 4. of the XLink specification for examples of application of a link with only one resource.

Non-XLink related markup in XLink elements

All XLink elements, that means elements with a *type* attribute from the XLink namespace, can contain other content and markup. This is a consequence from the fact that the XLink's namespace only provides global attributes rather than elements. The content does not have XLink specific meaning but could very well be interpreted by XLink enabled applications. An LBAP query that would generally strip all content away that is not related to XLink is not appropriate. For example it could become a de-facto standard that extended-type elements contain a copyright element to state the owner of legal rights for this link. User-agents would interpreted this element and display a copyright notice to the user. Such content should be preserved when subsetting a linkbase. However a requirement for an LBAP query is that it must be able to optionally filter non-XLink markup in order to reduce the size of the response. This filter should also provide options to restrict the delivery of non-XLink markup to certain namespaces.

Non-XLink related attributes in XLink elements

For attributes that don't have XLink specific meaning the same is true as for non-XLink related text nodes and elements. Stripping such attributes would restrict the future extension of XLink linking elements. As mentioned in the previous entry, it should be possible to filter for attributes that are from a certain namespace.

Multiple arcs with behavior show="replace" actuate="onLoad" for a single resource

XLink does not constrain the application behavior for such arcs. An XLink application should have a strategy for choosing which arc to traverse. All other arcs do not have any relevance anymore because they won't ever be traversed (at least with the same application). A query mechanism could hand the selection of the favored arc to the user or to the document author referencing the linkbase, respectively.

The discussed markup constraints and -defaulting in XLink are very reasonable if you consider the flexibility they give XLink and the reduction of the size of an XLink linkbase through defaulting. The transformation from the linking model to the XLink markup however is complicated. As discussed in section 4.2 this is the reason why linkbase access is settled on the linking model layer, operating on the XLink Infoset rather than on XLink markup.

4.4.2 Querying the XLink Infoset

Before proceeding to a definition of the linkbase protocol service primitives we discuss how a linkbase can be queried on layer two. We investigate what queries should be possible on the linking model. We will use the term XLink Infoset as an equivalent for the linking model, even though it is just one possible description of the linking model. Refer to section 2.5 for an informal description of the XLink Infoset.

For a short repetition, a *linkbase* consists of *links* which states a relationship between *link participants* and defines traversal between *starting participants* and *ending participants* in the form of *arcs*. All these linking entities have properties, manifested in attributes in the linking language.
The goal of a query on a linkbase is to select the relevant linking entities. It needs to handle three aspects:

1. Select relevant entities. The relevance of a linking entity is determined by its properties and also by the relevance of its sub entities. Properties are for example the role of a resource, the title of a link or the arcrole of an arc. The sub entities are modelled in the hierarchic structure of the Infoset. A linkbase consists of links, a link contains arcs, an arc contains two participants and all of them can contain title elements and any other element. A link for instance is only relevant if it contains relevant sub entities. If the query expresses that no link participant is of relevance, then the link isn't either.

2. Reduce the size of the result. Even though the reduction of the size of the result is a (notably very welcome) side-effect of the filtering for relevant entities, it might not be enough. If the number of relevant entities is huge for a query and it cannot be further reduced with a more detailed query, then a query should also facilitate the limitation of the number of entities in the result. Imagine a linkbase with links that associate news items with a stock quote. There might be hundreds of such links for one quote and they might all be relevant (because the query only says that relevant links originate from a particular resource identifying the stock in question). Of course it is a question of the design of the linkbase whether it allows to reduce the number of *relevant* entities to a reasonable number. If the design is not the best, the client needs to have a possibility to specify a limit on the number of relevant entities. It must be possible to specify a size limit on all potential entities of a query result.

3. Specify the entities of interest. Depending on the purpose of the linkbase client application, different linking entities are of interest. In the case of hypertext presentation to a user as it is done by a browser the entities of interest are arcs and their starting and ending participant. The presentation engines are interested in the styling of traversal and the resources that participate in the traversal. Other applications - various kinds are imaginable, like robots analyzing the relationship of resources - are to a lesser extent interested in traversal but have their focus on all resources that participate in a link. Consequently in a linkbase

access it needs to be possible to return resources that don't participate in an arc - the link participants. It must be possible to specify whether arcs should be returned, and if they should, whether the link participants that are not referenced by an arc should be contained in the query result.
Another aspect of specifying the entities of interest is non XLink information contained in the linkbase. The XLink syntax allows to have arbitrary sub elements of the linking elements. They have no XLink specific meaning, but might well be of interest for a client interpreting the links. It should be possible in a query to specify whether such information is of interest and should be returned. The query should also be able to filter this information based on its namespace. This is important since the amount of non XLink specific data can be very big and should not be delivered in a query result if the client does not handle it anyway.

From this discussion of a query on a linkbase it becomes apparent that the queries can be complex. The design of the query should keep this complexity low for simple requests, but allow complex queries to precisely specify what should be returned. Formally, the query operates on the Infoset and it must be possible to specify queries that take every Information in the Infoset into account. The queries should however be simple if only a few properties of the Infoset are involved. In other words: The queries should scale with their complexity.

4.5 Summary

One of the goals of this chapter was to separate linkbase access from related tasks. We have made the following separations:

- Concepts with a higher level of abstraction like semantic conclusions, ontologies, user interest or other ways to describe context should not be involved in the linkbase access protocol. But we have shown that a mapping of the more abstract concepts to the linkbase access protocol can be defined. Additionally, we have motivated that an LBAP should not be defined in terms of XLink markup, but rather in terms of the linking model. The processing of the markup is left to the parties participating in a linkbase access. Because of this separation we also accomplish that the storage format of a linkbase is not forced to be XLink.

- The presentation and processing of links does not need to be taken into account in the design of LBAP. Because the interface to the styling and processing software is defined to be XLink markup we can decouple this issues from the protocol.

- An LBAP request itself can be separated into a query part and a processing properties part. The query part is used to filter the linkbase by selecting relevant entities. Its definition is bound to the semantics of the XLink infoset. The processing properties control the filtering and define the behavior of the linkbase server. This part cannot be precisely defined and as we will see is based on various considerations. Because of the fuzzyness of this part it should be possible to extend it in the future.

- A clear separation to the underlying protocol for LBAP was made. There are no compelling facts that restrict the choice of the underlying protocol to a certain one. The requirements on the protocol that originate from distribution of linkbases or different approaches to access linkbases are low. Consequently, the design decisions about data representation, data encapsulation, etc. can base on an evaluation of the highest benefits that a protocol

can bring. We have also seen that with linkbase access many issues need to be resolved by the underlying protocol as for instance the authorization of an access. These issues are separated from LBAP.

Chapter 5

Requirements for a Linkbase Access Protocol (LBAP)

The preceding chapters introduced the concept of linkbases and their implementation with XLink, and the scenarios showed possible applications of this concept. They also motivated the need for a linkbase access protocol, discovered how it can help to facilitate a linkbase web infrastructure, what functionality is required to support and places where it comes into action. This chapter summarizes the findings and converts them into requirements. They will be used as a guideline to develop and specify LBAP.

5.1 Requirements

In order to make the list more easily understandable, we have chosen to adopt the terminology from RFC 2119[22], which is commonly used to indicate requirement levels.

- **Supported link model**
  The protocol MUST provide full support for the linking model of XLink. Servers MAY choose to use other linking models internally, but if they do so MUST provide a mapping to the linking model of XLink.

- **Access modes**
  The protocol MUST NOT allow write access to linkbases. Even though write access is desirable for many interesting application scenarios (such as annotation systems), the requirements of managing write accesses to linkbases are very complex and out of scope of our work. The protocol therefore MUST only allow read access to linkbases.

- **Adaptable to the existing Web infrastructure**
  The protocol SHOULD NOT require any changes in related standards. Existing technologies SHOULD be used as far as possible. However, because some of the required foundations are not yet in place (in particular, the XML Infoset), the necessary standards SHOULD be pushed and developed in parallel to the protocol activities.

- **Simplicity**
  To ensure quick adoption and implementation, easy understanding and maintainability of the protocol, the protocol SHOULD be as simple as possible, and the specification SHOULD be concise and easy to understand.

- **Extensibility**
  The protocol SHOULD be designed to be open for evolutionary steps in order to allow evolution of the implementing parties. In particular, even though the link model supported by the protocol itself is the XLink link model, it MUST be possible for implementations to exchange information about link model extensions, and to exchange information about the supported extensions. However, in order to avoid non-interoperable implementations, all implementations MUST support the XLink link model.

- **Openness**
  The protocol MUST be able to cover future extensions to XLink, without invalidating all existing implementations.

- **Related standards**
  Related standards SHOULD be considered and MAY be used as parts of the protocol if they satisfy all requirements. In particular, their stability, simplicity, and functionality SHOULD be appropriately developed. The protocol SHOULD not define yet another query mechanism if the existing ones cover the requirements.

- **Configuration language**
  User configurations for accessing linkbases include predefined queries or query parameters that are maintained by the client. These configurations can become complex, especially if they cover many different linkbases. There MUST be a language to submit configurations from a server to a client (eg, to communicate a configuration which has been determined via a Web-based user dialog), and from a client to a client (eg, if a user switches clients and wants to use the same configuration for linkbase access with both clients).

- **Generic links**
  XLink's data model does not explicitly include generic links. However, the protocol SHOULD include mechanisms so that generic linking applications are supported, for example by clients submitting content to the server, which then uses this content to compute the required links.

- **Access to every attribute in the XLink namespace**
  XLink's data model heavily depends on the XML attributes defined in the specification. The protocol MUST provide access to all these attributes and enable clients to request links based on filtering these values. However, the server MAY choose how to provide access to these attributes, or refuse to serve certain requests (eg, requests which would result in the whole linkbase being returned as a result).

- **Access to attributes of linking elements that are from a non-XLink namespace**
  XLink is open for extension. For example, XLink elements may have attributes with linking semantics that are not contained in the XLink namespace. Additionally, in future XLink versions, the set of global attributes in the XLink namespace might be extended. Therefore, it MUST be possible to specify conditions for non XLink-namespace attributes as well. However, the server MAY choose how to provide access to these attributes, or refuse to serve certain requests (eg, requests which would result in the whole linkbase being returned as a result).

- **Resources that are referenced in semantic attributes**
  XLink supports application linking semantics by defining attributes which must contain URI references to resources which then define the semantics. The format and interpretation of the resources is not part of the XLink specification. The linkbase access protocol MUST

provide access to these linking semantics by supporting some kind of container format for conditions for the content of XLink's semantic attributes. Implementations are free to support any number of these additional semantics (including none). The protocol SHOULD NOT make any assumptions about the format of these semantics resources.

- **Restrict the size of the returned linkbase**
  Linkbases can be very big. Thus, accessing a linkbase may result in very many links being returned, and in order to avoid unnecessary exchange of data, clients MUST be able to specify the maximum size of the returned linkbase in terms of number of links and/or number of arcs and/or number of locators. Additionally, servers MAY use internal limits and refuse to serve requests which result in more links/arcs/locators than permitted.

- **Delivery of local resources**
  XLink supports local resources, which are resources contained in the link itself. These resources may be of any size, so clients may want to exclude them. Consequently, clients MUST be able to suppress the delivery of local resources, in which case the server MUST strip the resources from the links before delivering them. The server MUST make sure that after stripping the local resources the links still remain valid by removing arcs that are pointing to stripped local resources.

- **Chained linkbases**
  XLink allows chained linkbases by using links within linkbases that point to linkbases. Servers MAY follow links to linkbases themselves (thus acting as clients in another linkbase access, creating a chain of linkbase accesses), or they MAY choose to not make chained requests and return the linkbase link to the client. Consequently, clients MUST be able to handle links to linkbases being returned from servers, and they MAY handle them as appropriate for the application (ie, either ignore them or use them for further linkbase accesses).

- **Processing steps to integrate query results must be provided**
  Linkbase accesses can be initiated because of different reasons, such as processing a resource referencing a linkbase, accessing a linkbase because of client configuration, or accessing a linkbase on user request. The protocol MUST define processing steps how to integrate the results of multiple queries.

- **Processing model**
  In general, the protocol MUST specify a processing model which describes how to process the data being exchanged. In particular, the connection between the linkbase schema (as represented by the attribute values of semantic attributes) and linkbase content must be clearly defined.

- **Usage scenarios**
  In order to make adoption and implementation of the protocol more easy, usage scenarios (including sample messages being exchanged) MUST be provided. This part probably is going to be informative only, but can nevertheless be crucial for the success of the protocol.

This list of requirements and design principles is rather long, and there are many possible designs to satisfy them. The next chapters present a specification of a protocol by defining services for accessing linkbases. At various places in the further reading we reference the list of design principles and requirements again to ensure that they are considered appropriately.

Chapter 6

LBAP Definition

From the findings in the previous chapters about the linking model, possible linkbase scenarios, requirements on linkbase access and the concepts on linkbase access we developed a proposal for a Linkbase Access Protocol (LBAP).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119[22].

6.1 Structure

The core elements of LBAP:

- *Operations*
  LBAP defines one operation (also known as protocol primitive). The *getLinkbase* operation is used to retrieve an XLink linkbase. It has request-response transmission semantic. The request is parameterized with all basic information that is necessary to process a linkbase. The response contains the XLink linkbase and status codes and optionally error messages. Further operations to interact with a linkbase can be added to LBAP in the future.

- *Datatypes*
  For the *getLinkbase* operation two datatypes are defined: *linkbaseRequest* and *linkbaseResponse*. The request contains a filter and processing properties that control the filtering. LBAP defines a common set of processing properties that cover the processing of the XLink linking model. It is extensible to hold additional processing properties that are proprietary for a linkbase service. The response carries the XLink linkbase and status information.

- *Protocol Binding*
  LBAP defines a binding to HTTP 1.1. This binding is mandatory for all LBAP implementations and is designed to harmonize with the existing web infrastructure. Additional bindings can be defined to increase the accessability of a linkbase.

- *Processing Model*
  LBAP defines the processing of a request in terms of how to query the XLink linking model based on the filter and processing properties as specified in a request.

The processing of an LBAP request - we don't speak of a query to express the extraction semantics of a *getLinkbase* operation - operates on the linking model as it is specified by XLink. Other hypermedia concepts like ontologies or the dynamic context of a user, etc. are

not supported. However, because of the high diversity of possible linkbase functionality LBAP defines a common denominator for linkbase access and specifies extension mechanisms. The common functionality as defined here MUST be supported whereas all extensions MUST be optional. The extension mechanism targets the possibility to increase the quality of the result from a linkbase access by adding specific information to a request, and not the formation of categories of linkbase services which all require specific request structures. With this restriction it will be guaranteed that all linkbase services can be accessed with a common linkbase client.

LBAP can optionally be extended in two orthogonal directions. First, additional operations MAY be defined. Examples are operations to retrieve the linkbase schema or linkbase management operations to update linkbases. These operations are fully independent from the *getLinkbase* operation. Second, the getLinkbase request datatype MAY be extended to facilitate more sophisticated linkbase processing. An example would be an extension to support a generic-link service. The request would be extended with the source-document itself or descriptions of the source document.
To illustrate the extensibility, the next chapter describes a Linkbase service that supports extensions together with LBAP.

The datatypes that LBAP uses are defined as XML Schema types which describe the involved data structures.

The following sections define the core elements of LBAP.

6.2 Linkbase Request and Response

6.2.1 Request

The linkbase request is issued to retrieve a queried linkbase. It's designed such that it is possible to perform comprehensive filtering on the linkbase infoset. Four concepts can be found in a request:

1. *Request processing properties*
   A request has global properties which provide general information about the processing of a request. They allow to specify the *entities of interest* (also referred to as *relevent entities*). E.g. that no traversal information (arcs) should be contained in the result or that no local resources should be returned.

2. *Size Limit*
   This is a special request processing property that allows to put a limit on the number of linking entities that are returned.

3. *Filter*
   For every entity in the infoset a filter is defined. The Filter model is a hierarchical structure that is consistent with the one of the linking model. A filter is defined for every linking construct and consists of subfilters for linking constructs that follow in the hierarchy. For instance there is a LinkFilter which consists of Arcfilters. The ArcFilter contains filters for the resources participating in the arc. The result of a filter is a set of corresponding linking constructs.

4. *Property assertions*
   They are used to describe conditions for infoset entities. In terms of XLink they specify

conditions on the attributes of a linking element. Their return value MUST be boolean. The filtering by the means of selecting relevant information is achieved through assertions on the properties of a linking element (e.g. the role of a link or the title of an arc). Consequently the result of a *filter* MUST be empty if the assertions are not fulfilled. If they are, it MUST contain the results from its subfilters.

The response to a request MUST contain the relevant entities together with a result code with service specific information like success/failure, etc.

Following the linkbase request datatype:

```
<xsd:complexType name="LinkbaseRequestType">
    <xsd:sequence>
        <xsd:element ref="lbap:ProcessingProperties" minOccurs="0"/>
        <xsd:element ref="lbap:SizeLimit" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="lbap:ExtensionDelivery" minOccurs="0"/>
        <xsd:element ref="lbap:LinkFilter" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
            <xsd:annotation>
                <xsd:documentation>
                    This element makes the extension of a request possible
                </xsd:documentation>
            </xsd:annotation>
        </xsd:any>
    </xsd:sequence>
</xsd:complexType>
```

The request specifies which linking elements should be returned (*ExtensionDelivery, ProcessingProperties*), the size of the response (*SizeLimit*), processing specifications (*ProcessingProperties*) and a filter to specify conditions for relevant links (*LinkFilter*).

All parts of a request are optional. On an empty request, the whole linkbase SHALL be returned.
A linkbase MUST support processing based on the defined parts (*ProcessingProperties, SizeLimit, EntensionDelivery and LinkFilter*) of a request.
The request MAY contain additional markup from other namespaces containing proprietary information to increase the precision of the linkbase processing. The additional markup MUST be optional and a linkbase MUST NOT depend on the information to result in a successful response.

ProcessingProperties

These properties specify the processing in a global scope.

```
<xsd:complexType name="ProcessingPropertiesType">
    <xsd:attribute name="timeLimit" type="xsd:int" use="optional" default="0"/>
    <xsd:attribute name="noLocalResources" type="xsd:boolean" use="optional" default="false"/>
    <xsd:attribute name="noTraversal" type="xsd:boolean" use="optional" default="false"/>
    <xsd:attribute name="lbChainLength" type="xsd:int" use="optional" default="0"/>
    <xsd:attribute name="deliverIfSizeLimitExceeded" type="xsd:boolean" use="optional" default="no"/>
</xsd:complexType>
<xsd:element name="ProcessingProperties" type="lbap:ProcessingPropertiesType"/>
```

- timeLimit
  The time limit sets a timeout on the processing of the request. If no result could be calculated in time then the processing MUST be aborted and the failure reported with a corresponding error in the response. A value of 0 indicates no time limit.

- noLocalResources
  A linkbase contains inbound and third-party links. For inbound links the ending resource is local. It is desirable to retrieve only third-party links if the links are not supposed to be presented but rather used for calculation or other interpretation. When *noLocalResources* has a value of *true*, local resources MUST NOT be delivered.

- noTraversal
  When this attribute is set to true, then arcs MUST NOT be delivered in the response. This is required if the requesting application is only interested in the information that XLink adds to a resource; for instance the semantic information.

- lbChainLength
  A linkbase server MAY follow linkbase arcs, even though this does not by definition lie in its responsibility - it only needs to interpret the linking model and not the semantics. This task is conceptually left to the client. But to make the filtering mechanism effective (for example in terms of the number of returned links) and to take responsibility from the client it makes sense that the server can interpret chained linkbases. This feature actually lies in a higher layer, but is included here because of its high value. The *lbChainLength* attribute is OPTIONAL for this reason. If it is not specified then linkbase arcs MUST NOT be traversed.
  When processing chains of linkbases it is necessary to restrict the number of indirections to avoid endless loops if the linkbases have circular references. The *lbChainLength* attribute allows to specify the number of indirections. The number of indirections MUST NOT exceed the value specified in *lbChainLength*. If it is set to zero linkbase arcs MUST NOT be traversed.

- deliverIfSizeLimitExceeded
  A limit on the size of the request result MAY be specified. *deliverIfSizeLimitExceeded* allows to specify whether a result should be delivered if the size limit was hit. A value of *true* indicates that a linkbase MUST return a linkbase in its response. The content of the resulting linkbase is however not defined. It is RECOMMENDED that it does not contain more entities than specified in the size limit. A value of *false* indicates that an appropriate error MUST be returned in the response if the size limit is hit. This property is OPTIONAL. If it is not specified, it defaults to a value of *false*.

SizeLimit

Optional part of the request that indicates the size of the response in terms of number of links, resources and/or arcs. If no limit on the size is specified, all relevant entities SHALL be returned. The size limit is defined as follows:

```
<xsd:complexType name="SizeLimitType">
    <xsd:attribute name="entity" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="links"/>
                <xsd:enumeration value="arcsPerLink"/>
                <xsd:enumeration value="linkParticipantsPerLink"/>
                <xsd:enumeration value="fanout"/>
                <xsd:enumeration value="fanin"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="limit" type="xsd:int" use="required"/>
</xsd:complexType>
<xsd:element name="SizeLimit" type="lbap:SizeLimitType"/>
```

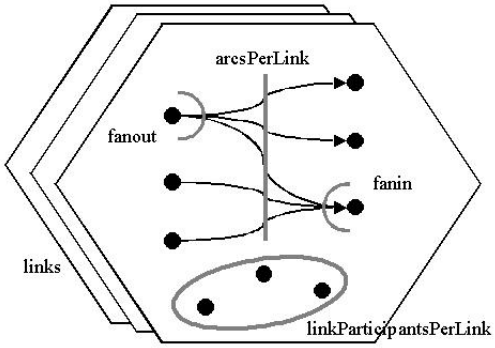

Figure 6.1: The five possible restrictions of a linkbases size

Figure 6.1 illustrates the limits that can be used in a linkbase request. Worth mentioning are fanin and fanout. With fanout a request can be forced to contain no more traversal items per starting resource than explicitly specified. With fanin the number of incoming links can be restricted.

If a size limit is exceeded and the processing property *deliverIfSizeLimitExceeded* is enabled, a linkbase MAY provide proprietary solutions to deliver a subset of the relevant entities.

Editorial note:

> Note that with the size limit on a request a problem arises: The linkbase server will process a linkbase in a certain order. If it only returns the first ten links for instance, then the rest of the links can only be accessed by weakening the size limit and requesting 20 links. This is not in the sense of a linkbase access protocol because it not only violates its goal to deliver relevant information only, but also makes the response to a request depend on the implementation of the server; on the order in which it handles the link information items.
> We see the reaching of the size limit as an error and therefore don't deliver the result. However there is a possibility to circumvent this rule by setting the *deliverIfSizeLimitExceeded* attribute to "yes". Which subset of the relevant entities is returned is implementation specific for every linkbase. Note however that the default for this attribute is "no" and the idea behind the inclusion of this feature into LBAP is its high convenience potential.
> Linkbases are free to provide other solutions to deliver a subset of the relevant entities. LBAP extensions can be used to specify the behavior. One possible solution would be to provide a way to get other results then just the first found ones. Similar to search engines it could be possible to retrieve a set of links with a limited size and then get the next bunch. This could be solved by providing a *start* attribute for every *sizeLimit*. It could also be solved by returning references for future searches in the result. The complexity of such a solution explodes, if the result of a limited set of the linkbase should be precisely defined. An order in the linking model would need to be defined which is not reasonable because a link only *states* relationship on resources and does not weight these relationships.

ExtensionDelivery

This part of the request allows to restrict the delivery of proprietary extensions of a linkbase, such as elements that don't have a type attribute out of the XLink namespace or attributes

from a "foreign" namespace. The ExtensionDeliveryType supports four modes, *xLinkOnly*, *attributesOnly*, *elementsOnly* and *all*. Optionally, namespaces can be specified to restrict the proprietary extensions. If none is specified all namespaces MUST be allowed for extensions.

```
<xsd:complexType name="ExtensionDeliveryType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="lbap:IncludeNamespace"/>
    </xsd:sequence>
    <xsd:attribute name="mode" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="xLinkOnly"/>
                <xsd:enumeration value="attributesOnly"/>
                <xsd:enumeration value="elementsOnly"/>
                <xsd:enumeration value="all"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="ExtensionDelivery" type="lbap:ExtensionDeliveryType"/>
```

LinkFilter

The LinkFilter is used to specify the relevant linking information. The result of a LinkFilter MUST be a set of links. Multiple LinkFilters MAY be specified in a request. The result MUST consist of the union of the return values of all the LinkFilters.

```
<xsd:complexType name="LinkFilterType">
    <xsd:sequence>
        <xsd:element ref="lbap:linkPropertyAssertion" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="lbap:linkParticipantFilter"/>
            <xsd:element ref="lbap:arcFilter"/>
            <xsd:element ref="lbap:titleElementFilter"/>
            <xsd:element ref="lbap:otherElementFilter"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="LinkFilter" type="lbap:LinkFilterType"/>
```

The LinkFilter colloquially spoken says: This link is relevant if the result of the linkAttrAssertion is *true* and contains subelements that are returned by the specified subFilters. The set of sub filters is bound to the child elements that a link can have. Multiple sub filters of one type can be specified. Their results are unioned.

The LinkFilter MUST be applied to every link in the linkbase and SHOULD be processed as follows:

1. Evaluate the linkPropertyAssertion. If it returns *false* then do not add this link to the result set and proceed with the next link. If it returns *true* or no linkPropertyAssertion is specified, continue

2. Evaluate the subfilters (linkParticipantFilter, arcFilter, titleElementFilter and otherElementFilter)

3. Union the resulting sets of the subfilters of the same type and add the sets to this link

4. If all result sets of all the subfilters are empty sets, then do not add this link to the result set and proceed with the next link

5. Add this link to the result set

LinkPropertyAssertion

A link hast two XLink specific properties (manifested as attributes in the XLink syntax), title and role, and can have other, proprietary properties. The link property assertion MAY provide assertions for both of this properties. An assertion MUST return a boolean value. Multiple assertions can be combined logically:

```
<xsd:complexType name="LinkPropertyAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:andLPA"/>
        <xsd:element ref="lbap:orLPA"/>
        <xsd:element ref="lbap:notLPA"/>
        <xsd:element ref="lbap:titleAssertion"/>
        <xsd:element ref="lbap:roleAssertion"/>
        <xsd:element ref="lbap:otherPropertyAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="linkPropertyAssertion" type="lbap:LinkPropertyAssertionType"/>

<xsd:complexType name="SetOfLPAType">
    <xsd:sequence maxOccurs="2">
        <xsd:element ref="lbap:linkPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AndLPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfLPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="andLPA" type="lbap:AndLPAType"/>

<xsd:complexType name="OrLPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfLPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="orLPA" type="lbap:OrLPAType"/>

<xsd:complexType name="NotLPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:linkPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="notLPA" type="lbap:NotLPAType"/>
```

ArcFilter

The *ArcFilter* is structured the same as the *LinkFilter* and follows the same processing steps.

```
<xsd:complexType name="ArcFilterType">
    <xsd:sequence>
        <xsd:element ref="lbap:arcPropertyAssertion" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="lbap:startingParticipantFilter"/>
            <xsd:element ref="lbap:endingParticipantFilter"/>
            <xsd:element ref="lbap:titleElementFilter"/>
            <xsd:element ref="lbap:otherElementFilter"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="arcFilter" type="lbap:ArcFilterType"/>
```

The property assertion and also the remaining filters follow the same pattern as we have seen for the link filter and the arc filter. Refer to appendix A.3 for a full reference of the service schema.

ValueAssertion

The base type for all of the assertions on the properties of linking elements is the value assertion. Its design allows many variations. A value assertion can consist of an equality match, that means it MUST evaluate to true when the property has exactly the value. More flexibility is added to the value assertion by allowing to define substring conditions. They follow the lead of XPath and allow to specify a substring that needs to exist in the value. Additionally, for the substring one can specify whether it needs to occur at the beginning or at the end or just somewhere in the value:

```
<xsd:complexType name="ValueAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:equalityMatch"/>
        <xsd:element ref="lbap:substringAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="valueAssertion" type="lbap:ValueAssertionType"/>
<xsd:element name="titleAssertion" type="lbap:ValueAssertionType"/>

<xsd:complexType name="SubstringAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:startsWith"/>
        <xsd:element ref="lbap:contains"/>
        <xsd:element ref="lbap:endsWith"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="substringAssertion" type="lbap:SubstringAssertionType"/>

<xsd:simpleType name="equalityMatchType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="equalityMatch" type="lbap:equalityMatchType"/>

<xsd:simpleType name="startsWithType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="startsWith" type="lbap:startsWithType"/>

<xsd:simpleType name="containsType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="contains" type="lbap:containsType"/>

<xsd:simpleType name="endsWithType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="endsWith" type="lbap:endsWithType"/>
```

Value assertions MUST be case sensitive.

Editorial note:

As mentioned, the value assertion can easily be mapped to an XPath expression. One could come up with the idea of defining the assertion in terms of a profile or subset of XPath, XPointer or XQuery expressions. This would be possible but is not appropriate because the assertions are not necessarily evaluated by XML processing software. It is a requirement on the linkbase access protocol that it does not force the linkbase server to represent and process the linkbase in XML. If the linking information is stored in a relational database then the design of the value assertions would much better if it defined them in terms of a profile on SQL WHERE clauses. Therefore we choose a common subset of operations that should be supported.

OtherPropertyAssertion

The value assertion is the super type for the URI assertion and is also used in the OtherPropertyAssertion:

```
<xsd:complexType name="OtherPropertyAssertionType">
    <xsd:sequence>
        <xsd:element ref="lbap:valueAssertion"/>
    </xsd:sequence>
    <xsd:attribute name="attributeName" type="xsd:NMTOKEN" use="required"/>
    <xsd:attribute name="namespace" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<xsd:element name="otherPropertyAssertion" type="lbap:OtherPropertyAssertionType"/>
```

Editorial note:

> The URI assertion, used on role and href properties, does currently not contain the constraints on URIs. The design of queries on URIs is out of scope of this work and we are not aware of much work done in this field. These queries can become quite complex, since their structure would be different for various schemas (e.g. http:, file:, etc.).

OtherElementFilter

The form of this Filter completely depends on the implementation of the linkbase server. Since the type of the "other" elements is not known, the filter can be defined as appropriate:

```
<xsd:complexType name="OtherElementFilterType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="otherElementFilter" type="lbap:OtherElementFilterType"/>
<xsd:element name="titleElementFilter" type="lbap:OtherElementFilterType"/>
```

6.2.2 Response

The response on a linkbase request contains information about the execution of the request and the resulting linkbase:

```
<xsd:complexType name="LinkbaseResponseType">
    <xsd:sequence>
        <xsd:element ref="lbap:error" minOccurs="0"/>
        <xsd:element ref="lbap:linkBase"/>
    </xsd:sequence>
    <xsd:attribute name="success" type="xsd:boolean" use="required" default="true"/>
</xsd:complexType>
<xsd:element name="linkbaseResponse" type="lbap:LinkbaseResponseType"/>
<xsd:complexType name="LinkBaseType">
    <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="linkBase" type="lbap:LinkBaseType"/>
<xsd:complexType name="ErrorType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
            <xsd:attribute name="class" type="xsd:string" use="required"/>
            <xsd:attribute name="code" type="xsd:int" use="optional"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="error" type="lbap:ErrorType"/>
```

The resulting XLink markup is fully contained in the *linkBase* element. Note that the whole response *is* a linkbase in the sense of XLink even though it contains status information related to LBAP. The consumer of an LBAP response will extract linking elements from the response and ignore the status information.

6.2.3 Example Request and Response

Following an example of a request and the corresponding linkbases response. The example is based on the family linkbase (see appendix B.1). This linkbase holds links for the pages of family Miller.
The request returns all links relevant for the document 'jane.xml'. Relevant means that the starting resources for the traversal are contained in jane.xml. Additionally it ensures that per start resource only two arcs are delivered. Links to pages outside www.stillhard.net are not delivered. The request also expresses that only links stating family relationships for the family Miller are relevant. No arcs with other behavior than "onRequest" and "replace" will be delivered.

```
<LinkbaseRequest
  xmlns="http://www.stillhard.net/2002/lbap"
  xmlns:lbap="http://www.stillhard.net/2002/lbap"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.stillhard.net/2002/lbap
                      http://www.stillhard.net/2002/lbap/lbaptypes.xsd">
  <ProcessingProperties lbChainLength="3" noLocalResources="false" timeLimit="20"/>
  <SizeLimit entity="links" limit="10"/>
  <SizeLimit entity="fanout" limit="2"/>
  <LinkFilter>
    <linkPropertyAssertion>
      <andLPA>
        <linkPropertyAssertion>
          <roleAssertion>
            <equalityMatch>http://www.stillhard.net/2002/xlink/samples/roles/family</equalityMatch>
          </roleAssertion>
        </linkPropertyAssertion>
        <linkPropertyAssertion>
          <titleAssertion>
            <equalityMatch>Family Miller</equalityMatch>
          </titleAssertion>
        </linkPropertyAssertion>
      </andLPA>
    </linkPropertyAssertion>
    <arcFilter>
      <arcPropertyAssertion>
        <andAPA>
          <arcPropertyAssertion>
            <actuateAssertion behaviour="onRequest"/>
          </arcPropertyAssertion>
          <arcPropertyAssertion>
            <showAssertion behaviour="replace"/>
          </arcPropertyAssertion>
        </andAPA>
      </arcPropertyAssertion>
      <startingParticipantFilter>
        <participantPropertyAssertion>
          <resourceAssertion>
            <substringAssertion>
              <contains>http://www.stillhard.net/2002/xlink/samples/resources/jane.xml</contains>
            </substringAssertion>
          </resourceAssertion>
        </participantPropertyAssertion>
      </startingParticipantFilter>
      <endingParticipantFilter>
```

```
        <participantPropertyAssertion>
          <resourceAssertion>
            <substringAssertion>
              <contains>www.stillhard.net</contains>
            </substringAssertion>
          </resourceAssertion>
        </participantPropertyAssertion>
      </endingParticipantFilter>
    </arcFilter>
  </LinkFilter>
</LinkbaseRequest>
```

Lets go through the request to see how it models all these conditions: The global properties in the *ProcessingProperties* element specifies that in case there are links to other linkbases, these should only be followed to a depth of indirection of three. The family linkbase does not reference any additional linkbases so this property is not of relevance here. The second global property asks the linkbase service to respond in at least 20 seconds - either with the generated result or with an error that indicates that the time limit was exceeded.
The result is restricted to a size of ten links each having a maximum fanout of two.
With these processing properties the filtering starts by evaluating the assertions on the links. The family linkbase contains one extended link for which the assertions are fulfilled (the role is *http://www.stillhard.net/da/Work/roles/family* and the title is *Family Miller*). The first condition to include the link in the result is met.
Now the subfilter is applied to all the arcs. Note that if this filter wouldn't return any arcs this would imply that the link was not included in the response either. All arcs pass the property assertion because they all have *onRequest/replace* behavior. Therefore the subfilters for the participants in the arc are evaluated.
The assertions on the starting participants evaluate to false for all starting participants that do not reference the resource *jane.xml* or any subresource in it. The ending resources must have an address in the *www.stillhard.net* domain. The response will only contain arcs for which the assertions on the starting- and the ending participant evaluate to true. Look at the linkbase contained in the response to see which arcs have been filtered out:

```
<linkbaseResponse
  xmlns:lbap="http://www.stillhard.net/2002/lbap"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns="http://www.stillhard.net/2002/lbap"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.stillhard.net/2002/lbap http://www.stillhard.net/2002/lbap/lbaptypes.xsd"
  success="true">
  <linkBase>
    <linkbase xmlns="http://www.stillhard.net/2002/xlink/linkbase"
          xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset">
      <extended xlink:type="extended" xlink:title="Family Miller"
              xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/family">
        <locator xlink:type="locator" xlink:title="Tanja"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/mother"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/tanja.xml"
             xlink:label="mother"/>
        <locator xlink:type="locator" xlink:title="Tom"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/father"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/tom.xml"
             xlink:label="father"/>
        <locator xlink:type="locator" xlink:title="James"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/james.xml"
             xlink:label="son"/>
        <locator xlink:type="locator" xlink:title="John"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/john.xml"
             xlink:label="son"/>
```

```
        <locator xlink:type="locator" xlink:title="Jane"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                         #xpointer(string-range(//*,'Jane'))"
             xlink:label="daughterocc"/>
        <locator xlink:type="locator" xlink:title="James"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                         #xpointer(string-range(//*,'James'))"
             xlink:label="sonocc"/>
        <locator xlink:type="locator" xlink:title="John"
             xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"
             xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                         #xpointer(string-range(//*,'John'))"
             xlink:label="sonocc"/>
        <arc xlink:type="arc" xlink:title="Father"
           xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
           xlink:from="daughterocc" xlink:to="father" xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:type="arc" xlink:title="Father"
           xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
           xlink:from="sonocc" xlink:to="father" xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:type="arc" xlink:title="Mother"
             xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
           xlink:from="daughterocc" xlink:to="mother" xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:type="arc" xlink:title="Mother"
           xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
           xlink:from="sonocc" xlink:to="mother" xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:type="arc" xlink:title="Brother"
           xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
           xlink:from="daughterocc" xlink:to="son" xlink:show="replace" xlink:actuate="onRequest"/>
      </extended>
    </linkbase>
  </linkBase>
</linkbaseResponse>
```

6.2.4 Discussion of the Linkbase Request/Response

The first thing one might notice when looking at a request is its size. Most of it are elements that are responsible to structure the request. The question whether this size is necessary is indeed allowed. We will discuss this and other questions about the design of a linkbase request in the following lines.

Note that this request is a very simple example operating on a linkbase containing only one link and a few locators and arcs. The more information is contained in a linkbase the better will the functionality of a request be exploited and the less redundant seeming elements will the request have. The following points also justify the decision:

- The structure of a request is hierarchical. A mapping to a flat structure would be of high complexity because of the deep nestings
- The flexibility of a request lies in its structure. Optional substructures are very common. When the whole request semantics are described in an other structure it will be quite long as well. The reduction of used space cannot be achieved.
- The structure of a request is aligned with the linking model which helps understanding the request. This would be lost if another data representation was chosen.
- An important feature of a request is that multiple separate filters can easily be merged into one filter (e.g. by adding an additional property assertion to a filter). This is needed when filters are defined in different places and are merged in one request. With other data representation the merging would probably cause a lot of inconvenience.

A linkbase request needs to fulfill requirements and design principles (see chapter 5). For the request we will have a look at the completeness of functionality, the usability and list open issues.

- **Completeness.**
  The completeness of the request can be measured in two dimensions: (1) Does it allow to retrieve every possible subset of the linkbase and (2) does it adopt all functionality required to fulfill the task of delivering all *relevant* arcs and reduce the result to a minimum. To formally proof completeness of the first point it would be necessary to have an approved formal definition of the XLink Infoset. This is currently not at hand, but for the Infoset we are building on (chapter 2.5) we cover all possible subsets. We waive a formal proof since the structure of the request strictly follows the structure of the Infoset and therefore a full coverage can intuitively be seen.
  For the latter point, because of (1), the relevance of a linking information items can sufficiently be expressed. The "size-limiting" requirements worked out earlier are all fulfilled as well: the size limit, delivery of local resources, chained linkbases, etc. (See chapter 5).

- **Usability.**
  Nielsen[14] mentions the following four aspects when defining usability: Learnability, Efficiency of Use, Memorability and Subjective Satisfaction. His definition is targeted on the engineering of graphical user interfaces. But the idea of usability is more general and we can use these aspects to verify the usability of the linkbase service. The learnability of the linkbase request/response concept is stimulated by its closeness to the linking model. The fact that the design of a request follows the hierarchical structure of the linking model makes it transparent and well understandable. Because we don't want to go to deep into this subject we let this also be an enough satisfying reason to state the memorability and subjective satisfaction.

- **Open issues.**
  The main open issue is the delivery of information when the size limit is exceeded. Currently we define to fail such a request, but optionally allow to still deliver a server implementation dependent result. Future work should investigate possible other solutions.

6.3 Protocol Binding

The protocol binding of LBAP takes into consideration that in order to being applicable it needs to be as simple as possible, should not require engraving changes to existing client and server technology and as far as possible adapt the existing infrastructure of the todays web. On the other side, linkbase services can provide complex functionality which demand a certain complexity to the protocol. The approach of LBAP to handle these oppositional positions is, as mentioned earlier, to define and fix a common set and allow future extensions.
In order to implement LBAP, a binding to HTTP 1.1 MUST be supported. A service MAY provide further bindings as required (e.g. a binding to SOAP).

6.3.1 HTTP 1.1 Binding

The HTTP transport protocol suits the requirements of LBAP very well. The request-response exchange pattern is well supported because of the implicit correlation of the request message with the response message.

The getLinkbase operation is bound to a HTTP 1.1 POST method. A request conforming to the definition from section 6.2 is sent in the body of the POST method. The HTTP Content-type header MUST be chosen as text/xml.
LBAP clients and servers MUST implement this binding.

The following example shows an LBAP request using HTTP transport:

```
POST /MyLinkbase HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
    <LinkbaseRequest>
        <ProcessingProperties lbChainLength="3" noLocalResources="false" timeLimit="20"/>
        <SizeLimit entity="links" limit="10"/>
        <SizeLimit entity="fanout" limit="2"/>
        <LinkFilter>
            <linkPropertyAssertion>
...
```

In case of success an HTTP 200 "OK" MUST be returned from the linkbase containing a response structure as defined in 6.2:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
    <linkbaseResponse success="true">
        <linkBase>
            <linkbase>
                <extended xlink:title="Family Miller">
...
```

In case of failure a HTTP 500 "Internal Server Error" MUST be used to indicate the processing error. It MUST contain a response structure as defined in 6.2:

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
    <linkbaseResponse success="false" operationError="false" protocolError="false" timeLimitExceeded="true">
        <errorMessage>
            Time limit exceeded
        </errorMessage>
    </linkbaseResponse>
```

All other HTTP status codes have no LBAP specific meaning and SHOULD not occur.

The handling of a response with an unsuccessful state depends on the implementation. A user-agent might present the error to the user (with the delivered error description) or trigger alternative actions.

6.4 LBAP Discussion

The simplicity and the reduction of the functionality to a common nominator while leaving space for extensions allows to easily implement clients and servers that will successfully interact. The extensibility mechanism allows to tunnel requests with enhanced query semantics through LBAP. For example could a LBAP request contain information about user interests.
It is one of the strengths of LBAP that it reduces the functionality to querying the XLink infoset. It allows to establish the protocol without excluding certain hypermedia concepts from

the beginning on.
To measure the quality of the proposal we will investigate how well it fulfills the requirements that we posted in chapter 5. Table 6.1 lists all design principles and requirements and indicates in the second column the degree of fulfillment. - means not fullfilled, + partly fulfilled or could be improved, and ++ stands for fullfillment. Refer to chapter 5 for the detailed descriptions of the requirements.

Requirement		**Comment**
Supported link model	++	XLink fully supported
Access modes	++	No write access allowed
Adaptable to the existing Web infrastructure	++	No changes in related standards are required
Simplicity	++	As easy as possible
Extensibility	+	Metadata about extensions not supported
Openness	++	All possible XLink extensions are supported
Related standards	+	A proprietary query mechanism is introduced with LBAP
Configuration Language	-	No configuration language is defined
Generic Links	+	Not directly supported. The extension mechanisms allow to use generic links
Access to every attribute in the XLink namespace	++	Supported
Access to attributes of linking elements that are from a non-XLink namespace	+	Supported. The server has however not much flexibility in restricting access
Resources that are referenced in semantic attributes	++	Can be accessed through LBAP and it does not restrict their use in any way
Restrict the size of the returned linkbase	++	Supported
Delivery of local resources	++	Can be controlled
Chained linkbases	++	Can be controlled
Processing steps to integrate query results must be provided	+	Pass all results to the XLink interpreting application is enough.
Processing model	+	Metadata aspects not covered well

Table 6.1: Evaluation of LBAP with respect to the posted requirements

Chapter 7

LBAP Service

To illustrate the usage of LBAP this chapter describes a linkbase service. The service is described using the Webservice Description Language (WSDL[1]) and therefore also shows that LBAP conforms to existing activities. After the definition of the service the chapter proceeds with examples of LBAP extensions.

7.1 Linkbase Service

The linkbase service imports the schema definitions for the request and response. It defines the types used in the input and output messages. The LinkbaseServicePort contains the GetLinkbase operation. This operation is a request-response primitive as indicated by the usage of the input and output elements. The optional fault element is skipped because the linkbase response already defines failure semantics.
LinkbaseService supports two protocol bindings - the mandatory HTTP POST and a SOAP binding.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="LinkbaseService"
    targetNamespace="http://www.stillhard.net/da/2002/01/LBAPService"
    xmlns:tns="http://www.stillhard.net/da/Work/LBAPService"
    xmlns:lbasd="http://www.stillhard.net/da/2002/01/LBAPServiceDefinition"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    elementFormDefault="qualified">
    <!-- import the types -->
    <import namespace="http://www.stillhard.net/da/2002/01/LBAPServiceDefinition"
              location="http://www.stillhard.net/da/Work/LBAPService/servicedefinition.xsd"/>

    <message name="LBRequestInput">
        <part name="body" element="lbasd:LinkbaseRequest"/>
    </message>
    <message name="LBRequestOutput">
        <part name="body" element="lbasd:LinkbaseResponse"/>
    </message>
    <portType name="LinkbaseServicePortType">
        <!-- Request-response operation to retreive a linkbase -->
        <operation name="GetLinkbase">
            <input message="tns:LBRequestInput"/>
            <output message="tns:LBRequestOutput"/>
            <!-- no fault element. Errors are handled by the protocol -->
        </operation>
    </portType>
    <!-- soap 1.1 HTTP binding -->
    <binding name="LinkbaseServiceSoapBinding" type="tns:LinkbaseServicePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLinkbase">
```

[1] http://www.w3.org/TR/wsdl

```
            <soap:operation
                soapAction="http://www.stillhard.net/da/LinkbaseServiceSOAP/QueryLinkbase"
                style="document"/>
            <!-- input and output without any special encoding -->
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <!-- HTTP POST binding -->
    <binding name="LinkbaseServiceHTTPBinding" type="tns:LinkbaseServicePortType">
        <http:binding verb="POST"/>
        <operation name="GetLinkbase">
            <http:operation location="GetLinkbase"/>
            <input>
                <mime:content type="text/xml"/>
            </input>
            <output>
                <mime:content type="text/xml"/>
            </output>
        </operation>
    </binding>
    <service name="LinkbaseService">
        <documentation>Prototype of Linkabase Service</documentation>
        <port name="LinkbaseServicePort" binding="tns:LinkbaseServiceSoapBinding">
            <soap:address location="http://www.stillhard.net/da/LinkbaseServiceSOAP"/>
        </port>
        <port name="LinkbaseServicePort" binding="tns:LinkbaseServiceHTTPBinding">
            <http:address location="http://www.stillhard.net/da/LinkbaseServicePOST"/>
        </port>
    </service>
</definitions>
```

7.2 LBAP Extensions

LBAP supports two extension types - extension of the request and definition of additional operations. The former we illustrate by outlining a generic-link service, the latter with an operation to extract meta information about the linkbase.

7.2.1 Request Extension

Generic links are based on words. This means that the starting resource of a link is described by a string rather than by a URI reference. Of course, ultimately the starting resource will be expressed as a URI reference (containing an XPointer expression) but every possible starting resource of one link will consist of a string representing the same word. These links are called generic, because they are not bound to a predefined document, but rather every document can potentially contain a starting resource. Generic links are of considerable benefit in that a new document may be created and immediately have access to all the generic links that have been defined previously.
A generic link could for example associate *http://www.stillhard.net* with the strings *"Christian Stillhard" and "Ch. Stillhard"*. Note that the ending resources of a generic link could be generic as well. For example could a link with semantic property *"german"* associate *"generic"* with *"generisch"* indicating the german translation of a word. If the ending resources are resolved against a document that contains explanations and phonetic spelling definitions we have a dictionary service.

A generic link needs to be transformed to XLink prior to delivery via LBAP because there exists no concept in XLink for generic links. This step involves the calculation of URI references:

```
%word%  --> %someURI%/#xpointer(string-range(//*,%word%))
```

For the example mentioned earlier:

```
"Christian Stillhard" --> %someURI%/#xpointer(string-range(//*,'Christian Stillhard'))
```

In order to perform this transformation upon an LBAP request, the linkbase needs to have knowledge about which *words* should be used and what *URI* should be used. Multiple ways are possible to acquire this knowledge. To mention some:

- The linkbase can index sites on the web similarly as todays search engines do in order to gather information about occurrences of the words it contains links for. The indexing process might also analyze the context of the occurrence to increase the quality of the links. The LBAP request would then filter these links for relevance, meaning that it would contain an assertion on the URI reference of the starting resource. This approach has the disadvantage that the links become static again to a certain degree. Also, if the linkbase has not indexed a certain document it would not return any links in the LBAP response.

- The linkbase receives the URI reference of the starting resource from an assertion on the starting participant in an LBAP request. It can then request the document, index and analyze it on the fly, generate the links and return them in the LBAP response. With this approach the links keep their generic nature, but the request will not perform well. Additionally, the starting resource needs to be publicly accessible for the linkbase (this is also necessary for the previous approach).

- The client can deliver a description of the starting resource in the LBAP request. Together with the LBAP specific information in the request (processing properties, size limit, filter, etc.) the linkbase can transform the generic links into XLink and return them in the LBAP response. With this solution the drawbacks of the other two are solved: The links are generic and *live* (if a new generic link is added it is immediately available), the linkbase is not required to have knowledge of the document prior to the request, no additional request is necessary and the document is not required to be publicly available.

The definition of the LBAP request does not provide a datastructure to deliver a description about a resource. The request needs to be extended. Figure 7.1 illustrates this scenario.
The additional data that needs to be delivered is a *description* of the starting resource. This could be a set of keywords that occur in the document, a description encoded in a standardized way like for instance RDF[6] or the whole resource.

The WSDL service definition for a linkbase that provides generic linking needs to contain the following additional declarations:

```
...
<types>
    <xsd:schema
        targetNamespace="http://www.stillhard.net/da/2002/01/LBAPGenericLinkServiceDefinition"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:lbasd="http://www.stillhard.net/da/2002/01/LBAPServiceDefinition"
        xmlns="http://www.stillhard.net/da/2002/01/LBAPGenericLinkServiceDefinition"
        elementFormDefault="qualified">
        <xsd:import
            namespace="http://www.stillhard.net/da/2002/01/LBAPServiceDefinition"
            schemaLocation="http://www.stillhard.net/da/Work/LBAPService/lbaptypes.xsd"/>
        <xsd:element name="GenericLinksLinkbaseRequestType">
            <xsd:complexType>
                <xsd:complexContent>
```

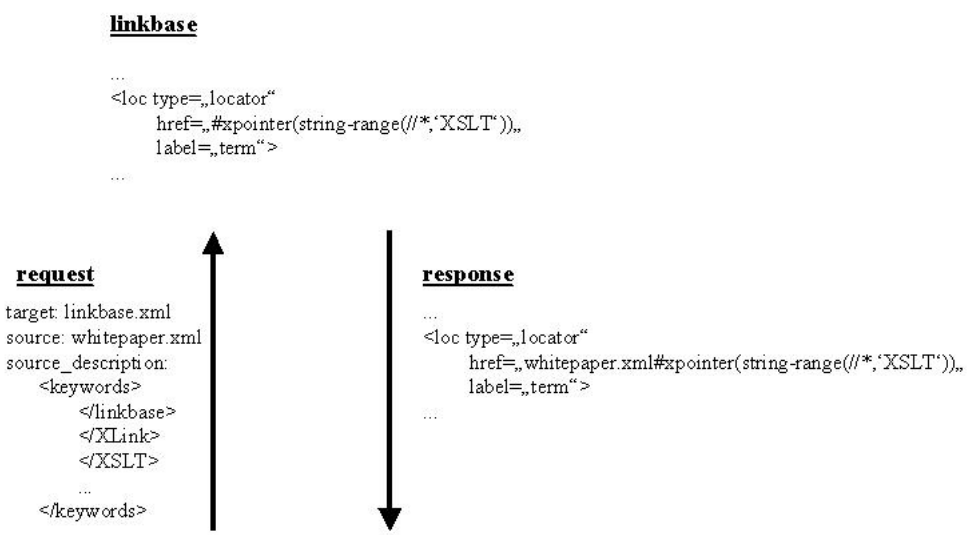

Figure 7.1: Request and response to a linkbase containing locators with fragment identifiers

```
                <xsd:extension base="lbasd:LinkbaseRequestType">
                    <xsd:sequence>
                        <xsd:element ref="RecourceDescription"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RecourceDescription">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:restriction base="xsd:string">
                    <xsd:attribute name="type" type="xsd:string" use="required"/>
                </xsd:restriction>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
</types>
<message name="LBRequestInput">
    <part name="body" element="GenericLinksLinkbaseRequestType"/>
</message>
<message name="LBRequestOutput">
    <part name="body" element="lbasd:LinkbaseResponse"/>
</message>
...
```

In the *types* element the LBAP request is extended with an additional element conforming to the schema definition of the request. The *ResouceDescription* element contains a string and is of a certain type. The type attribute could take values ”RDF”, ”DOC”, ”DC”, etc. indicating the encoding of the resource description. The input *message* element differs in that it now needs to contain a *GenericLinksLinkbaseRequestType* element. Note that extensions to LBAP are only valid as long as the request instance is valid against the LBAP schema. This means that e.g. the filter must not be altered. The addition of the resource description is however valid (refer to the definition of the request in section 6.2).

7.2.2 Additional Operations

The goal of a metadata request is to gather information about the linkbase in terms of what what resources it links, what traversal it defines, or which semantic properties it uses. Conceptually, this information is needed to map the context of a linkbase access onto a request (See chapter 4 for an explanation of dynamic and static context, users interest, etc.). In other words, without metadata about the linkbase it is difficult to create meaningful requests. The goal is not to allow full introspection of the linkbase, but rather gather an overview of the contents.

In contrast to the extension of LBAP to support generic links, a metadata request does not fit into the *getLinkbase* operation. It forms a closely related, but independent operation of a linkbase service. To be precise, an additional operation does not belong to LBAP and is mentioned here only to show LBAP in an extended service framework. Hence, the WSDL definition would need to be enhanced with additional type, message, portType and binding definitions.

The specification of a metadata request is not subject of this work. Still some questions and observation shall be mentioned:

- *Approach*
  Inspection is a subject with opportunities and threats in equal measure. It is important to focus on the information that is relevant to retrieve and refrain from full inspection. The question "what needs to be known about the linkbase" should be asked rather than "how should the mechanism be to retrieve meta information about the infoset". However, note that it would also be possible to start with the second approach and let the linkbase server restrict what information it really wants to return to a user. This would then be similar to the concept used in RDBMS: A user can only request the schema of a table if he is granted the necessary rights.

- *What information can/should be retrieved*
  Is it enough to request semantic information only (role, title, arcrole, proprietery attributes, title elements and proprietary elements)? What about the URI references in locators? Conceptually the client should not be able to request them. But this would be a threat because it would for instance allow to introspect the set of available documents of a website. Other queries that should be considered as a metadata request:

  - Number of links, number of link participants, number of traversal definitions, etc.
  - Does the linkbase contain local resources referenced in inbound arcs?
  - Do the linking elements have proprietary attributes?
  - Does the linkbase specify a certain proprietary attribute. E.g. does the linkbase provide an attribute that rates the quality of a link, etc.

- *Size of the response*
  The amount of information that can result from a metadata request should be considered. Size limits might be necessary for metadata requests as well.

- *Orthogonality to the getLinkbase operation*
  Note that much information can also be gathered by the LBAP *getLinkbase* operation. For instance it makes little sense to allow a metadata query that asks whether a linkbase contains links starting from a certain resource. This can be emulated by a LBAP request with a low size limit.

- *Degree of structure of a metadata response*
  If you consider a request for semantic properties of the linking entities contained in the linkbase, the question arises how this information shall be structured in the response. The straightforward approach is to deliver lists of values, one for each type of semantic property - a list of link roles, a list of arc roles, etc. The use of these lists is questionable, because for instance it cannot be decided whether a role belongs to a linking entity or to a resource entity. Therefore the structure should allow to conclude on the affiliation of a semantic property to a linking entity. If this requirement is considered and the possibilities to structure are exploited to the extreme, one could come to a proposal to introduce the hierarchy of the linking model in the metadata response as well. Meaning that e.g. a role property is marked as related to a set of semantic properties for a resource which are again related to a set of properties for the arc in which the resource participates. The quality of the response is very high in this case, but the quantity probably gets close to the size of the response of a *getLinkbase* operation. To summarize, the structure of the metadata response needs to be carefully designed and different objectives need to be considered intensively.

Chapter 8

Proof of Concept: Implementation

Within this project, a prototype to execute an LBAP request on a linkbase has been conceived and implemented. The prototype is based on the LBAP definition in chapter 6 and the principles discussed in chapter 4. The objective is to clarify and substantiate the idea of operating on the XLink infoset and the processing of LBAP filters. Importance was attached to the comprehensibility of the provided code. It was not the intention to create a library that can be used in LBAP protocol handlers. Performance, full coverage of the functionality or configurability are not the main criteria. To improve clarity the interaction with the underlying protocol is also not part of the prototype.
The first section in this chapter gives a general overview of the prototype. The second and third section describe the design principles and implementation, separated into infoset operations and request processing. In the last section the prototype implementation is summarized.

8.1 Outline of the Prototype Implementation

The prototype covers the processing of a request on the server side of a *getLinkbase* operation. Figure 8.1 shows the parts in a linkbase access that the prototype implements.

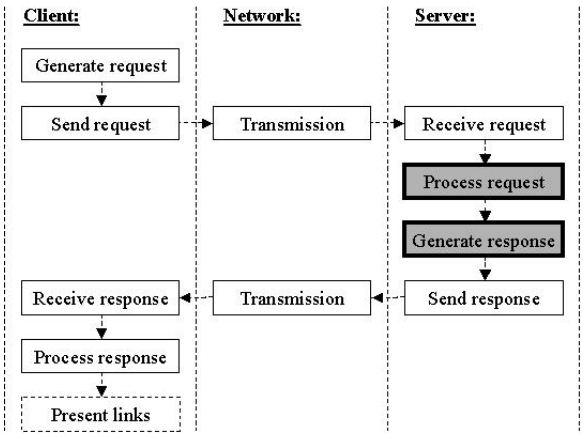

Figure 8.1: LBAP prototype: Covered Functionality

8.1.1 Concept

The input for the prototype consists of an XLink linkbase and an LBAP request. Five phases can be differentiated in order to generate the output, which is an LBAP response.

1. *Generate infoset*
   The linkbase is analyzed and the infoset is generated. In this process the markup constraints and -defaulting in XLink are resolved and the linking entities are brought into a unified representation. This is necessary because the LBAP filters operate on the infoset and not the XLink markup itself. Note that this step is not required to be done for every request. Depending on the requirements on the actuality of the infoset and the frequency of changes to the linkbase a periodical generation of the infoset is sufficient.

2. *Analyze request*
   The LBAP request is prepared for execution on the infoset in this phase. It is tested for correctness and validity. This step is optional and all the tasks can also be performed in the next phase. But it is explicitly required in our implementation.

3. *Execute request on the infoset*
   The infoset is filtered according to the filter criteria in the request. Processing properties like size- and time limits or constraints on the delivery of extensions are taken into account. The result of this phase is the filtered infoset - reduced to the relevant linking entities - and status information indicating success or failure of the processing.

4. *Generate linkbase from the infoset*
   The filtered infoset is serialized to XLink. Again, the markup constraints and -defaulting in XLink need to be considered.

5. *Generate response*
   A response object, conforming to the LBAP definition is generated. It includes the generated linkbase and comprehends the status information from phase three.

The differentiation of these phases will not be as clear as here in other implementations of LBAP request processing. The generation of the infoset might happen implicitly or be obsolete if the links are stored in an appropriate way. It depends on the implementation and on the storage of the links.

8.1.2 Implementation

The implementation is fully coded in XSL. For the different phases stylesheets describe the involved transformations. The choice has fallen to XSLT as the technology to be used in the prototype because it is concise and well understandable. The concepts presented in the prototype can be encoded in some hundred lines of code. This allows to get an impression at first sight. For every phase there is one stylesheet (Table 8.1).
All used documents, including the mentioned schema definitions can be found in the appendices.

Figure 8.2 shows the flow of processing with the involved stylesheets.

8.1.3 Coverage

The following features of LBAP are covered in the prototype:

Generation of an Infoset from an XLink Linkbase

- Fully implemented. It operates exclusively on the XLink attributes
- Covers any possible proprietary extension (non-Xlink markup)

Phase and Stylesheet	Input	Output
Phase: generate infoset, Stylesheet: linkbase2infoset.xsl	The XLink linkbase. This is a valid XML document containing elements with attributes from the XLink namespace	An XML representation of the XLink infoset. The document is an instance of the schema defined in xlinkinfoset.xsd
Phase: analyze request, Stylesheet: request_processor_generator.xsl	The LBAP request (an instance of the schema defined in lbaptypes.xsd)	A stylesheet that is used to process the request
Phase: execute request on the infoset, Stylesheet: request_processor_general.xsl and the output of the *analyze request* phase	The infoset representation generated in phase one	The filtered infoset and a document containing status information
Phase: generate linkbase from the infoset, Stylesheet: infoset2linkbase.xsl	The filtered infoset from the previous phase	The filtered XLink linkbase
Phase: generate response, Stylesheet: response_generator.xsl	The filtered XLink linkbase and the status information generated in phase three	The LBAP response

Table 8.1: Values of the XLink show attribute.

- Handles arc-type elements which have missing *from* and/or *to* attributes
- Handles extended-type elements which have only resources and/or locators, but no arcs
- Supports inbound links (local resources)

Processing of a LBAP Request

- Checks the following size limits and generates an error message in the response:
  - Number of links
  - Number of arcs per link
  - Number of participants per link
- Filters links, arcs, and participants by the following properties:
  - title
  - role/arcrole
  - actuate and show
  - resource of locators (href attribute in XLink)
  - non-XLink attributes (only by value, not by namespace)
- Property assertions support:
  - Logical operators and, or, and not
  - String equality operator
  - Substring comparison operator: contains

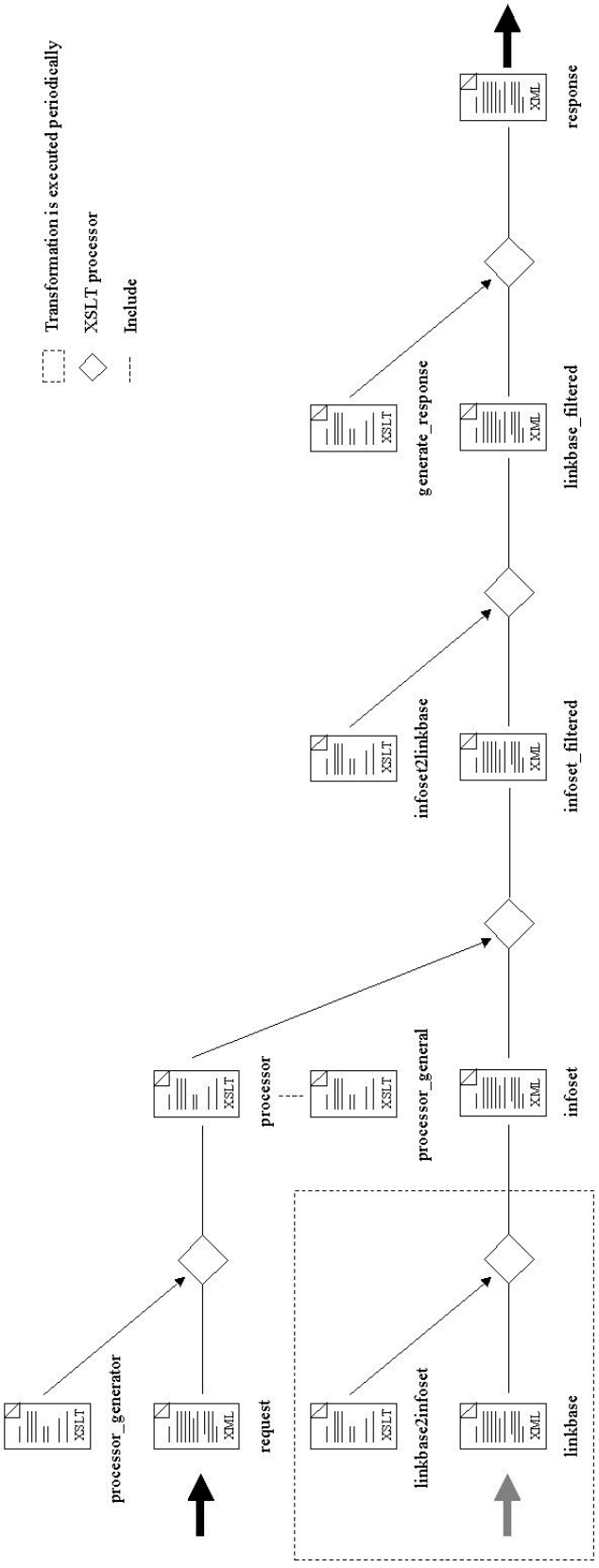

Figure 8.2: Information flow in the LBAP prototype

Generation of an XLink Linkbase from an Infoset

All necessary transformations implemented except for the handling of infosets that have arcs defined for every combination of the resources in the link. Since this should anyway not be the case after a filter process we left this open.

Generation of the Response

- Evaluation if success or failure
- Error handling for general operation errors and size limit crossing
- Insertion of the error messages that where generated during the processing of the request
- Insertion of the resulting linkbase

Limitations

- If the document type of a Linkbases is defined in a DTD then the result will not be valid because *DOCTYPE* information is not accassible by the XSLT processor. If the Linkbase is a XML schema instance, then the XSLT processor needs to use a schema-validating parser to make the prototype work properly. This is because the XLink *type* attribute is usually defined as an attribute with a *fixed* value and this information is only accessible in the XML Infoset if the parser is schema-validating.
- Multiple link and arc filter in one LBAP request are not supported. According to the definition of LBAP the results of the filters would be unioned. Note that in spite of this limitation all possible filter criteria can still be represented.
- Filters for elements that have no XLink *type* attribute are not supported.
- Filters for title-type elements are not supported.
- Global request processing properties that are not listed above are not supported.
- If the size limit is exceeded in a request, this always results in an error. The delivery of a subset of the relevant linking items is not supported. Note that this is anyway a convenience function only.

The prototype covers all aspects of infoset generation and request processing. To eliminate the mentioned limitations these concepts just need to be applied again. However with the XSLT implementation the complexity would increase exponentially if new functionality is added. The length of the code is at a point where it is getting hard to keep the overview. This is the reason why we avoided the implementation of some features.

8.1.4 Running the Prototype

All that is required to run the prototype is an XSLT processor which conforms to the XSLT 1.0 recommendation[24] and implements the extensions as specified in the XSLT 1.1 Working Draft.

For the windows platform a batch program that automates the transformation flow is provided in appendix C.6.

8.2 Specific Design

This section discusses the transformations in more detail and is intended to help understanding the implementation. The first part concentrates on the generation of the infoset representation of a linkbase and the serialization of the infoset into XLink. The second part gives an insight to the processing of the request. The last part discusses the generation of the LBAP response.

8.2.1 Infoset Operations

Linkbase to Infoset

The stylesheet that transforms an XLink linkbase into its infoset representation can be found in appendix C.4. The source needs to be a linkbase as defined by the XLink recommendation. Refer to the exemplary linkbase DTD (appendix A.1) or schema (appendix A.2) to see how a linkbase document can be defined. The output is an instance of the infoset schema (appendix A.2).

The transformation basically copies all information that is contained in the source document to elements from the infoset namespace. Templates are implemented for four of the five values that an XLink *type* attribute can have: *extended, locator, resource, title, and arc*. For *extended* type elements first the content is copied and then the templates for its XLink children are applied. The *locator* and *resource* type elements are copied as well, including the *title*-type element. The copy process is not just a simple XSLT *copy-of* function, but is performed by special templates in order to control what content is copied and to ensure that for the child elements templates are applied again.
For *locators* and *resources* an id is generated and attached. The following line of code produces the id:

```
<xsl:attribute name="infoset:id">
    l<xsl:number count="*[@xlink:type='locator' or @xlink:type='resource']"/>
</xsl:attribute>
```

The id attribute is defined as an XML ID-type attribute and therefore has to match the *Name* production. That is the reason for the preceding "l" in an id. The value is the sequential number of the locator or resource node in the source document. The *count* attribute specifies that only locators and resources are counted.

Note that the templates ensure that the hierarchy in the link only consists of parent/child relationships. According to XLink e.g. a *locator* type element has no XLink specific meaning if it is not a direct child of an *extended*-type element.

Special handling is needed for the *arc*-type elements for the following reasons:

- An arc in the infoset connects exactly two resources. If the value of the *from* or *to* attribute of an arc occurs in multiple *label* attributes in locators or resources, then an infoset arc element is created for every occurrence of the value.
- If the *from* and/or *to* attribute(s) are missing, then this means that the arc relates to all resources that are defined in the link.
- If there are no arcs present in the link, then an infoset arc element needs to be created for every possible pair of resources.

All three issues can be covered in one strategy. The idea is that the infoset arc element is not copied in the template that matches arc-type elements. In that template the values of the *from* and *to* attributes are memorized and used as parameters to a second template that matches locators. This template is decorated with a mode attribute with value *startParticipant*.

Consequently it is applied to every locator and can check whether its *label* attribute matches the memorized value of the arcs *from* attribute. If the start participant of the arc is found, the pattern is applied again, this time to find the *endParticipant*. Only then, the infoset arc element is copied.
Missing *from* and *to* attributes are handled in the templates for the start and end participants. When the id's are compared, a missing *from* or *to* attribute means that that locator belongs to the arc (the *$fromlabel* is the parameter with the memorized value of the *from* attribute):

```
<xsl:if test="@xlink:label=$fromlabel or $fromlabel=''">
    <!-- now for every endParticipant -->
    <xsl:apply-templates select="../*[@xlink:type='locator' or @xlink:type='resource']" mode="endParticipant">
        <!-- here go the parameters -->
    </xsl:apply-templates>
</xsl:if>
```

This excerpt also shows how the templates for locators and resources are called in different modes.

With this strategy it is easy to implement the handling of links that contain no arcs at all. The template with name *noArc* is called if the link has no arcs and it behaves like an arc with both attributes *from* and *to* missing. According to the XLink recommendation such an arc must in fact be interpreted as an arc connecting every resource with every other - and this exactly what happens in the stylesheet.

The comments in the stylesheet should help to understand the rest.

Infoset to Linkbase

The stylesheet in appendix C.5 inverses the just described transformation. It takes an infoset representation of the linkbase as input and transforms it into an XLink linkbase.

With the understanding of the previous section this should be a walk in the park. The linkParticipants are again just copied. Worth mentioning is the mechanism to recognize arcs that are merged because they reference resources with the same *from* or *to* attribute value. The key is a parameter that holds pairs of *from/to* attribute values of arcs that have already been copied. If an arc references resources that result in the same *from/to* combination, then it is not copied at all, because it is redundant. The parameter is as a result built up during the process and could look as follows:

```
l3-l7;l1-l2;l15-l2;
```

Remember that the ids start with an "l" character. Pairs are built with a hyphen between them and separated with a semicolon.
Before an arc is copied to the output, the following test is made ($donePairs is the parameter with the *from/to* pairs):

```
<xsl:if test="not( contains($donePairs, $fromto) )">
```

The implementation of the transformations for the infoset was relatively straight forward. One major concern of the implementation is that everything needs to be copied correctly. Conceptually the infoset is not another representation of the linkbase. It is additional information that is available when operating on the linkbase. At most it is an interface through which the linkbase is accessed. But with our transformation we actually copied the information from the linkbase and augmented it. The problem with this approach is that care must be taken that really everything is copied. E.g. the public and system id of the DOCTYPE are not accessible in the XSLT-processor and this information is lost during the transformation. But it is in the nature of this prototype implementation that that a copy needs to be made. Another possibility would be to generate a separate infoset document and in the filtering use both, the infoset and the original linkbase, for the processing. But this approach is probably more complex than the way we have chosen.

8.2.2 Request Processing

The processing of a request consists of two steps: first, the request is transformed into a stylesheet, then this stylesheet is applied to the infoset representation of the linkbase (refer to Figure 8.2 for an illustration of this two steps). The first step - the transformation of the request - is not discussed in detail here. Rather we base the discussion of the second step on the result of the transformation of the example request from section 6.2.3:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset"
    xmlns="http://www.w3.org/1999/XSL/Transform">
    <xsl:include href="http://www.stillhard.net/2002/lbap/request_processor_general.xsl"/>
    <xsl:variable name="arcLimit">99999999</xsl:variable>
    <xsl:variable name="linkLimit">10</xsl:variable>
    <xsl:variable name="participantLimit">99999999</xsl:variable>
    <xsl:template name="linkPropertyAssertion">
        <xsl:variable name="vAnd11">
            <xsl:if test="@infoset:role = 'http://www.stillhard.net/2002/xlink/samples/roles/family'">1</xsl:if>
        </xsl:variable>
        <xsl:variable name="vAnd12">
            <xsl:if test="@infoset:title = 'Family Miller'">1</xsl:if>
        </xsl:variable>
        <xsl:if test="$vAnd11 != '' and $vAnd12 != ''">1</xsl:if>
    </xsl:template>
    <xsl:template name="arcPropertyAssertion">
        <xsl:variable name="vAnd11">
            <xsl:if test="@infoset:actuate = 'onRequest'">1</xsl:if>
        </xsl:variable>
        <xsl:variable name="vAnd12">
            <xsl:if test="@infoset:show = 'replace'">1</xsl:if>
        </xsl:variable>
        <xsl:if test="$vAnd11 != '' and $vAnd12 != ''">1</xsl:if>
    </xsl:template>
    <xsl:template name="startParticipantPropertyAssertion">
        <xsl:variable name="idref">
            <xsl:value-of select="@infoset:idref"/>
        </xsl:variable>
        <for-each select="//infoset:participant[@infoset:id = $idref]">
            <xsl:if test="contains(@infoset:resource,
                          'http://www.stillhard.net/2002/xlink/samples/resources/jane.xml')">1</xsl:if>
        </for-each>
    </xsl:template>
    <xsl:template name="endParticipantPropertyAssertion">
        <xsl:variable name="idref">
            <xsl:value-of select="@infoset:idref"/>
        </xsl:variable>
        <for-each select="//infoset:participant[@infoset:id = $idref]">
            <xsl:if test="contains(@infoset:resource, 'www.stillhard.net')">1</xsl:if>
        </for-each>
    </xsl:template>
</xsl:stylesheet>
```

First, we have a look at the sequence of transformations applied to the infoset. See Figure 8.3 for an illustration. The diamonds indicate input, output or temporary data. Some of them represent node sets (*linkSet, empty and element*), some one or more string variables (*processingProperties and relevantParticipants*), and some mixed data. Rectangles are transformations which in most cases represent XSLT templates.
Remember that some templates to process a request don't depend on the request. They can be found in the request_processor_general.xsl stylesheet (appendix C.1). In the picture these templates have a white background. Gray shaded shapes represent templates and variables that are generated from the request and are only temporarily available. It is amazing how much of the functionality is general. Only the processing properties and property assertions depend on the

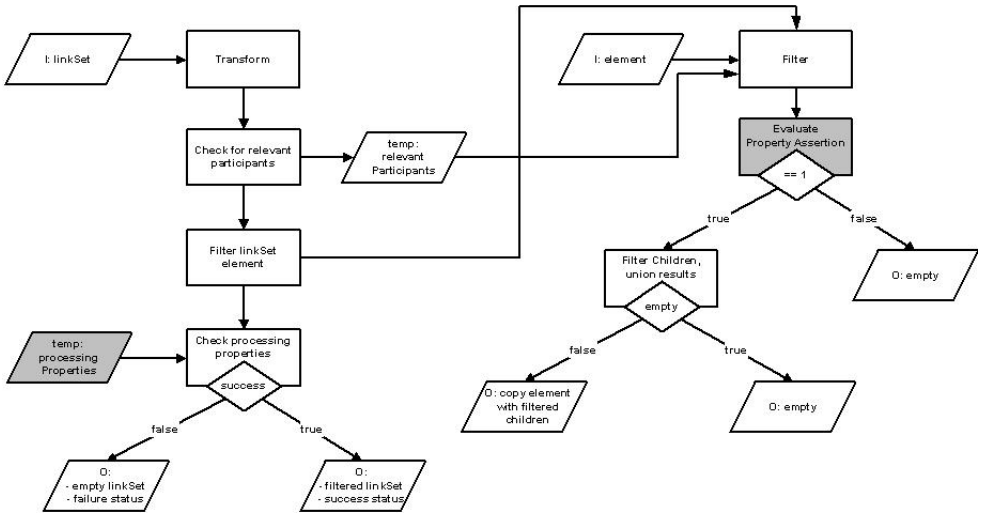

Figure 8.3: Processing of a request

request. Everything else, namely the control over a filter, the evaluation of processing properties (e.g. size limits) and the generation of status information is generic.
Note that, according to the definition of a LBAP filter, property assertions result in a boolean value (in XSLT represented by the "1" character for true and all other character for false), while a filter results in a node set.
We will now discuss all the transformations mentioned in Figure 8.3.

- *Transform*
  This is the entry point to the transformation of the infoset and relates to the *request_processor_general.xsl* as a whole.

- *Check for relevant participants*
  The result of this step is a string variable containing the values of the *idref* attribute of start and endParticipant elements. The *idref* is only added if the start or endParticipant, respectively, is not filtered out. Consequently this check needs to apply all filters - but it does so in a different mode to indicate that the result is not a node set, but rather a string with ids. The discussion of the filtering comes later, just note here that all filter templates have a duplicate with a *getRelevantParticipants* mode attribute value.
  The string with the ids of relevant participants will later be used in the filter.

- *Filter linkSet element*
  The appropriate filter is applied to the linkSet element. It returns the filtered linkSet. In the stylesheet the linkSet is stored in the global *filterResult* variable.

- *Check processing properties*
  The prototype supports three processing properties; the size limits for number of links, number of arcs, and number of participants. These limits are used to define global variables when the request is analyzed. The processed request holds three variables: *arcLimit, linkLimit and participantLimit*. They are initialized with a high default value if the request does not specify a limit.
  These values are used to check the linkSet in the *filterResult* variable:

```
<xsl:variable name="failure">
    <xsl:if test="count( $filterResult/*/*/infoset:arc ) &gt; $arcLimit">arcLimit</xsl:if>
    <xsl:if test="count( $filterResult/*/infoset:link ) &gt; $linkLimit">linkLimit</xsl:if>
    <xsl:if test="count( $filterResult/*/*/infoset:participant ) &gt; $participantLimit">
```

```
        participantLimit
    </xsl:if>
</xsl:variable>
```

The failure variable will contain an empty string if everything is fine, or strings indicating the reason for the failure if a size limit is exceeded. Depending on this value the stylesheet writes a status document and writes the linkSet to the output.
The status document contains the status of the request processing (success/failure) and optionally error elements that specify the type of error and an error message. The type of the status document is defined as follows:

```
<!ELEMENT status (error*)>
<!ATTLIST status
    success (true | false) #IMPLIED
>
<!ELEMENT error (#PCDATA)>
<!ATTLIST error
    type CDATA #REQUIRED
>
```

- *Filter*
  For every element in the infoset (*link, arc, start and endParticipant, and participant*) a filter template is defined in *request_processor_general.xsl*. A filter calls the corresponding propertyAssertion template and evaluates the return value. If it is false, then the filter returns an empty node set. If it is true, then filter for its subelements are invoked. If these filters all return empty node sets then the filter returns an empty node set itself. If not, the currently filtered element is written to the output together with the result of the subfilters. Reproduce this pattern in the following code excerpt (it exemplarily shows the arc filter. All other filters are similar):

```
<xsl:template match="infoset:arc">
    <!-- evaluate arc property assertion -->
    <xsl:variable name="valid">
        <xsl:call-template name="arcPropertyAssertion"/>
    </xsl:variable>

    <xsl:if test="$valid != ''">
        <!-- arc is valid -->
        <!-- filter participants, store in variables -->
        <xsl:variable name="startParticipant">
            <xsl:apply-templates select="infoset:startParticipant"/>
        </xsl:variable>
        <xsl:variable name="endParticipant">
            <xsl:apply-templates select="infoset:endParticipant"/>
        </xsl:variable>

        <!-- the arc should only be output if none of the participants is filtered out -->
        <xsl:if test="count( $startParticipant/infoset:startParticipant ) &gt; 0
                          and count( $endParticipant/infoset:endParticipant ) &gt; 0">
            <!-- the arc will be taken, so output it -->
            <xsl:copy>
                <xsl:apply-templates select="@*"/>
                <xsl:copy-of select="$startParticipant"/>
                <xsl:copy-of select="$endParticipant"/>
            </xsl:copy>
        </xsl:if>
    </xsl:if>
</xsl:template>
```

The example shows that the arc is only relevant if both, the start and endParticipant, are relevant as well. Note that the arcPropertyAssertion template is produced from the request.

What is missing until now, is the structure of the propertyAssertion templates.
As with the filters, there is a property assertion template for every infoset element. They are

called by name, but always out of the corresponding filter and therefore in the context of the corresponding infoset element. The task of a propertyAssertion is to check the values of the attributes of the infoset element. For instance the *arcPropertyAssertion* checks the *show, actuate, role, etc.* properties of an arc element. The complexity of a propertyAssertion results from the sophisticated features that a LBAP request provides to constrain properties. Namely the valueAssertion that can be an equality or substring operation and the logical operators that can be used to combine assertions. Following the arcPropertyAssertion as it resulted from the example request. Remember, the request expressed that an arc must have *show="replace" and actuate="onRequest"* behavior::

```
<xsl:template name="arcPropertyAssertion">
    <xsl:variable name="vAnd11">
        <xsl:if test="@infoset:actuate = 'onRequest'">1</xsl:if>
    </xsl:variable>
    <xsl:variable name="vAnd12">
        <xsl:if test="@infoset:show = 'replace'">1</xsl:if>
    </xsl:variable>
    <xsl:if test="$vAnd11 != '' and $vAnd12 != ''">1</xsl:if>
</xsl:template>
```

The actual assertions on the attribute values are done in an *xsl:if* element. The result - "1" for success, "" otherwise - is stored in two variables. The *and* operation is then done in another *xsl:if* element, resulting in "1" if both operands are not empty strings.
This pattern is extended hierarchically if more logical operations are used for the property assertion.

8.2.3 Response Generation

The stylesheet in appendix C.3 transforms the filtered linkbase into a LBAP response. The input of the stylesheet is the XLink linkbase which was produced by a infoset-to-linkbase transformation and the status document, written during the processing of the request.
It's implementation is straight forward and should not need any detailed explanation.

8.3 Summary

The declarative nature of XSL has forced us to utilize its functionality to an extreme. It was exciting to see the power and also the limitations of XSLT. The declarative nature of XSLT unveiled to suit the task very well.

The design choice to transform the LBAP request into an XSL stylesheet increased the complexity. Debugging of the written code was a challenge that was sometimes bigger than the logic that was to be implemented. The consequence is that an extension of the prototype is very complicated and not recommended. A solution with just one stylesheet that operates on the request document and the infoset document might be more transparent.

We showed that the prototype can be used to successfully query a linkbase. The reduction of the resulting linkbase to relevant linking elements can comfortably be controlled through the request. The limitations of the prototype have been reduced as far as necessary to show all concepts involved in a LBAP getLinkbase operation.
The testing of the prototype was only possible in a limited scope. First of all we could not increase the size of a linkbase and emulate a scenario like the one described in section 3.3. This simply because XSLT is too slow to handle big amounts of data. Further more, it is hard to find XLink compliant linkbases. It might have reduced the representativeness of the tests.

The next step for a prototype would be to implement the missing facets of a LBAP getLinkbase operation. Namely the interaction with the underlying protocol and the cooperation of a client and server.

Chapter 9

Future Work

The presented work covered two dimensions: The role of the XLink linking model and linkbase access in the web infrastructure in general, and the specification of LBAP as a particular piece of this environment. Accordingly, future activities are necessary in both dimensions.

9.1 Linking on the Web

In spite of the benefits of XLink (on which the majority of involved people agrees) it was quiet around the subject since it received W3C recommendation status in June 2001. We identified three issues that need to be developed further in the future:

- *Linking model specification*
  The XLink linking model is outlined in the recommendation in prose. Because of its important role, it should however be formally defined. This should be done in form of a specification of XLink infoset contributions. The specification is not only vital for LBAP, it is also needed for other XLink related activities like presentation.

- *Link presentation*
  The convincing enhancements over the HTML linking model can only be brought to bear, if the user-agents support their presentation.

- *Investigate XLink usage*
  Because of its restricted scope and the limitation for extensions XLink markup will not be the favored technology to represent relationship between resources. A known example is the W3C Annotea project that decided for RDF to describe annotations. It is important to understand, that XLink markup is the media to transport linking information on the web, but the modelling of it will mostly be done with other technologies.
  The key is however not the XLink markup but the XLink linking model. The relationship between resources needs to comply to this model in order to make it usable for linkbases. The impact of the linking model in other technologies should be addressed by research in the future. An interesting task would for instance be to work out the inverse of *Harvesting RDF Statements from XLinks* [30]. The relationship of the data model in RDBMSes to the linking model is another subject that could result in valuable insight.

9.2 LBAP

The LBAP *getLinkbase* operation is from a functionality point of view complete. It should not be fundamentally extended anymore. The work showed that there are operations related

to linkbase access. For instance the requesting of metadata from a linkbase, or the storage of request properties on the client side. It should be investigated whether such services should be included natively in the protocol.

The current development in the web services area will have an impact on the web infrastructure. It should be investigated how LBAP fits into this development. It must be ensured that it is compatible with such related technology.

The degree of precision in the specification is still very low. It should be increased by specifying the details like for instance the reaction on errors of the underlying protocol.

The prototype should be extended to cover all phases of a LBAP request-response operation. To get representative results, the prototype should be rewritten with performance and stability in mind.

9.3 Public Discussion

The probably most important future activity is to trigger a public discussion about XLink and linkbase access. It is the way to find out why the XLink area is so quiet at the moment and to estimate the chance of XLink in the future of the web.

Chapter 10

Conclusions

In this work we presented the vision of *Open Hypermedia* on the World Wide Web. We motivated the vision with scenarios, analyzed the concepts behind it, showed which pieces are still missing, and how the picture can be completed. In particular we presented a solution for linkbase access in form of *LBAP*, the Linkbase Access Protocol.

We believe that linkbases will become very popular with Web users, and that they must be a part of the Web infrastructure in the future. However, the development of linkbase servers and clients by the key players in the software market can only be expected if the communication is defined by a standardized, broadly accepted protocol.

We proposed an approach to linkbase access which is different from the ones found in today's *Open Hypermedia Systems*: The foundation and common denominator of the protocol must be the linking model and not the set of possible features that a linkbase might provide. The advantage of the approach is that on the one hand it will be possible to quickly establish a broad agreement on the protocol because of its clear focus, and on the other hand the complexity of the protocol stays low and makes it usable.

The development of LBAP was guided by a clearly defined, and carefully worked out list of requirements. The resulting specification is therefore focussed to the problem it is supposed to solve. The cornerstones of LBAP are it's delimitation (but with full coverage) to the linking model, the simplicity of the binding to the underlying protocol and it's support for future extensions.

To verify the protocol specification, a prototype was implemented. It proved that an XLink linkbase can successfully be accessed and queried for relevant information through LBAP.

In order to realize *Open Hypermedia* on the web, many barriers need to be taken in the future. We think that this work is a step into the direction of better understanding the concept of linkbase access. We hope that it will be a base for discussions in the web community about how to pave the way to a better, linkbase-enabled, web infrastructure.

Appendix A

Document type definitions

A.1 XLink Linkbase DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document: http://www.stillhard.net/2002/xlink/xlinksample.dtd

    This DTD defines the required elements for linkbases

-->
<!ELEMENT linkbase (extended*)>
<!ATTLIST linkbase
    xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
>
<!ELEMENT simple ANY>
<!ATTLIST simple
    xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
    xlink:type (simple) #FIXED "simple"
    xlink:href CDATA #IMPLIED
    xlink:role CDATA #IMPLIED
    xlink:arcrole CDATA #IMPLIED
    xlink:title CDATA #IMPLIED
    xlink:show (new | replace | embed | other | none) #IMPLIED
    xlink:actuate (onLoad | onRequest | other | none) #IMPLIED
>
<!ELEMENT extended ((title | resource | locator | arc)*)>
<!ATTLIST extended
    xlink:type (extended) #FIXED "extended"
    xlink:role CDATA #IMPLIED
    xlink:title CDATA #IMPLIED
>
<!ELEMENT title ANY>
<!-- xml:lang is not required, but provides much of the motivation
      for title elements in addition to attributes, and so is provided
      here for convenience -->
<!ATTLIST title
    xlink:type (title) #FIXED "title"
    xml:lang CDATA #IMPLIED
>
<!ELEMENT resource ANY>
<!ATTLIST resource
    xlink:type (resource) #FIXED "resource"
    xlink:role CDATA #IMPLIED
    xlink:title CDATA #IMPLIED
    xlink:label NMTOKEN #IMPLIED
>
<!ELEMENT locator (title*)>
<!-- label is not required, but locators have no particular XLink
     function if they are not labeled -->
<!ATTLIST locator
    xlink:type (locator) #FIXED "locator"
    xlink:href CDATA #REQUIRED
    xlink:role CDATA #IMPLIED
```

```
    xlink:title CDATA #IMPLIED
    xlink:label NMTOKEN #IMPLIED
>
<!ELEMENT arc (title*)>
<!-- from and to have default behavior when values are missing -->
<!ATTLIST arc
    xlink:type (arc) #FIXED "arc"
    xlink:arcrole CDATA #IMPLIED
    xlink:title CDATA #IMPLIED
    xlink:show (new | replace | embed | other | none) #IMPLIED
    xlink:actuate (onLoad | onRequest | other | none) #IMPLIED
    xlink:from NMTOKEN #IMPLIED
    xlink:to NMTOKEN #IMPLIED
>
```

A.2 XLink Information Set Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document: http://www.stillhard.net/2002/xlink/infoset.xsd

    Schema for the XLink Infoset

-->
<xsd:schema
    targetNamespace="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:link="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name="linkSet" type="link:linkSetType"/>
    <xsd:complexType name="linkSetType">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="link:link"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="link" type="link:linkType"/>
    <xsd:complexType name="linkType">
        <xsd:sequence>
            <xsd:element ref="link:participant" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="link:arc" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="title"/>
                <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="type">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="simple"/>
                    <xsd:enumeration value="extended"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="role" type="xsd:anyURI"/>
        <xsd:attribute name="title" type="xsd:string"/>
        <xsd:anyAttribute namespace="##other" processContents="strict"/>
    </xsd:complexType>
    <xsd:element name="arc" type="link:arcType"/>
    <xsd:complexType name="arcType">
        <xsd:sequence>
            <xsd:element ref="link:startParticipant"/>
            <xsd:element ref="link:endParticipant"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="link:title"/>
                <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="role" type="xsd:anyURI"/>
        <xsd:attribute name="title" type="xsd:string"/>
```

```
        <xsd:attribute name="show">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="new"/>
                    <xsd:enumeration value="replace"/>
                    <xsd:enumeration value="embed"/>
                    <xsd:enumeration value="other"/>
                    <xsd:enumeration value="none"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="actuate">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="onLoad"/>
                    <xsd:enumeration value="onRequest"/>
                    <xsd:enumeration value="other"/>
                    <xsd:enumeration value="none"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="target-stylesheet" type="xsd:anyURI"/>
        <xsd:attribute name="target-processing-context" type="xsd:anyURI"/>
        <xsd:attribute name="target-presentation-context" type="xsd:anyURI"/>
        <xsd:anyAttribute namespace="##any" processContents="strict"/>
    </xsd:complexType>
    <xsd:complexType name="participantType">
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="link:title"/>
            <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:choice>
        <xsd:attribute name="role" type="xsd:anyURI"/>
        <xsd:attribute name="title" type="xsd:string"/>
        <xsd:attribute name="resource" type="xsd:string"/>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
        <xsd:attribute name="traversalDefined"
                       type="xsd:boolean" use="required"/>
        <xsd:anyAttribute namespace="##any" processContents="strict"/>
    </xsd:complexType>
    <xsd:element name="participant">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="link:participantType"/>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="startParticipant">
        <xsd:complexType>
            <xsd:attribute name="idref" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:IDREF"/>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="endParticipant">
        <xsd:complexType>
            <xsd:attribute name="idref" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:IDREF"/>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="title">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:any namespace="##any" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="type" fixed="title"/>
            <xsd:attribute name="lang" type="xsd:string" use="required"/>
```

```
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

A.3 Linkbase Access Service Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document: http://www.stillhard.net/2002/lbap/lbaptypes.xsd

    This schema provides the definition for the datatypes used in LBAP.

    Changes:

    Notes:

-->
<xsd:schema
    targetNamespace="http://www.stillhard.net/2002/lbap"
    xmlns:lbap="http://www.stillhard.net/2002/lbap"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:annotation>
        <xsd:documentation>
            This schema provides the definition for the datatypes used in LBAP.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType name="LinkbaseRequestType">
        <xsd:annotation>
            <xsd:documentation>
            The base type for the input of a getLinkbase operation
        </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="lbap:ProcessingProperties" minOccurs="0"/>
            <xsd:element ref="lbap:SizeLimit" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="lbap:ExtensionDelivery" minOccurs="0"/>
            <xsd:element ref="lbap:LinkFilter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>
                        This element makes the extension of a request possible
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="LinkbaseRequest"
                 type="lbap:LinkbaseRequestType"/>
    <xsd:complexType name="ProcessingPropertiesType">
        <xsd:attribute name="timeLimit"
                       type="xsd:int" use="optional" default="0"/>
        <xsd:attribute name="noLocalResources"
                       type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="noTraversal"
                       type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="lbChainLength"
                       type="xsd:int" use="optional" default="0"/>
        <xsd:attribute name="deliverIfSizeLimitExceeded"
                       type="xsd:boolean" use="optional" default="no"/>
    </xsd:complexType>
    <xsd:element name="ProcessingProperties" type="lbap:ProcessingPropertiesType"/>
    <xsd:complexType name="SizeLimitType">
        <xsd:attribute name="entity" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="links"/>
                    <xsd:enumeration value="arcsPerLink"/>
                    <xsd:enumeration value="linkParticipantsPerLink"/>
                    <xsd:enumeration value="fanout"/>
```

```
                <xsd:enumeration value="fanin"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="limit" type="xsd:int" use="required"/>
</xsd:complexType>
<xsd:element name="SizeLimit" type="lbap:SizeLimitType"/>
<xsd:complexType name="ExtensionDeliveryType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="lbap:IncludeNamespace"/>
    </xsd:sequence>
    <xsd:attribute name="mode" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="xLinkOnly"/>
                <xsd:enumeration value="attributesOnly"/>
                <xsd:enumeration value="elementsOnly"/>
                <xsd:enumeration value="all"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="ExtensionDelivery" type="lbap:ExtensionDeliveryType"/>
<xsd:complexType name="LinkFilterType">
    <xsd:sequence>
        <xsd:element ref="lbap:linkPropertyAssertion" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="lbap:linkParticipantFilter"/>
            <xsd:element ref="lbap:arcFilter"/>
            <xsd:element ref="lbap:titleElementFilter"/>
            <xsd:element ref="lbap:otherElementFilter"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="LinkFilter" type="lbap:LinkFilterType"/>
<xsd:complexType name="LinkPropertyAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:andLPA"/>
        <xsd:element ref="lbap:orLPA"/>
        <xsd:element ref="lbap:notLPA"/>
        <xsd:element ref="lbap:titleAssertion"/>
        <xsd:element ref="lbap:roleAssertion"/>
        <xsd:element ref="lbap:otherPropertyAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="linkPropertyAssertion"
             type="lbap:LinkPropertyAssertionType"/>
<xsd:complexType name="SetOfLPAType">
    <xsd:sequence maxOccurs="2">
        <xsd:element ref="lbap:linkPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AndLPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfLPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="andLPA" type="lbap:AndLPAType"/>
<xsd:complexType name="OrLPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfLPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="orLPA" type="lbap:OrLPAType"/>
<xsd:complexType name="NotLPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:linkPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="notLPA" type="lbap:NotLPAType"/>
<xsd:complexType name="ArcFilterType">
```

```
    <xsd:sequence>
        <xsd:element ref="lbap:arcPropertyAssertion" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="lbap:startingParticipantFilter"/>
            <xsd:element ref="lbap:endingParticipantFilter"/>
            <xsd:element ref="lbap:titleElementFilter"/>
            <xsd:element ref="lbap:otherElementFilter"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="arcFilter" type="lbap:ArcFilterType"/>
<xsd:complexType name="ArcPropertyAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:andAPA"/>
        <xsd:element ref="lbap:orAPA"/>
        <xsd:element ref="lbap:notAPA"/>
        <xsd:element ref="lbap:titleAssertion"/>
        <xsd:element ref="lbap:arcRoleAssertion"/>
        <xsd:element ref="lbap:showAssertion"/>
        <xsd:element ref="lbap:actuateAssertion"/>
        <xsd:element ref="lbap:otherPropertyAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="arcPropertyAssertion"
             type="lbap:ArcPropertyAssertionType"/>
<xsd:complexType name="SetOfAPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:arcPropertyAssertion" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AndAPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfAPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="andAPA" type="lbap:AndAPAType"/>
<xsd:complexType name="OrAPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfAPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="orAPA" type="lbap:OrAPAType"/>
<xsd:complexType name="NotAPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:arcPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="notAPA" type="lbap:NotAPAType"/>
<xsd:complexType name="ShowAssertionType">
    <xsd:attribute name="behaviour" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="new"/>
                <xsd:enumeration value="replace"/>
                <xsd:enumeration value="other"/>
                <xsd:enumeration value="none"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="showAssertion" type="lbap:ShowAssertionType"/>
<xsd:complexType name="ActuateAssertionType">
    <xsd:attribute name="behaviour" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="onLoad"/>
                <xsd:enumeration value="onRequest"/>
                <xsd:enumeration value="other"/>
                <xsd:enumeration value="none"/>
            </xsd:restriction>
        </xsd:simpleType>
```

```
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="actuateAssertion"
             type="lbap:ActuateAssertionType"/>
<xsd:complexType name="ParticipantFilterType">
    <xsd:sequence>
        <xsd:element ref="lbap:participantPropertyAssertion" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="lbap:titleElementFilter"/>
            <xsd:element ref="lbap:otherElementFilter"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="startingParticipantFilter"
             type="lbap:ParticipantFilterType"/>
<xsd:element name="endingParticipantFilter"
             type="lbap:ParticipantFilterType"/>
<xsd:element name="linkParticipantFilter"
             type="lbap:ParticipantFilterType"/>
<xsd:complexType name="ParticipantPropertyAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:andPPA"/>
        <xsd:element ref="lbap:orPPA"/>
        <xsd:element ref="lbap:notPPA"/>
        <xsd:element ref="lbap:titleAssertion"/>
        <xsd:element ref="lbap:roleAssertion"/>
        <xsd:element ref="lbap:resourceAssertion"/>
        <xsd:element ref="lbap:otherPropertyAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="participantPropertyAssertion"
             type="lbap:ParticipantPropertyAssertionType"/>
<xsd:complexType name="SetOfPPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:participantPropertyAssertion" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AndPPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfPPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="andPPA" type="lbap:AndPPAType"/>
<xsd:complexType name="OrPPAType">
    <xsd:complexContent>
        <xsd:extension base="lbap:SetOfPPAType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="orPPA" type="lbap:OrPPAType"/>
<xsd:complexType name="NotPPAType">
    <xsd:sequence>
        <xsd:element ref="lbap:participantPropertyAssertion"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="notPPA" type="lbap:NotPPAType"/>
<xsd:complexType name="URIAssertionType">
    <xsd:complexContent>
        <xsd:extension base="lbap:ValueAssertionType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="uriAssertion" type="lbap:URIAssertionType"/>
<xsd:element name="roleAssertion" type="lbap:URIAssertionType"/>
<xsd:element name="arcRoleAssertion" type="lbap:URIAssertionType"/>
<xsd:element name="resourceAssertion" type="lbap:URIAssertionType"/>
<xsd:complexType name="ValueAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:equalityMatch"/>
        <xsd:element ref="lbap:substringAssertion"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="valueAssertion" type="lbap:ValueAssertionType"/>
```

```
<xsd:element name="titleAssertion" type="lbap:ValueAssertionType"/>
<xsd:complexType name="SubstringAssertionType">
    <xsd:choice>
        <xsd:element ref="lbap:startsWith"/>
        <xsd:element ref="lbap:contains"/>
        <xsd:element ref="lbap:endsWith"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="substringAssertion" type="lbap:SubstringAssertionType"/>
<xsd:complexType name="OtherElementFilterType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
</xsd:complexType>
<xsd:element name="otherElementFilter" type="lbap:OtherElementFilterType"/>
<xsd:element name="titleElementFilter" type="lbap:OtherElementFilterType"/>
<xsd:complexType name="OtherPropertyAssertionType">
    <xsd:sequence>
        <xsd:element ref="lbap:valueAssertion"/>
    </xsd:sequence>
    <xsd:attribute name="attributeName" type="xsd:NMTOKEN" use="required"/>
    <xsd:attribute name="namespace" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<xsd:element name="otherPropertyAssertion" type="lbap:OtherPropertyAssertionType"/>
<xsd:simpleType name="equalityMatchType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="equalityMatch" type="lbap:equalityMatchType"/>
<xsd:simpleType name="startsWithType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="startsWith" type="lbap:startsWithType"/>
<xsd:simpleType name="containsType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="contains" type="lbap:containsType"/>
<xsd:simpleType name="endsWithType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:element name="endsWith" type="lbap:endsWithType"/>
<xsd:complexType name="LinkbaseResponseType">
    <xsd:annotation>
        <xsd:documentation>
            The type for the output of a getLinkbase operation
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="lbap:error" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="lbap:linkBase"/>
    </xsd:sequence>
    <xsd:attribute name="success" type="xsd:boolean" use="required" default="true"/>
</xsd:complexType>
<xsd:element name="linkbaseResponse" type="lbap:LinkbaseResponseType"/>
<xsd:complexType name="ErrorType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
            <xsd:attribute name="class" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:NMTOKEN">
                        <xsd:enumeration value="protocol"/>
                        <xsd:enumeration value="operation"/>
                        <xsd:enumeration value="extension"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="code" type="xsd:int" use="optional"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="error" type="lbap:ErrorType"/>
```

```
    <xsd:element name="timeLimitExceeded">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:restriction base="lbap:ErrorType">
                    <xsd:attribute name="class" fixed="operation"/>
                    <xsd:attribute name="code" fixed="510"/>
                    <xsd:attribute name="name" fixed="timeLimitExceededError"/>
                </xsd:restriction>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="LinkBaseType">
        <xsd:sequence>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="linkBase" type="lbap:LinkBaseType"/>
    <xsd:simpleType name="IncludeNamespaceType">
        <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <xsd:element name="IncludeNamespace"
                 type="lbap:IncludeNamespaceType"/>
</xsd:schema>
```

Appendix B

Sample Documents

B.1 Family Linkbase

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE linkbase SYSTEM "http://www.stillhard.net/2002/xlink/xlinksample.dtd">

<linkbase xmlns:xlink="http://www.w3.org/1999/xlink">

    <extended xlink:title="Family Miller"
        xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/family">

        <!-- peoples homepages -->
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/tanja.xml"
            xlink:label="mother" xlink:title="Tanja"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/mother"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/tom.xml"
            xlink:label="father" xlink:title="Tom"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/father"/>
        <locator
            xlink:href="http://www.bruce.miller.com/me.xml"
            xlink:label="uncle1" xlink:title="Bruce"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/uncle"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml"
            xlink:label="daugther" xlink:title="Jane"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/daughter"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/james.xml"
            xlink:label="son" xlink:title="James"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/john.xml"
            xlink:label="son" xlink:title="John"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>

        <!-- starting resources, include the kids pages -->
        <!-- occurrences of 'Jane' -->
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                        #xpointer(string-range(//*,'Jane'))"
            xlink:label="daughterocc" xlink:title="Jane"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                        #xpointer(string-range(//*,'Jane'))"
            xlink:label="daughterocc" xlink:title="Jane"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                        #xpointer(string-range(//*,'Jane'))"
```

```
            xlink:label="daughterocc" xlink:title="Jane"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"/>

        <!-- occurrences of 'James' -->
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                        #xpointer(string-range(//*,'James'))"
            xlink:label="sonocc" xlink:title="James"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                        #xpointer(string-range(//*,'James'))"
            xlink:label="sonocc" xlink:title="James"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                        #xpointer(string-range(//*,'James'))"
            xlink:label="sonocc" xlink:title="James"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>

        <!-- occurrences of 'John' -->
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                        #xpointer(string-range(//*,'John'))"
            xlink:label="sonocc" xlink:title="John"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                        #xpointer(string-range(//*,'John'))"
            xlink:label="sonocc" xlink:title="John"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>
        <locator
            xlink:href="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                        #xpointer(string-range(//*,'John'))"
            xlink:label="sonocc" xlink:title="John"
            xlink:role="http://www.stillhard.net/2002/xlink/samples/roles/son"/>

        <!-- traversal -->
        <arc xlink:title="Father"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
            xlink:from="daughterocc" xlink:to="father"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Father"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
            xlink:from="sonocc" xlink:to="father"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Mother"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
            xlink:from="daughterocc" xlink:to="mother"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Mother"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
            xlink:from="sonocc" xlink:to="mother"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Brother"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
            xlink:from="daughterocc" xlink:to="son"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Uncle"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
            xlink:from="daughterocc" xlink:to="uncle1"
            xlink:show="replace" xlink:actuate="onRequest"/>
        <arc xlink:title="Uncle"
            xlink:arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
            xlink:from="sonocc" xlink:to="uncle1"
            xlink:show="replace" xlink:actuate="onRequest"/>

    </extended>
</linkbase>
```

B.2 Family Linkbase Information Set

```
<?xml version="1.0" encoding="utf-8"?>
<linkSet
    xmlns="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.stillhard.net/2002/xlink/infoset
                        http://www.stillhard.net/2002/xlink/infoset/xlinkinfoset.xsd">
    <link type="extended"
          title="Family Miller"
          role="http://www.stillhard.net/2002/xlink/samples/roles/family"
          elem="extended">
        <participant title="Tanja"
            role="http://www.stillhard.net/2002/xlink/samples/roles/mother"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/tanja.xml"
            participantType="locator" elem="locator" label="mother" id="l1"/>
        <participant title="Tom"
            role="http://www.stillhard.net/2002/xlink/samples/roles/father"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/tom.xml"
            participantType="locator" elem="locator" label="father" id="l2"/>
        <participant title="Bruce"
            role="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
            resource="http://www.bruce.miller.com/me.xml"
            participantType="locator" elem="locator" label="uncle1" id="l3"/>
        <participant title="Jane"
            role="http://www.stillhard.net/2002/xlink/samples/roles/daughter"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml"
            participantType="locator" elem="locator" label="daugther" id="l4"/>
        <participant title="James"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/james.xml"
            participantType="locator" elem="locator" label="son" id="l5"/>
        <participant title="John"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/john.xml"
            participantType="locator" elem="locator" label="son" id="l6"/>
        <participant title="Jane"
            role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                      #xpointer(string-range(//*,'Jane'))"
            participantType="locator" elem="locator" label="daughterocc" id="l7"/>
        <participant title="Jane"
            role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                      #xpointer(string-range(//*,'Jane'))"
            participantType="locator" elem="locator" label="daughterocc" id="l8"/>
        <participant title="Jane"
            role="http://www.stillhard.net/2002/xlink/samples/roles/daugther"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                      #xpointer(string-range(//*,'Jane'))"
            participantType="locator" elem="locator" label="daughterocc" id="l9"/>
        <participant title="James"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                      #xpointer(string-range(//*,'James'))"
            participantType="locator" elem="locator" label="sonocc" id="l10"/>
        <participant title="James"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                      #xpointer(string-range(//*,'James'))"
            participantType="locator" elem="locator" label="sonocc" id="l11"/>
        <participant title="James"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
            resource="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                      #xpointer(string-range(//*,'James'))"
            participantType="locator" elem="locator" label="sonocc" id="l12"/>
        <participant title="John"
            role="http://www.stillhard.net/2002/xlink/samples/roles/son"
```

```
    resource="http://www.stillhard.net/2002/xlink/samples/resources/john.xml
                #xpointer(string-range(//*,'John'))"
    participantType="locator" elem="locator" label="sonocc" id="l13"/>
<participant title="John"
    role="http://www.stillhard.net/2002/xlink/samples/roles/son"
    resource="http://www.stillhard.net/2002/xlink/samples/resources/jane.xml
                #xpointer(string-range(//*,'John'))"
    participantType="locator" elem="locator" label="sonocc" id="l14"/>
<participant title="John"
    role="http://www.stillhard.net/2002/xlink/samples/roles/son"
    resource="http://www.stillhard.net/2002/xlink/samples/resources/james.xml
                #xpointer(string-range(//*,'John'))"
    participantType="locator" elem="locator" label="sonocc" id="l15"/>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l7"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l8"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l9"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l10"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l11"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l12"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l13"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l14"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Father" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/father"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l15"/>
    <endParticipant idref="l2"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l7"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l8"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l9"/>
```

```
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l10"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l11"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l12"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l13"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l14"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Mother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/mother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l15"/>
    <endParticipant idref="l1"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l7"/>
    <endParticipant idref="l5"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l7"/>
    <endParticipant idref="l6"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l8"/>
    <endParticipant idref="l5"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l8"/>
    <endParticipant idref="l6"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l9"/>
    <endParticipant idref="l5"/>
</arc>
<arc title="Brother" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/brother"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l9"/>
    <endParticipant idref="l6"/>
</arc>
<arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l7"/>
    <endParticipant idref="l3"/>
</arc>
<arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
     show="replace" actuate="onRequest" elem="arc">
    <startParticipant idref="l8"/>
    <endParticipant idref="l3"/>
```

```
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l9"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l10"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l11"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l12"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l13"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l14"/>
            <endParticipant idref="l3"/>
        </arc>
        <arc title="Uncle" arcrole="http://www.stillhard.net/2002/xlink/samples/roles/uncle"
             show="replace" actuate="onRequest" elem="arc">
            <startParticipant idref="l15"/>
            <endParticipant idref="l3"/>
        </arc>
    </link>
</linkSet>
```

Appendix C

LBAP Prototype

C.1 Request Processor Generator Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Note:
        - startsWith and endsWith are mapped to contains (lack of functionality in the XSLT processor)
        - only the arcsPerLink size limit is supported
-->

<xsl:stylesheet version="1.1"
    xmlns:lbap="http://www.stillhard.net/2002/lbap"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">

        <xsl:text disable-output-escaping="yes">&lt;xsl:stylesheet version="1.1"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset"
                xmlns="http://www.w3.org/1999/XSL/Transform"&gt;</xsl:text>

            <xsl:element name="xsl:include">
                <xsl:attribute name="href">
                    http://www.stillhard.net/2002/lbap/request_processor_general.xsl
                </xsl:attribute>
            </xsl:element>

            <xsl:call-template name="initVariables"/>

            <!-- apply the link filter and store it in a variable.
                 It will generate the propertyAssertions -->
            <xsl:variable name="assertions">
                <xsl:apply-templates select="//lbap:LinkFilter"/>
            </xsl:variable>

            <!-- output the assertions -->
            <xsl:copy-of select="$assertions"/>

            <!-- default assertions that have not been generated (because they
                 were not specified in the request -->
            <xsl:if test=" count( $assertions/xsl:template[@name='arcPropertyAssertion'] ) = 0">
                <!-- no arcPropertyAssertion, add default assertion (always true) -->
                <xsl:element name="xsl:template">
                    <xsl:attribute name="name">arcPropertyAssertion</xsl:attribute>1
                </xsl:element>
            </xsl:if>
            <xsl:if test=" count( $assertions/xsl:template[@name='linkPropertyAssertion'] ) = 0">
                <!-- no linkPropertyAssertion, add default assertion (always true) -->
                <xsl:element name="xsl:template">
                    <xsl:attribute name="name">linkPropertyAssertion</xsl:attribute>1
                </xsl:element>
            </xsl:if>
```

```
        <xsl:if test=" count( $assertions/xsl:template[@name='startParticipantPropertyAssertion'] ) = 0">
            <!-- no startParticipantPropertyAssertion, add default assertion (always true) -->
            <xsl:element name="xsl:template">
                <xsl:attribute name="name">startParticipantPropertyAssertion</xsl:attribute>1
            </xsl:element>
        </xsl:if>
        <xsl:if test=" count( $assertions/xsl:template[@name='endParticipantPropertyAssertion'] ) = 0">
            <!-- no endParticipantPropertyAssertion, add default assertion (always true) -->
            <xsl:element name="xsl:template">
                <xsl:attribute name="name">endParticipantPropertyAssertion</xsl:attribute>1
            </xsl:element>
        </xsl:if>

    <xsl:text disable-output-escaping="yes">&lt;/xsl:stylesheet&gt;</xsl:text>

</xsl:template>

<xsl:template match="lbap:LinkbaseRequest">

        <xsl:apply-templates/>

</xsl:template>

<xsl:template name="initVariables">

    <!-- get the size limit variables -->
    <xsl:variable name="vars">
        <xsl:call-template name="sizeLimit"/>
    </xsl:variable>

    <!-- if a certain size limit was specified in the request, then output it,
         otherwise add a variable with the default value -->
    <xsl:choose>
        <xsl:when test="count( $vars/xsl:variable[@name='arcLimit'] ) = 0">
            <xsl:element name="xsl:variable">
                <xsl:attribute name="name">arcLimit</xsl:attribute>99999999
            </xsl:element>
        </xsl:when>
        <xsl:otherwise>
            <xsl:copy-of select="$vars/xsl:variable[@name='arcLimit']"/>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:choose>
        <xsl:when test="count( $vars/xsl:variable[@name='linkLimit'] ) = 0">
            <xsl:element name="xsl:variable">
                <xsl:attribute name="name">linkLimit</xsl:attribute>99999999
            </xsl:element>
        </xsl:when>
        <xsl:otherwise>
            <xsl:copy-of select="$vars/xsl:variable[@name='linkLimit']"/>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:choose>
        <xsl:when test="count( $vars/xsl:variable[@name='participantLimit'] ) = 0">
            <xsl:element name="xsl:variable">
                <xsl:attribute name="name">participantLimit</xsl:attribute>99999999
            </xsl:element>
        </xsl:when>
        <xsl:otherwise>
            <xsl:copy-of select="$vars/xsl:variable[@name='participantLimit']"/>
        </xsl:otherwise>
    </xsl:choose>

</xsl:template>

<xsl:template name="sizeLimit">
    <xsl:for-each select="//lbap:SizeLimit">
        <xsl:choose>
            <xsl:when test="@entity='arcsPerLink'">
                <xsl:element name="xsl:variable">
                    <xsl:attribute name="name">arcLimit</xsl:attribute>
```

```
                    <xsl:value-of select="@limit"/>
                </xsl:element>
            </xsl:when>
            <xsl:when test="@entity='links'">
                <xsl:element name="xsl:variable">
                    <xsl:attribute name="name">linkLimit</xsl:attribute>
                    <xsl:value-of select="@limit"/>
                </xsl:element>
            </xsl:when>
            <xsl:when test="@entity='linkParticipantsPerLink'">
                <xsl:element name="xsl:variable">
                    <xsl:attribute name="name">participantLimit</xsl:attribute>
                    <xsl:value-of select="@limit"/>
                </xsl:element>
            </xsl:when>
            <!-- here we would initialize the oder sizeLimit variables (fanin and fanout) -->
        </xsl:choose>
    </xsl:for-each>
</xsl:template>

<xsl:template match="lbap:LinkFilter">
    <!-- the filter template is included from request_processor_general.xsl -->
    <!-- create the property assertion template -->
    <xsl:element name="xsl:template">
        <xsl:attribute name="name">linkPropertyAssertion</xsl:attribute>
        <!-- output the content of the linkPropertyAssertion template -->
        <xsl:apply-templates select="lbap:linkPropertyAssertion"/>

        <!-- default linkPropertyAssertion in case there is no assertion in the request
            it needs to exist because it is called anyway -->
        <xsl:variable name="assertion">
            <xsl:apply-templates select="lbap:linkPropertyAssertion"/>
        </xsl:variable>
        <xsl:if test="$assertion = ''">1</xsl:if>

    </xsl:element>

    <xsl:apply-templates select="lbap:arcFilter"/>

</xsl:template>

<xsl:template match="lbap:linkPropertyAssertion">
    <!-- nothing special to do here (yet) -->
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="lbap:arcFilter">
    <!-- the filter template is included from request_processor_general.xsl -->
    <!-- create the property assertion template -->
    <xsl:element name="xsl:template">
        <xsl:attribute name="name">arcPropertyAssertion</xsl:attribute>

        <!-- create the arcPropertyAssertion template -->
        <xsl:variable name="assertion">
            <xsl:apply-templates select="lbap:arcPropertyAssertion"/>
        </xsl:variable>

        <!-- output it -->
        <xsl:copy-of select="$assertion"/>

        <!-- default arcPropertyAssertion in case there is no assertion in the request
            it needs to exist because it is called anyway -->
        <xsl:if test="$assertion = ''">1</xsl:if>

    </xsl:element>

    <xsl:apply-templates select="lbap:startingParticipantFilter"/>
    <xsl:apply-templates select="lbap:endingParticipantFilter"/>

</xsl:template>
```

```
<xsl:template match="lbap:arcPropertyAssertion">
    <!-- nothing special to do here (yet) -->
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="lbap:startingParticipantFilter">
    <!-- the filter template is included from request_processor_general.xsl -->
    <!-- create the property assertion template -->
    <xsl:element name="xsl:template">
        <xsl:attribute name="name">startParticipantPropertyAssertion</xsl:attribute>

        <!-- create the startParticipantPropertyAssertion template -->
        <xsl:variable name="assertion">
            <xsl:apply-templates select="lbap:participantPropertyAssertion"/>
        </xsl:variable>

        <!-- output it -->
        <xsl:copy-of select="$assertion"/>

        <!-- default endParticipantPropertyAssertion in case there is no assertion in the request
            it needs to exist because it is called anyway -->
        <xsl:if test="$assertion = ''">1</xsl:if>

    </xsl:element>
</xsl:template>

<xsl:template match="lbap:endingParticipantFilter">
    <!-- the filter template is included from request_processor_general.xsl -->
    <!-- create the property assertion template -->
    <xsl:element name="xsl:template">
        <xsl:attribute name="name">endParticipantPropertyAssertion</xsl:attribute>

        <!-- create the endParticipantPropertyAssertion template -->
        <xsl:variable name="assertion">
            <xsl:apply-templates select="lbap:participantPropertyAssertion"/>
        </xsl:variable>

        <!-- output it -->
        <xsl:copy-of select="$assertion"/>

        <!-- default endParticipantPropertyAssertion in case there is no assertion in the request
            it needs to exist because it is called anyway -->
        <xsl:if test="$assertion = ''">1</xsl:if>

    </xsl:element>
</xsl:template>

<xsl:template match="lbap:participantPropertyAssertion">
    <!-- to make the assertions on the participants we have to select them first.
        Create a variable that will contain the calculated idref and select the participant
        with an id that equals the idref variable. -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">idref</xsl:attribute>
        <xsl:element name="xsl:value-of">
            <xsl:attribute name="select">@infoset:idref</xsl:attribute>
        </xsl:element>
    </xsl:element>
    <xsl:element name="for-each">
        <xsl:attribute name="select">//infoset:participant[@infoset:id = $idref]</xsl:attribute>
        <!-- no create the assertions on the participant -->
        <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:andLPA">
    <!-- logic and between linkPropertyAssertions -->
    <!-- first output variables that will be filled with a value if the operands are true -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vAnd<xsl:number level="any"/>1</xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>
```

```
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vAnd<xsl:number level="any"/>2</xsl:attribute>
            <xsl:apply-templates select="*[2]"/>
        </xsl:element>

        <!-- now output the comparison of the operands -->
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">
                $vAnd<xsl:number level="any"/>1 != '' and $vAnd<xsl:number level="any"/>2 != ''
            </xsl:attribute>1
        </xsl:element>
    </xsl:template>

    <xsl:template match="lbap:orLPA">
        <!-- logic or between linkPropertyAssertions -->
        <!-- first output variables that will be filled with a value if the operands are true -->
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vOr<xsl:number level="any"/>1</xsl:attribute>
            <xsl:apply-templates select="*[1]"/>
        </xsl:element>
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vOr<xsl:number level="any"/>2</xsl:attribute>
            <xsl:apply-templates select="*[2]"/>
        </xsl:element>

        <!-- now output the comparison of the operands -->
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">
                $vOr<xsl:number level="any"/>1 != '' or $vOr<xsl:number level="any"/>2 != ''
            </xsl:attribute>1
        </xsl:element>
    </xsl:template>

    <xsl:template match="lbap:notLPA">
        <!-- logic not of a arcPropertyAssertion -->
        <!-- first output a variable that will be filled with the value if the operand is true -->
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vNot<xsl:number level="any"/></xsl:attribute>
            <xsl:apply-templates select="*[1]"/>
        </xsl:element>

        <!-- now output the comparison of the operands -->
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">$vNot<xsl:number level="any"/> = ''</xsl:attribute>1
        </xsl:element>
    </xsl:template>

    <xsl:template match="lbap:andAPA">
        <!-- logic and between arcPropertyAssertions -->
        <!-- first output variables that will be filled with a value if the operands are true -->
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vAnd<xsl:number level="any"/>1</xsl:attribute>
            <xsl:apply-templates select="*[1]"/>
        </xsl:element>
        <xsl:element name="xsl:variable">
            <xsl:attribute name="name">vAnd<xsl:number level="any"/>2</xsl:attribute>
            <xsl:apply-templates select="*[2]"/>
        </xsl:element>

        <!-- now output the comparison of the operands -->
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">
                $vAnd<xsl:number level="any"/>1 != '' and $vAnd<xsl:number level="any"/>2 != ''
            </xsl:attribute>1
        </xsl:element>
    </xsl:template>

    <xsl:template match="lbap:orAPA">
        <!-- logic or between arcPropertyAssertions -->
        <!-- first output variables that will be filled with a value if the operands are true -->
        <xsl:element name="xsl:variable">
```

```
        <xsl:attribute name="name">vOr<xsl:number level="any"/>1</xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vOr<xsl:number level="any"/>2</xsl:attribute>
        <xsl:apply-templates select="*[2]"/>
    </xsl:element>

    <!-- now output the comparison of the operands -->
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">
            $vOr<xsl:number level="any"/>1 != '' or $vOr<xsl:number level="any"/>2 != ''
        </xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:notAPA">
    <!-- logic not of a arcPropertyAssertion -->
    <!-- first output a variable that will be filled with the value if the operand is true -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vNot<xsl:number level="any"/></xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>

    <!-- now output the comparison of the operands -->
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">$vNot<xsl:number level="any"/> = ''</xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:andPPA">
    <!-- logic and between participantPropertyAssertion -->
    <!-- first output variables that will be filled with a value if the operands are true -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vAnd<xsl:number level="any"/>1</xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vAnd<xsl:number level="any"/>2</xsl:attribute>
        <xsl:apply-templates select="*[2]"/>
    </xsl:element>

    <!-- now output the comparison of the operands -->
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">
            $vAnd<xsl:number level="any"/>1 != '' and $vAnd<xsl:number level="any"/>2 != ''
        </xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:orPPA">
    <!-- logic or between participantPropertyAssertion -->
    <!-- first output variables that will be filled with a value if the operands are true -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vOr<xsl:number level="any"/>1</xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vOr<xsl:number level="any"/>2</xsl:attribute>
        <xsl:apply-templates select="*[2]"/>
    </xsl:element>

    <!-- now output the comparison of the operands -->
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">
            $vOr<xsl:number level="any"/>1 != '' or $vOr<xsl:number level="any"/>2 != ''
        </xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:notPPA">
```

```
    <!-- logic not of a participantPropertyAssertion -->
    <!-- first output a variable that will be filled
         with the value if the operand is true -->
    <xsl:element name="xsl:variable">
        <xsl:attribute name="name">vNot<xsl:number level="any"/></xsl:attribute>
        <xsl:apply-templates select="*[1]"/>
    </xsl:element>

    <!-- now output the comparison of the operands -->
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">$vNot<xsl:number level="any"/> = ''</xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:titleAssertion">
    <xsl:apply-templates>
        <xsl:with-param name="property">@infoset:title</xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<xsl:template match="lbap:arcRoleAssertion">
    <xsl:apply-templates>
        <xsl:with-param name="property">@infoset:arcrole</xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<xsl:template match="lbap:roleAssertion">
    <xsl:apply-templates>
        <xsl:with-param name="property">@infoset:role</xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<xsl:template match="lbap:showAssertion">
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">
            @infoset:show = '<xsl:value-of select="@behaviour"/>'
        </xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:actuateAssertion">
    <xsl:element name="xsl:if">
        <xsl:attribute name="test">
            @infoset:actuate = '<xsl:value-of select="@behaviour"/>'
        </xsl:attribute>1
    </xsl:element>
</xsl:template>

<xsl:template match="lbap:resourceAssertion">
    <!-- Set the property for the value assertion -->
    <xsl:apply-templates>
        <xsl:with-param name="property">infoset:resource</xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<xsl:template match="lbap:valueAssertion">
    <!-- just forward the parameter -->
    <xsl:param name="property"/>
    <xsl:apply-templates select="lbap:equalityMatch | lbap:substringAssertion">
        <xsl:with-param name="property"><xsl:value-of select="$property"/></xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<xsl:template match="lbap:substringAssertion">
    <!-- just forward the parameter -->
    <xsl:param name="property"/>
    <xsl:apply-templates select="lbap:contains | lbap:startsWith | lbap:endsWith">
        <xsl:with-param name="property"><xsl:value-of select="$property"/></xsl:with-param>
    </xsl:apply-templates>
</xsl:template>
```

```
    <xsl:template match="lbap:otherPropertyAssertion">
        <!-- Set the property for the value assertion -->
        <!-- TBD support namespace assertion -->
        <xsl:apply-templates>
            <xsl:with-param name="property"><xsl:value-of select="@attributeName"/></xsl:with-param>
        </xsl:apply-templates>
    </xsl:template>

    <!-- value assertion - equalityMatch -->
    <xsl:template match="lbap:equalityMatch">
        <!-- output a test on the property that is delivered in the parameter -->
        <xsl:param name="property"/>
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">
                <xsl:value-of select="$property"/> = '<xsl:value-of select="text()"/>'
            </xsl:attribute>1
        </xsl:element>
    </xsl:template>

    <!-- value assertion - substring assertion -->
    <xsl:template match="lbap:startsWith | lbap:contains | lbap:endsWith">
        <!-- all are mapped to contains because the XSLT
             processor does not support the others -->
        <!-- output a test on the property that is delivered in the parameter -->
        <xsl:param name="property"/>
        <xsl:element name="xsl:if">
            <xsl:attribute name="test">
                contains(@<xsl:value-of select="$property"/>, '<xsl:value-of select="text()"/>')
            </xsl:attribute>1
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>
```

C.2 Request Processor General Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset">

    <xsl:variable name="relevantParticipants">
        <xsl:apply-templates select="//infoset:arc" mode="getRelevantParticipants"/>
    </xsl:variable>

    <!-- execute the filter and put into a variable -->
    <xsl:variable name="filterResult">
        <xsl:apply-templates/>
    </xsl:variable>

    <xsl:template match="/" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <!-- check Size Limits, fail, if it is exceeded -->
        <xsl:variable name="failure">
            <xsl:if test="count( $filterResult/*/*/infoset:arc ) &gt; $arcLimit">arcLimit</xsl:if>
            <xsl:if test="count( $filterResult/*/infoset:link ) &gt; $linkLimit">linkLimit</xsl:if>
            <xsl:if test="count( $filterResult/*/*/infoset:participant ) &gt;
                          $participantLimit">participantLimit</xsl:if>
        </xsl:variable>

        <!-- generate the result -->
        <xsl:if test="$failure = ''">
            <!-- Success, write out the result of the filter -->
            <xsl:copy-of select="$filterResult"/>
        </xsl:if>
        <xsl:if test="$failure != ''">
            <!-- output the empty linkSet -->
            <xsl:for-each select="//infoset:linkSet">
```

```
            <xsl:copy>
                <xsl:apply-templates select="@*"/>
            </xsl:copy>
        </xsl:for-each>
    </xsl:if>

    <!-- generate the status -->
    <xsl:document href="lbap_status.xml">
        <xsl:if test="$failure = ''">
            <!-- Success, write success status -->
            <status success="true"/>
        </xsl:if>
        <xsl:if test="$failure != ''">
            <status success="false">
                <xsl:choose>
                    <!-- Arc limit exceeded. Comment and generate error -->
                    <xsl:when test="contains( $failure, 'arcLimit')">
                        <xsl:comment>
                            Error while filtering the linkbase: arc limit exceeded
                        </xsl:comment>
                        <error type="arcLimitExceeded">
                            Error while filtering the linkbase: arc limit exceeded
                        </error>
                    </xsl:when>
                    <!-- Link  limit exceeded. Comment and generate error -->
                    <xsl:when test="contains( $failure, 'linkLimit')">
                        <xsl:comment>
                            Error while filtering the linkbase: link limit exceeded
                        </xsl:comment>
                        <error type="linkLimitExceeded">
                            Error while filtering the linkbase: link limit exceeded
                        </error>
                    </xsl:when>
                    <!-- Participant limit exceeded. Comment and generate error -->
                    <xsl:when test="contains( $failure, 'participantLimit')">
                        <xsl:comment>
                            Error while filtering the linkbase: participant limit exceeded
                        </xsl:comment>
                        <error type="participantLimitExceeded">
                            Error while filtering the linkbase: participant limit exceeded
                        </error>
                    </xsl:when>
                    <!-- Default -->
                    <xsl:otherwise>
                        <xsl:comment>
                            Unexpeced error while filtering the linkbase
                        </xsl:comment>
                        <error type="unknown">
                            Unexpeced error while filtering the linkbase:
                                <xsl:value-of select="$failure"/>
                        </error>
                    </xsl:otherwise>
                </xsl:choose>
            </status>
        </xsl:if>
    </xsl:document>

</xsl:template>

<xsl:template match="infoset:link">

    <!-- evaluate link property assertion -->
    <xsl:variable name="valid">
        <xsl:call-template name="linkPropertyAssertion"/>
    </xsl:variable>
    <xsl:if test="$valid != ''">
        <!-- link is valid -->
        <!-- filter arcs -->
        <xsl:variable name="arcs">
            <xsl:apply-templates select="infoset:arc"/>
        </xsl:variable>
```

```
        <!-- TODO: link participants for which no traversal is defined
        <xsl:variable name="participants">
            <xsl:apply-templates select="infoset:participant" mode="check"/>
            </xsl:variable>
        -->
        <!-- the link should only be output if not all the arcs are filtered out -->
        <xsl:if test="count( $arcs/infoset:arc ) &gt; 0">
            <!-- the link will be taken, so output it -->
            <xsl:copy>
                <xsl:apply-templates select="@*"/>
                <xsl:apply-templates select="infoset:participant"/>
                <xsl:copy-of select="$arcs"/>
            </xsl:copy>
        </xsl:if>
    </xsl:if>

</xsl:template>

<!-- arc filter -->
<xsl:template match="infoset:arc">
    <!-- evaluate arc property assertion -->
    <xsl:variable name="valid">
        <xsl:call-template name="arcPropertyAssertion"/>
    </xsl:variable>

    <xsl:if test="$valid != ''">
        <!-- arc is valid -->
        <!-- filter participants, store in variables -->
        <xsl:variable name="startParticipant">
            <xsl:apply-templates select="infoset:startParticipant"/>
        </xsl:variable>
        <xsl:variable name="endParticipant">
            <xsl:apply-templates select="infoset:endParticipant"/>
        </xsl:variable>

        <!-- the arc should only be output if none of the participants is filtered out -->
        <xsl:if test="count( $startParticipant/infoset:startParticipant ) &gt; 0 and
                            count( $endParticipant/infoset:endParticipant ) &gt; 0">
            <!-- the arc will be taken, so output it -->
            <xsl:copy>
                <xsl:apply-templates select="@*"/>
                <xsl:copy-of select="$startParticipant"/>
                <xsl:copy-of select="$endParticipant"/>
            </xsl:copy>
        </xsl:if>
    </xsl:if>
</xsl:template>

<!-- start and end participant filter (start and end
     are taken together to reduce the amount of code. -->
<xsl:template match="infoset:startParticipant | infoset:endParticipant">
    <!-- evaluate participant property assertion -->
    <xsl:variable name="valid">
        <xsl:if test="name() = 'infoset:startParticipant'">
            <xsl:call-template name="startParticipantPropertyAssertion"/>
        </xsl:if>

        <xsl:if test="name() = 'infoset:endParticipant'">
            <xsl:call-template name="endParticipantPropertyAssertion"/>
        </xsl:if>
    </xsl:variable>

    <xsl:if test="$valid != ''">
        <!-- particpant is valid -->
        <xsl:call-template name="copyAndApply"/>
    </xsl:if>
</xsl:template>

<!-- the templates with mode "getRelevantParticipants" are only used to output the idref attribute
    of relevant participants -->
<xsl:template match="infoset:arc" mode="getRelevantParticipants">
```

```
        <!-- evaluate arc property assertion -->
        <xsl:variable name="valid">
            <xsl:call-template name="arcPropertyAssertion"/>
        </xsl:variable>

        <xsl:if test="$valid != ''">
            <!-- arc is valid -->
            <!-- filter participants, store there ids in variables -->
            <xsl:variable name="startParticipant">
                <xsl:apply-templates select="infoset:startParticipant" mode="getRelevantParticipants"/>
            </xsl:variable>
            <xsl:variable name="endParticipant">
                <xsl:apply-templates select="infoset:endParticipant" mode="getRelevantParticipants"/>
            </xsl:variable>

            <!-- the arc should only be output if none of the participants is filtered out -->
            <xsl:if test="$startParticipant != '' and $endParticipant != ''">valid
                <xsl:value-of select="$startParticipant"/><xsl:value-of select="$endParticipant"/>
            </xsl:if>
        </xsl:if>
    </xsl:template>
    <xsl:template match="infoset:startParticipant |
                         infoset:endParticipant" mode="getRelevantParticipants">
        <!-- evaluate participant property assertion -->
        <xsl:variable name="valid">
            <xsl:if test="name() = 'infoset:startParticipant'">
                <xsl:call-template name="startParticipantPropertyAssertion"/>
            </xsl:if>
            <!-- TODO thats wrong! only valid if both participants valid -->
            <xsl:if test="name() = 'infoset:endParticipant'">
                <xsl:call-template name="endParticipantPropertyAssertion"/>
            </xsl:if>
        </xsl:variable>
        <!-- output the id of the participant, because it might be relevant.
            Note that we have to surround the id with ';' to guarantee,
            that it cannot happen that one id contains another -->
        <xsl:if test="$valid != ''">;<xsl:value-of select="@infoset:idref"/>;</xsl:if>
    </xsl:template>

    <!-- participant filter. Only output a participant, if it is
         chosen as a start or endparticipant of an arc note that this template
         should not be called before the arcs have been checked.
        The global relevantParticipants variable contains all the ids of locaters that are relevant -->
    <xsl:template match="infoset:participant">
        <xsl:if test="contains( $relevantParticipants, concat(';', concat(@infoset:id, ';')))">
            <xsl:call-template name="copyAndApply"/>
        </xsl:if>
    </xsl:template>

    <!-- default template, copies everything (text, attributes, children) to the output -->
    <xsl:template match="*|text()|@*" priority="-1" name="copyAndApply">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

C.3 Response Generator Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:lbap="http://www.stillhard.net/2002/lbap"
    xmlns="http://www.stillhard.net/2002/lbap">
```

```
    <xsl:variable name="status" select="document('lbap_status.xml')"/>

    <xsl:template match="/">

        <lbap:linkbaseResponse>
            <!-- add status attributes -->
            <xsl:call-template name="statusAttributes"/>

            <!-- error messages if there are any -->
            <xsl:apply-templates select="$status/status/error"/>

            <!-- the linkbase (which is an empty linkset if there was an error) -->
            <lbap:linkBase>
                <xsl:copy-of select="/"/>
            </lbap:linkBase>

        </lbap:linkbaseResponse>

    </xsl:template>

    <xsl:template name="statusAttributes">

        <xsl:variable name="sizeLimitExceeded">
            <xsl:for-each select="$status/status/error">
                <xsl:if test="@type='arcLimitExceeded' or
                              @type='linkLimitExceeded' or
                              @type='participantLimitExceeded'">true</xsl:if>
            </xsl:for-each>
        </xsl:variable>
        <xsl:variable name="operationError">
            <xsl:for-each select="$status/status/error">
                <xsl:if test="@type='unknown'">true</xsl:if>
            </xsl:for-each>
        </xsl:variable>

        <!-- add error type attributes -->
        <xsl:if test="$sizeLimitExceeded = 'true'">
            <xsl:attribute name="sizeLimitExceeded">true</xsl:attribute>
        </xsl:if>
        <xsl:if test="$operationError = 'true'">
            <xsl:attribute name="operationError">true</xsl:attribute>
        </xsl:if>

        <!-- success/failure attribute -->
        <xsl:choose>
            <xsl:when test="$sizeLimitExceeded = 'true'">
                <xsl:attribute name="success">false</xsl:attribute>
            </xsl:when>
            <xsl:when test="$operationError = 'true'">
                <xsl:attribute name="success">false</xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name="success">true</xsl:attribute>
            </xsl:otherwise>
        </xsl:choose>

    </xsl:template>

    <!-- transform error elements from the lbap status document to error messages for the response -->
    <xsl:template match="error">
        <lbap:errMessage>
            <xsl:value-of select="text()"/>
        </lbap:errMessage>
    </xsl:template>

</xsl:stylesheet>
```

C.4 Linkbase to Infoset Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<!--    Document: http://www.stillhard.net/2002/xlink/infoset/infoset_from_linkbase.xsl

    Generates an XML Document that describes the information set (InfoSet) of an XLink linkbase

    Notes:
    - The linkbase should only contain schema document type definitions - no DTD
    - This version correctly generates arc elements for every defined traversal.
    - It also works correctly, if the from and to attributes are missing in an arc.
    - Additionally it supports linkbases with multiple links, title elements,
      proprietary elements and attributes and local resources

    Changes:
        - Defaulting if no arc is specified in the linkbase
        - If the xlink specific attributes on an arc are not specified they won't occur in the infoset
-->

<xsl:stylesheet version="1.1"
    xmlns="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:xlink="http://www.w3.org/1999/xlink">

    <xsl:template match="/">
        <!-- The linkSet -->
        <infoset:linkSet>
            <xsl:apply-templates/>
        </infoset:linkSet>
    </xsl:template>

    <!-- extended links -->
    <xsl:template match="*[@xlink:type='extended']">
        <infoset:link infoset:type="extended" infoset:title="{@xlink:title}" infoset:role="{@xlink:role}">
            <xsl:call-template name="anyAttribute"/>
            <xsl:attribute name="infoset:elem"><xsl:value-of select="name()"/></xsl:attribute>

            <xsl:apply-templates select="*[@xlink:type='locator']"/>
            <xsl:apply-templates select="*[@xlink:type='resource']"/>
            <xsl:apply-templates select="*[@xlink:type='arc']"/>

            <!-- in case that there are no arcs in the link, traversal
                 is defined between every resource -->
            <xsl:if test="count( *[@xlink:type='arc'] ) = 0">
                <xsl:call-template name="noarc"/>
            </xsl:if>

            <xsl:apply-templates select="*[@xlink:type='title']"/>
            <xsl:call-template name="anyContent"/>
        </infoset:link>
    </xsl:template>

    <!-- locators -->
    <xsl:template match="*[@xlink:type='locator']">
        <infoset:participant infoset:title="{@xlink:title}"
                             infoset:role="{@xlink:role}"
                             infoset:resource="{@xlink:href}"
                             infoset:participantType="locator">
            <xsl:call-template name="anyAttribute"/>
            <xsl:attribute name="infoset:elem"><xsl:value-of select="name()"/></xsl:attribute>
            <xsl:attribute name="infoset:label"><xsl:value-of select="@xlink:label"/></xsl:attribute>
            <xsl:attribute name="infoset:id">
                l<xsl:number count="*[@xlink:type='locator' or @xlink:type='resource']"/>
            </xsl:attribute>
            <xsl:call-template name="anyContent"/>
            <xsl:apply-templates select="*[@xlink:type='title']"/>
        </infoset:participant>
    </xsl:template>
```

```
<!-- resources -->
<xsl:template match="*[@xlink:type='resource']">
    <infoset:participant infoset:title="{@xlink:title}"
                         infoset:role="{@xlink:role}"
                         infoset:participantType="resource">

        <xsl:call-template name="anyAttribute"/>
        <xsl:attribute name="infoset:elem"><xsl:value-of select="name()"/></xsl:attribute>
        <xsl:attribute name="infoset:label"><xsl:value-of select="@xlink:label"/></xsl:attribute>

        <!-- The content of the resource element does not go into
             the resource attribute, but into the element -->
        <xsl:attribute name="infoset:resource"/>
        <xsl:attribute name="infoset:id">
            l<xsl:number count="*[@xlink:type='locator' or @xlink:type='resource']"/>
        </xsl:attribute>
        <xsl:apply-templates select="*[@xlink:type='title']"/>
        <xsl:call-template name="anyContent"/>
    </infoset:participant>
</xsl:template>

<!-- arcs -->
<xsl:template match="*[@xlink:type='arc']">
    <!-- Idea: For every startParticipant (identified by the from attribute of the arc)
         we look up the corresponding locator. If it was found, then we look up the
         locator for every endParticipant in the arc. This is done recursively.
    In other words:
    for every start participant do
        for every end participant do
            print arc
            print start participant
            print end participant
-->
    <!-- start participant -->
    <xsl:apply-templates select="../*[@xlink:type='locator']" mode="startParticipant">
        <!-- these params will be used to identify the correct participants -->
        <xsl:with-param name="fromlabel"><xsl:value-of select="@xlink:from"/></xsl:with-param>
        <xsl:with-param name="tolabel"> <xsl:value-of select="@xlink:to"/></xsl:with-param>

        <!-- the information about the current arc needs to be "forwarded",
             because the arc element will be printed later -->
        <xsl:with-param name="arctitle">    <xsl:value-of select="@xlink:title"/></xsl:with-param>
        <xsl:with-param name="arcrole"><xsl:value-of select="@xlink:arcrole"/></xsl:with-param>
        <xsl:with-param name="show"><xsl:value-of select="@xlink:show"/></xsl:with-param>
        <xsl:with-param name="actuate"><xsl:value-of select="@xlink:actuate"/></xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<!-- This template for arcs is only used if there are no arcs in the link -->
<xsl:template name="noarc">
    <!-- Idea: For every startParticipant (identified by the from
         attribute of the arc) we look up the corresponding locator.
         If it was found, then we look up the locator for every
         endParticipant in the arc. This is done recursively.
    In other words:
    for every start participant do
        for every end participant do
            print arc
            print start participant
            print end participant
-->
    <!-- start participant -->
    <xsl:apply-templates select="*[@xlink:type='locator']" mode="startParticipant">
        <!-- these params will be used to identify the correct participants -->
        <xsl:with-param name="fromlabel"></xsl:with-param>
        <xsl:with-param name="tolabel"></xsl:with-param>
        <!-- the information about the current arc needs to be "forwarded",
             because the arc element will be printed later -->
        <xsl:with-param name="arctitle">    <xsl:value-of select="@xlink:title"/></xsl:with-param>
        <xsl:with-param name="arcrole"><xsl:value-of select="@xlink:arcrole"/></xsl:with-param>
        <xsl:with-param name="show"><xsl:value-of select="@xlink:show"/></xsl:with-param>
```

```
        <xsl:with-param name="actuate"><xsl:value-of select="@xlink:actuate"/></xsl:with-param>
    </xsl:apply-templates>
</xsl:template>

<!-- startParticipant locators -->
<xsl:template match="//*[@xlink:type='locator']" mode="startParticipant">
    <xsl:param name="fromlabel"/>
    <xsl:param name="tolabel"/>
    <xsl:param name="arctitle"/>
    <xsl:param name="arcrole"/>
    <xsl:param name="show"/>
    <xsl:param name="actuate"/>
    <!-- Check whether the label matches. If no from is specified, then default to all participants -->
    <xsl:if test="@xlink:label=$fromlabel or  $fromlabel=''">
        <!-- now for every endParticipant -->
        <xsl:apply-templates select="../*[@xlink:type='locator' or @xlink:type='resource']"
                             mode="endParticipant">
            <!-- the idRef of this starting resource -->
            <xsl:with-param name="startIdRef">
                l<xsl:number count="*[@xlink:type='locator' or @xlink:type='resource']"/>
            </xsl:with-param>
            <!-- pass on the parameters -->
            <xsl:with-param name="fromlabel">   <xsl:value-of select="$fromlabel"/></xsl:with-param>
            <xsl:with-param name="tolabel"> <xsl:value-of select="$tolabel"/></xsl:with-param>
            <xsl:with-param name="arctitle"><xsl:value-of select="$arctitle"/></xsl:with-param>
            <xsl:with-param name="arcrole"><xsl:value-of select="$arcrole"/>    </xsl:with-param>
            <xsl:with-param name="show"><xsl:value-of select="$show"/></xsl:with-param>
            <xsl:with-param name="actuate"><xsl:value-of select="$actuate"/></xsl:with-param>
        </xsl:apply-templates>
    </xsl:if>
</xsl:template>

<!-- endParticipant locators -->
<xsl:template match="//*[@xlink:type='locator' or @xlink:type='resource']" mode="endParticipant">
    <xsl:param name="startIdRef"/>
    <xsl:param name="fromlabel"/>
    <xsl:param name="tolabel"/>
    <xsl:param name="arctitle"/>
    <xsl:param name="arcrole"/>
    <xsl:param name="show"/>
    <xsl:param name="actuate"/>
    <!-- Check whether the label matches. If no to is specified, then default to all participants -->
    <xsl:if test="@xlink:label=$tolabel or $tolabel=''">
        <!-- print arc element -->
        <infoset:arc infoset:title="{$arctitle}">
            <!-- attach xlink specific attributes, but only if they
                 are specified (the attributes are optional in XLink-->
            <xsl:if test="$arcrole != ''">
                <xsl:attribute name="infoset:arcrole"><xsl:value-of select="$arcrole"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="$show!= ''">
                <xsl:attribute name="infoset:show"><xsl:value-of select="$show"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="$actuate!= ''">
                <xsl:attribute name="infoset:actuate"><xsl:value-of select="$actuate"/></xsl:attribute>
            </xsl:if>
            <!-- find the arc element again -->
            <xsl:for-each select="../*[@xlink:type='arc']">
                <!-- There are no two arcs with the same label information -->
                <xsl:if test="@xlink:from=$fromlabel and @xlink:to=$tolabel">
                    <!-- Now copy all the content in the arc -->
                    <xsl:call-template name="anyAttribute"/>
                    <xsl:attribute name="infoset:elem"><xsl:value-of select="name()"/></xsl:attribute>
                    <xsl:call-template name="anyContent"/>
                    <xsl:apply-templates select="*[@xlink:type='title']"/>
                </xsl:if>
            </xsl:for-each>

            <!-- print startParticipant -->
            <infoset:startParticipant>
                <xsl:attribute name="infoset:idref"><xsl:value-of select="$startIdRef"/></xsl:attribute>
```

```
                </infoset:startParticipant>
                <!-- print endParticipant -->
                <infoset:endParticipant>
                    <xsl:attribute name="infoset:idref">
                        l<xsl:number count="*[@xlink:type='locator' or @xlink:type='resource']"/>
                    </xsl:attribute>
                </infoset:endParticipant>
            </infoset:arc>
        </xsl:if>
    </xsl:template>

    <!-- titles -->
    <xsl:template match="*[@xlink:type='title']">
        <infoset:title>
            <xsl:attribute name="infoset:elem"><xsl:value-of select="name()"/></xsl:attribute>
            <xsl:call-template name="anyAttribute"/>
            <xsl:call-template name="anyContent"/>
        </infoset:title>
    </xsl:template>

    <!-- any content: Used to copy the content of an element and all non
         XLink child elements (elements that don't have the xlink:type attribute -->
    <xsl:template name="anyContent">
            <xsl:value-of select="text()"/>
            <xsl:for-each select="*[ not (@xlink:type)]">
                <xsl:copy-of select="."/>
            </xsl:for-each>
    </xsl:template>

    <!-- any element: Used to copy the content of an element (without the character content) and all non
         XLink child elements (elements that don't have the xlink:type attribute -->
    <xsl:template name="anyElement">
            <xsl:for-each select="*[ not (@xlink:type)]">
                <xsl:copy-of select="."/>
            </xsl:for-each>
    </xsl:template>

    <!-- any attribute: Used to attach all attributes from another namespace then xlink -->
    <xsl:template name="anyAttribute">
            <xsl:for-each select="./@*">
                <!-- don't add xlink specific attributes. They are obvious.-->
                <xsl:if test="namespace-uri(.) != 'http://www.w3.org/1999/xlink'">
                    <xsl:copy/>
                </xsl:if>
            </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

C.5 Infoset to Linkbase Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
    Document: http://www.stillhard.net/2002/xlink/infoset/linkbase_from_infoset.xsl

    Generates an XML Document that describes the information set (InfoSet) of an XLink linkbase
-->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:infoset="http://www.stillhard.net/2002/xlink/infoset"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns="http://www.stillhard.net/2002/xlink/linkbase">

    <xsl:template match="/">
        <!-- The linkbase -->
        <xsl:text disable-output-escaping="yes">
            &lt;!DOCTYPE linkbase SYSTEM "http://www.stillhard.net/da/Work/XLink/xlinksample.dtd"&gt;
        </xsl:text>
        <linkbase>
            <xsl:apply-templates/>
```

```
    </linkbase>
</xsl:template>

<!-- extended links -->
<xsl:template match="infoset:link">
    <xsl:element name="{@infoset:elem}">
        <xsl:attribute name="xlink:type">extended</xsl:attribute>
        <xsl:attribute name="xlink:title"><xsl:value-of select="@infoset:title"/></xsl:attribute>
        <xsl:attribute name="xlink:role"><xsl:value-of select="@infoset:role"/></xsl:attribute>
        <xsl:call-template name="anyAttribute"/>
        <xsl:call-template name="anyContent"/>
        <xsl:apply-templates select="infoset:title"/>
        <xsl:apply-templates select="infoset:participant"/>
        <xsl:apply-templates select="infoset:resource"/>
        <xsl:call-template name="arcs"/>
    </xsl:element>
</xsl:template>

<!-- locators and resource -->
<xsl:template match="infoset:participant">
    <xsl:element name="{@infoset:elem}">
        <xsl:attribute name="xlink:type"><xsl:value-of select="@infoset:participantType"/></xsl:attribute>
        <xsl:attribute name="xlink:title"><xsl:value-of select="@infoset:title"/></xsl:attribute>
        <xsl:attribute name="xlink:role"><xsl:value-of select="@infoset:role"/></xsl:attribute>
        <xsl:if test="@infoset:participantType = 'locator'">
            <xsl:attribute name="xlink:href"><xsl:value-of select="@infoset:resource"/></xsl:attribute>
         </xsl:if>
        <xsl:attribute name="xlink:label"><xsl:value-of select="@infoset:label"/></xsl:attribute>
        <xsl:call-template name="anyAttribute"/>
        <xsl:call-template name="anyContent"/>
        <xsl:apply-templates select="infoset:title"/>
    </xsl:element>
</xsl:template>


<!-- arcs -->
<xsl:template name="arcs">
    <!-- This will output traversal information. In procedural language this means:
        If there are no arcs:
            output nothing
        Else If there are arcs connecting every resource with every other:
            output nothing
        Else
            For every infoset:arc element
                Lookup the from and to label of the participants referenced by the idref attribute
            For every different from-to pair output an arc
    -->

    <!-- find out if there are any arcs and store this in a variable -->
    <xsl:variable name="noarcs">
        <xsl:for-each select="infoset:arc">1</xsl:for-each>
    </xsl:variable>

    <!-- find out if there are arcs connecting every resource
         with every other and store this in a variable -->
    <!-- We just assume that it is not the case -->
    <xsl:variable name="allarcs"></xsl:variable>

    <!-- DEBUG
    noarcs=<xsl:value-of select="$noarcs"/>
    allarcs=<xsl:value-of select="$allarcs"/>
    END DEBUG -->

    <xsl:if test="$noarcs != '' and $allarcs = ''">
        <!-- we need to output arcs! -->

        <xsl:apply-templates select="infoset:arc[1]">
            <xsl:with-param name="donePairs"/>
            <xsl:with-param name="pos">1</xsl:with-param>
        </xsl:apply-templates>
```

```
        </xsl:if>

    </xsl:template>

    <!-- match arcs -->
    <xsl:template match="infoset:arc">
        <xsl:param name="donePairs"/>
        <xsl:param name="pos"/>

        <!-- first get the refIds of the participants in variables -->
        <xsl:variable name="startId">
            <xsl:for-each select="infoset:startParticipant">
                <xsl:value-of select="@infoset:idref"/>
            </xsl:for-each>
        </xsl:variable>
        <xsl:variable name="endId">
            <xsl:for-each select="infoset:endParticipant">
                <xsl:value-of select="@infoset:idref"/>
            </xsl:for-each>
        </xsl:variable>

        <!-- get the from and to label of the resources connected by
             this arc and put them into vars for later usage-->
        <xsl:variable name="from">
            <xsl:for-each select="../infoset:participant">
                <!-- startID -->
                <xsl:if test="@infoset:id = $startId"><xsl:value-of select="@infoset:label"/></xsl:if>
            </xsl:for-each>
        </xsl:variable>

        <xsl:variable name="to">
            <xsl:for-each select="../infoset:participant">
                <!-- endID -->
                <xsl:if test="@infoset:id = $endId">
                    <xsl:value-of select="@infoset:label"/>
                </xsl:if>
            </xsl:for-each>
        </xsl:variable>

        <xsl:variable name="fromto">
            <xsl:value-of select="$from"/>-<xsl:value-of select="$to"/>
        </xsl:variable>

        <!-- DEBUG
        donePairs: <xsl:value-of select="$donePairs"/>
        startID: <xsl:value-of select="$startId"/>
        endID: <xsl:value-of select="$endId"/>
        fromto: <xsl:value-of select="$fromto"/>
        pos: <xsl:value-of select="$pos"/>
        END DEBUG -->

        <!-- only output the arc if it's corresponding from-to pair was not already written -->
        <xsl:if test="not( contains($donePairs, $fromto) )">
            <xsl:element name="{@infoset:elem}">
                <xsl:attribute name="xlink:type">arc</xsl:attribute>
                <xsl:attribute name="xlink:title"><xsl:value-of select="@infoset:title"/></xsl:attribute>
                <xsl:attribute name="xlink:arcrole"><xsl:value-of select="@infoset:arcrole"/></xsl:attribute>
                <xsl:attribute name="xlink:from"><xsl:value-of select="$from"/></xsl:attribute>
                <xsl:attribute name="xlink:to"><xsl:value-of select="$to"/></xsl:attribute>
                <xsl:attribute name="xlink:show"><xsl:value-of select="@infoset:show"/></xsl:attribute>
                <xsl:attribute name="xlink:actuate"><xsl:value-of select="@infoset:actuate"/></xsl:attribute>
                <xsl:call-template name="anyAttribute"/>
                <xsl:call-template name="anyContent"/>
                <xsl:apply-templates select="infoset:title"/>
            </xsl:element>
        </xsl:if>

        <!-- go to the next arc element -->
        <xsl:apply-templates select="../infoset:arc[$pos+1]">
            <xsl:with-param name="donePairs"><xsl:value-of select="$donePairs"/>
                <xsl:value-of select="$fromto"/>;
```

```
            </xsl:with-param>
            <xsl:with-param name="pos"><xsl:value-of select="$pos + 1"/></xsl:with-param>
        </xsl:apply-templates>

    </xsl:template>

    <!-- titles -->
    <xsl:template match="infoset:title">
        <xsl:element name="{@infoset:elem}">
            <xsl:attribute name="xlink:type">title</xsl:attribute>
            <xsl:call-template name="anyAttribute"/>
            <xsl:call-template name="anyContent"/>
        </xsl:element>
    </xsl:template>

    <!-- any element: Used to copy the content of an element and all non XLink
         child elements (elements that don't have the xlink:type attribute -->
    <xsl:template name="anyContent">
            <xsl:value-of select="text()"/>
            <xsl:for-each select="*[ namespace-uri(.) != 'http://www.stillhard.net/2002/xlink/infoset' ]">
                <xsl:copy-of select="."/>
            </xsl:for-each>
    </xsl:template>

    <!-- any attribute: Used to attach all attributes from another namespace then xlink -->
    <xsl:template name="anyAttribute">
            <xsl:for-each select="./@*">
                <!-- don't add xlink specific attributes. They are obvious.-->
                <xsl:if test="namespace-uri(.) != 'http://www.stillhard.net/2002/xlink/infoset'">
                    <xsl:copy/>
                </xsl:if>
            </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

C.6 LBAP Prototype MS-DOS Batch

Note: Remove the line breaks in the calls to java. They have been added for formatting reasons.

```
@echo off

cls

set requestFile=%1
set linkbaseFile=%2

if "%1"=="" GOTO DEFAULTBOTH
if "%2"=="" GOTO DEFAULTLB

GOTO NODEFAULT

:DEFAULTBOTH
set requestFile=sample_request.xml
echo No request file specified. Taking default %requestFile%

:DEFAULTLB
set linkbaseFile=family_linkbase.xml
echo No linkbase file specified. Taking default %linkbaseFile%

:NODEFAULT
echo applying request from file %requestFile%
echo transforming the linkbase into it's infoset
java com.icl.saxon.StyleSheet
    -o temp\temp_family_infoset.xml
    %linkbaseFile%
    infoset_from_linkbase.xsl

echo transforming the request into a request-processor (xsl)
```

```
java com.icl.saxon.StyleSheet
    -o temp\temp_request_processor.xsl
    %requestFile%
    request_processor_generator.xsl

echo applying the request to the infoset
java com.icl.saxon.StyleSheet
    -o temp\temp_family_infoset_filtered.xml
    temp\temp_family_infoset.xml
    temp\temp_request_processor.xsl

echo transforming the infoset back into a linkbase
java com.icl.saxon.StyleSheet
    -o temp\result_family_linkbase.xml
    temp\temp_family_infoset_filtered.xml
    linkbase_from_infoset.xsl

echo generating the response
java com.icl.saxon.StyleSheet
    -o response.xml
    temp\result_family_linkbase.xml
    response_generator.xsl

echo opening the response
response.xml

echo DONE...
```

Appendix D

Assignment

Zürich, 9.11.01

Aufgabenstellung Diplomarbeit WS 2001/2002 für Christian Stillhard

In der XML-Standardisierung sind mit XLink und XPointer zwei wichtige Standards verabschiedet worden, die für die Realisierung von XML-basiertem Hypertext auf dem Web den Kernpunkt darstellen. XLink verallgemeinert das Link-Modell von HTML um eine ganze Reihe von Aspekten, insbesondere um die Möglichkeit zur Erzeugung von Out-of-line Links, d.h. Links, die nicht Bestandteil einer der Resourcen sind, die sie verbinden. Einer der Kernaspekte bei diesen Links ist die Existenz von Linkbases, d.h. einer externen Resource, die Links enthält. XLink definiert die Arten, wie Links codiert werden müssen, liefert aber keine Grundlage dazu, wie man auf eine Linkbase mit einer Query-Sprache zugreifen kann. Angesichts der Tatsache, dass Linkbases gross sein können, ist ein solcher Verfahren jedoch wünschenswert, und das Design und die Implementierung eines solches Zugriffsprotokolls sind das Thema dieser Diplomarbeit.

Auf dem Weg zu einem XML-basierten Web ist hinsichtlich der Standardisierung momentan im Bereich des Zugriffsprotokolls auf Linkbases eine Lücke. Diese Lücke soll mit Hilfe dieser Diplomarbeit gefüllt werden, deshalb ist es wichtig, dass die Arbeit seriös und methodisch angegangen wird. Wichtige Aspekte dabei sind:

- Wie sehen die Anforderungen aus? Am Beginn der Arbeit steht eine detaillierte und umfassende Anforderungsanalyse, die als Richtlinie für die darauffolgenden Konzeptarbeiten dienen wird.
- Welches Transportprotokoll soll verwendet werden, und in welcher Art? Wahrscheinlich wird man auf HTTP zurückgreifen, aber auch dann gibt es weitere wichtige Designfragen zu beschreiben und zu entscheiden, z.B. die Einbettung des Queries in einen Request (mögliche Varianten wären hier als URI Query String oder als Request Header Field).
- Welche Query-Mechansimen gibt es? Die Idee des Linkbase-Protokolls hat als zentralen Punkt die Query-Möglichkeit, aber die Art und Weise der möglichen Queries muss noch festgelegt werden. Damit hängt natürlich zusammen, welches Linkmodell als Grundlage genommen wird.
- Wie sollen die Query-Mechansimen definiert werden? Mögliche Varianten hier sind eine komplette eigene Spezifikation (sehr aufwendig und wahrscheinlich nicht empfehlenswert),

eine Abbildung auf bestehende Sprachen mit Query-Funktionalität (z.B. XPath oder XQuery), oder eine Abbildung auf eine Referenzimplementierung mit XSLT.

- Wie ist das Processing Model? Wie verhalten sich die Endsysteme des Zugriffsprotokolls?

Ziel der Diplomarbeit ist es, ein Konzept und eine prototypische Implementierung zu erstellen, die es ermöglicht, mit den erarbeiteten Konzepten praktische Tests durchzuführen. Die wichtigsten Aspekte bei diesen Tests sind Robustheit, Skalierbarkeit, Flexibiliät und Erweiterbarkeit der Lösung. Einer der Kernpunkte des Berichts sollen zwei technische Reports (Requirements Analysis und Protocol Specification) sein, die man mit wenigen Überarbeitungen direkt an die relevanten Gremien (W3C und IETF) als Standardisierungsvorschlag weiterleiten kann.

Appendix E

Project Schedule

This project was done from November 2001 to March 2002. Figure E.1 shows a detailed plan which lists the single tasks and shows the amount of time invested in them.

Task Name	Duration	Start	Finish	Predecessor
General	**14 days**	**Mon 11/5/01**	**Thu 11/22/01**	
Overview	4 days	Mon 11/5/01	Thu 11/8/01	
Basics	4 days	Fri 11/9/01	Wed 11/14/01	2
Structure the work	6 days	Thu 11/15/01	Thu 11/22/01	3
The linking model	**6 days**	**Fri 11/23/01**	**Fri 11/30/01**	**1**
Describe Xlink linking model	1 day	Fri 11/23/01	Fri 11/23/01	
Extension of Xlink link semantics	1 day	Mon 11/26/01	Mon 11/26/01	7
Impact on the web	2 days	Tue 11/27/01	Wed 11/28/01	8
Other linking models	1 day	Thu 11/29/01	Thu 11/29/01	9
Comparison of Xlink with other lin	1 day	Fri 11/30/01	Fri 11/30/01	10
Scenarios	**4 days**	**Mon 12/3/01**	**Thu 12/6/01**	**6**
Work out and describe	4 days	Mon 12/3/01	Thu 12/6/01	
Linkbase access semantics	**10 days**	**Fri 12/7/01**	**Thu 12/20/01**	**13**
Work out and write	10 days	Fri 12/7/01	Thu 12/20/01	
Requirements	**10 days**	**Fri 12/21/01**	**Thu 1/3/02**	**16**
Querying/Data model	1 day	Fri 12/21/01	Fri 12/21/01	
Processing model	1 day	Thu 12/27/01	Thu 12/27/01	20
Compatibility with XML standards	1 day	Fri 12/28/01	Fri 12/28/01	21
Protocol binding	1 day	Mon 12/31/01	Mon 12/31/01	22
Bring it together	1 day	Tue 1/1/02	Tue 1/1/02	23
Document	2 days	Wed 1/2/02	Thu 1/3/02	24
Protocol specification	**17 days**	**Fri 1/4/02**	**Mon 1/28/02**	**19**
Mapping of linkbase queries	4 days	Fri 1/4/02	Wed 1/9/02	
Data model	2 days	Thu 1/10/02	Fri 1/11/02	26
Processing model	2 days	Mon 1/14/02	Tue 1/15/02	29
Underlying protocol binding	3 days	Wed 1/16/02	Fri 1/18/02	30
Syntax definition	2 days	Mon 1/21/02	Tue 1/22/02	31
Write specification	4 days	Wed 1/23/02	Mon 1/28/02	32
Implementation	**10 days**	**Tue 1/29/02**	**Mon 2/11/02**	**27**
Design	2 days	Tue 1/29/02	Wed 1/30/02	
Implementation	5 days	Thu 1/31/02	Wed 2/6/02	36
Testing	2 days	Thu 2/7/02	Fri 2/8/02	37
Documentation	1 day	Mon 2/11/02	Mon 2/11/02	38
Documentation	**14 days**	**Tue 2/12/02**	**Fri 3/1/02**	**35**
Writing	12 days	Tue 2/12/02	Wed 2/27/02	
Review	3 days	Tue 2/12/02	Thu 2/14/02	
Finish-up	2 days	Thu 2/28/02	Fri 3/1/02	

Figure E.1: Tasks and timeplan for this project

Bibliography

[1] T. BERNERS-LEE, R. CAILLIAU, A. LUOTONEN, H.F. NIELSEN, A. SECRET. The World-Wide Web, Communications of the ACM. 37. 1994.

[2] DAN CONNOLLY. A Little History of the World Wide Web, `http://www.w3.org/History.html`, (Access: November 19th 2001).

[3] R. FIELDING, J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH, T. BERNERS-LEE. Hypertext Transfer Protocol – HTTP/1.1. Internet Proposed Standard RFC2616. June 1999.

[4] S. CHAKRABARTI, B. DOM, S. R. KUMAR, P. RAGHAVAN, S. RAJAGOPALAN, A. TOMKINS. Hypersearching the Web. Scientific American. June 1999.

[5] Google Inc., Google, `http://www.google.com`

[6] ORA LASSILA and RALPH R. SWICK. Resource Description Framwork (RDF) Model and Syntax Specification, World Wide Web Consortium, Recommondation REC-rdf-syntax-19990222, February 1999.

[7] DAVE RAGGETT, ARNAUD LE HORS, IAN JACOBS. HTML 4.01 Specification. World Wide Web Consortium, Recommendation REC-html401-19991224, December 1999.

[8] TIM BRAY, JEAN PAOLI, C. M. SPERBERG-MCQUEEN, AND EVE MALER. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, REC-xml-20001006, October 2000.

[9] D. LOWE, E. WILDE. Improving Web linking using XLink. In Proceedings of Open Publish 2001.

[10] T. H. NELSON. Literary Machines. Mindful Press, Sausalito, California, 1982.

[11] STEVEN J. DEROSE, EVE MALER, AND RON DANIEL. XML Pointer Language (XPointer) Version 1.0. World Wide Web Consortium, Candidate Recommendation CR-xptr-20010911, September 2001.

[12] STEVEN J. DEROSE, EVE MALER, AND DAVID ORCHARD. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[13] NORMAN WALSH. XML Linking and Style. World Wide Web Consortium, Note NOTE-xml-link-style-20010605, June 2001.

[14] JAKOB NIELSEN. Usability Engineering. Published by Morgan Kaufmann, San Francisco (originally published by AP Professional, Boston). 1994.

[15] UFFE KOCK WIIL AND JOHN J. LEGGETT. The HyperDisco Approach to Open Hypermedia Systems, UK Conference on Hypertext 1996. 1996.

[16] KENNETH m. ANDERSON. Integrating Open Hypermedia Systems with the World Wide Web. In Proceedings of the 1997 ACM Conference on Hypertext. Pages 157-166. Southampton. UK, April 1997. ACM Press.

[17] KAY GRØNBÆK AND UFFE KOCK WIIL. Towards a Common Reference Architecture for Open Hypermedia. Journal of Digital Information, 1(2), 1997.

[18] PETER J. NÜRNBERG and JOHN J. LEGGETT. A Vision for Open Hypermedia Systems. Journal of Digital Information, 1(2), 1997.

[19] JOHN COWAN and RICHARD TOBIN. XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[20] SAMHAA EL-BELTAGY, WENDY HALL, DAVID C. DE ROURE, AND LESLIE A. CARR. Linking in Context. In Proceedings of the 12th ACM Conference on Hypertext and Hypermedia, pages 151–160.

[21] DAVID C. DE ROURE, NIGEL G.WALKER, AND LESLIE A. CARR. Investigating Link Service Infrastructures. In Proceedings of the 11th ACM Conference on Hypertext and Hypermedia. pages 151–160.

[22] SCOTT BRADNER. Key words for use in RFCs to Indicate Requirement Levels. Internet Best Current Practice RFC 2119, March 1997.

[23] SHISHIR GUNDAVARAM. CGI Programming on the World Wide Web. O'Reilly & Associates, Sebastopol, California, April 1996.

[24] JAMES CLARK. XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium, Recommendation REC-xslt-19991116, November 1999.

[25] T. BERNERS-LEE, R. FIELDING, U.C. IRVINE, L. MASINTER. Uniform Resource Identifiers (URI): Generic Syntax. Internet Draft Standard RFC 2616. August 1998.

[26] STUART L. WEIBEL, JOHN A. KUNZE, CARL LAGOZE, and MISHA WOLF. Dublin Core Metadata for Resource Discovery. Internet informational RFC 2413, September 1998.

[27] STEVE PEPPER and GRAHAM MOORE. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification xtm1-20010806, August 2001.

[28] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Information technology — SGML Applications — Topic Maps. ISO/IEC 13250, 2000.

[29] ERIK WILDE and DAVID LOWE. Transcluding the Web. Addison-Wesley, Reading, Massachusetts, to be published in 2002.

[30] RON DANIEL JR. Harvesting RDF Statements from XLinks. World Wide Web Consortium, Note NOTE-xlink2rdf-20000929, September 2000.