Graz University of Technology

Institute for Information Processing
and Computer Supported New Media

Diplomarbeit aus Telematik
Informationssysteme

Advanced Link Management in Hypermedia Systems

Karl Bluemlinger

Betreuer
DI. Dr. techn. Klaus Schmaranz
Begutachter
Univ. Prof. Dr. phil. Dr. h. c. Hermann Maurer

Graz, November 2000

Acknowledgments

I would like to thank Klaus Schmaranz and Heimo Haub who urged me to finish my studies. Special thanks to Prof. Hermann Maurer who supports the whole Dinopolis team. I also want to thank the readers of this work, Christof Dallermassl and, once again, Klaus Schmaranz.

Greetings to the developers of Linux and LaTeX who enabled me to write this master's thesis on a reliable environment.

I highly estimate the support of my parents through my studies which goes far beyond of what a son could expect. Thanks also to Lena, who guided me through the last years of my studies.

I hereby certify that the work reported in this thesis is my own and that work performed by others is appropriately cited.

Signature of the author:

Ich versichere hiermit, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten Anderer unverändert oder mit Abänderungen entnommen wurde.

Abstract

The link capabilities provided by the hypertext markup language (HTML) do not comply with the demands of modern hypermedia systems. This master's thesis discusses the requirements on a modern link management system which are link consistency, typed links, personalized links, and the ability to create dynamic links. Two prerequisites to fulfill these requirements are important: A flexible and standardized markup language to describe links and a distributed link service which implement those features. The markup language discussed in this thesis is XLink which currently undergoes the standardization process of the World Wide Web Consortium. The distributed link service described is part of the Dinopolis middleware system developed at the IICM. It uses XLink as its linking language and supports the requirements stated above. The fundamentals of this link service are outlined and the implementation of its efficient forwarding mechanism to speed up link consistency is described.

Kurzfassung

Die Unterstützung von Hyperlinks der im World Wide Web vorherrschenden markup Sprache HTML entspricht nicht den Erfordernissen eines modernen Hypermedia Systems. Die vorliegende Diplomarbeit zeigt die Anforderungen eines Link Management Systems, zu denen die Konsistenz von Links, typisierte und personalisierte Links als auch die Fähigkeit, dynamisch Links zu generieren, zählt. Zwei Voraussetzungen, um diese Anforderungen zu erfüllen, müssen gegeben sein: Eine standardisierte Sprache, um Hyperlinks zu beschreiben und ein verteiltes Link Service, welches die oben genannten Eigenschaften erfüllt. Die sich derzeit im Standardisierungsprozess befindliche markup Sprache "XLink" wird in dieser Arbeit diskutiert. Die Grundlagen und die Implementierung des verteilten Link Service des Dinopolis Middleware Systems, welches am IICM entwickelt wird, werden ausführlich behandelt. Es verwendet XLink zur Beschreibung von Hyperlinks und entspricht den Anforderungen eines modernen Link Managements Systems.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The World Wide Web gives authors the possibility to easily and cheaply distribute electronic documents to an international audience. One of the driving forces of the Web was the concept of links [Rag98] which provide a convenient way to navigate between documents. Nevertheless as Web documents became larger and more complex in their structure, the limitations of the predominant mark-up language in the Web, HTML [Gro99a] , became obvious: The demand for separation of structure and content, extensibility and data checking grew [Bos97]. Links no longer serve as a pure navigation tool but are also used to compose documents. The commercial use of the WWW led to new requirements such as easy administration of servers, no dangling links, and personalized web sites. Hypermedia systems that implement these requirements suffer from the lack of standardization and hence were forced to develop a proprietary link language . While this is well suited for closed intranet environments, it is problematic in an heterogenous environment like the World Wide Web.

This thesis discusses the features a modern link management system must provide to fulfil commercial requirements it also shows an implementation of a distributed link service (DLS) . It describes the usage and benefits of the new specification of the XML linking language called XLink which enables designers of a DLS to use a

standardized format that facilitates link maintenance. The distributed link service presented in this thesis is part of the Dinopolis project , an open source project launched by the IICM. The Dinopolis middleware system provides a sophisticated link management even to systems which do not support hyperlinks themselves. Designed as an open hypermedia system and using XML as the primary document format, its DLS uses XLink to describe hyperlinks.

1.2 Chapter Outline

- *Chapter 2* gives an introduction to hyperlinks. It provides a formal definition of hyperlinks and presents some historical facts about their origin. The weak points of hyperlinks in the World Wide Web are examined. Important issues such as link consistency, personalized and typed links are discussed as well as the limitations of addressing individual elements within a document.

- *Chapter 3* discusses the XLink specification along with the XPath and XPointer specifications. The improvements to HTML and the impact of the capabilities of XLink on link services are pointed out.

- *Chapter 4* gives an introduction to distributed link services and addresses some important issues of link servcices such as dangling links, performance, standardization, and communication between client applications and the DLS.

- *Chapter 5* is about the DLS in the Dinopolis system. A general introduction to the system is provided. The extensive requirements of the DLS as well as the implementation of its effective link maintenance mechanism are are discussed in detail.

1.3 Related Work

Several master's thesis have been written about the Dinopolis middlware system:

- Christof Dallermassl describes in [Dal99] the architecture of Dinopolis.
- Heimo Haub exposes in detail the user management of Dinopolis [Hau99].
- Philipp Zambelle developed an application based on Dinopolis that manages shared bookmarks [Zam00].

The use of the Dinopolis system in teaching environments is described in [DHK+00a, DHK+00b]. The benefits of Dinopolis and its use as an application framework is discussed in [CHKZ00].

Chapter 2

Hyperlinks

This chapter introduces hyperlinks . A definition derived from hypertext and a formal definition of hyperlinks are presented. A short outline of the history of hypertext follows, showing how old many of the ideas of hypertext and hyperlinks are. It is amazing how long it could take that certain ideas and visions are implemented even in the fast moving world of information technology. The last section in this chapter deals with links and the World Wide Web. Hyperlinks are considered to be one of the driving forces of the WWW. The immense growth of the WWW and its success in commercial use requires a more sophisticated link management. Some "missing links" such as typed links , dynamic links , and personalized links as well as link consistency are examined in more detail.

2.1 Introduction to Hyperlinks

In contrary to traditional printed text (see figure 2.1), hypertext is a nonlinear sequence of text nodes interconnected by hyperlinks. Readers navigate through a set of nodes by the use of links and it is up to the users which paths they follow (see figure 2.2).

Hypertext combined with multimedia is also called *hypermedia* which is an acronym combining both terms [Nie95]. The strength of hypermedia lies in its ability to produce large, complex, richly connected, and cross-referenced bodies of information.

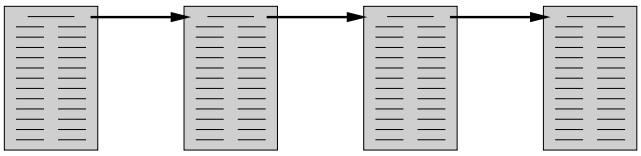

Figure 2.1: Sequential reading of documents.

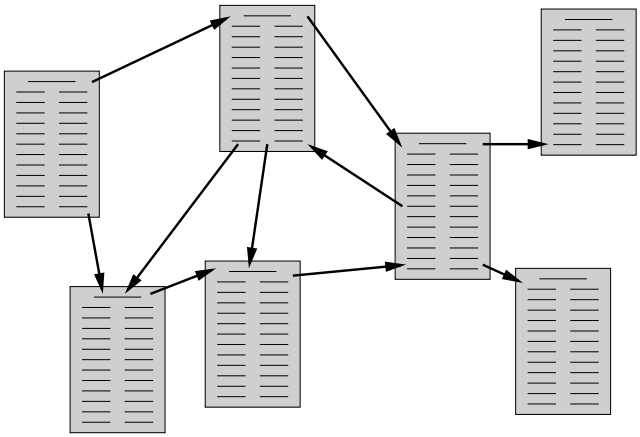

Figure 2.2: Non sequential reading of documents

John B. Smith defines Hypertext as [SW88]

> *an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video as well as source code or other forms of data.*

Smith does not distinguish between hypertext and hypermedia. Throughout this thesis the term hypermedia is used.

2.1.1 Nodes

Nodes are self-contained information units [TD96], the reader can understand the node's content without knowledge of any other node. In contrary to linear text, this is important in hypermedia because the reader can jump from node to node in an arbitrary sequence. In hypertext systems, the content of nodes can either be

discrete media such as text and pictures or continuous media like videos and audios. In addition to content, attributes can be attached to nodes which store meta-data like creation date, author, language and name of the node.

2.1.2 Hyperlinks

The essence of hypermedia is the concept of links which provide the structure of hypermedia. Basically, a link connects two nodes. A link's anchor describes the attachment of a link to a node. The starting point of a link is called source anchor , the endpoint of a link is called destination anchor [Mau96]. The direction of a link indicates whether the link is unidirectional or bidirectional . Whereas unidirectional links can only be followed from the source to the destination, bidirectional links can also be followed from the destination to the source. The structure formed by links can abstractly be seen as a graph with subgraphs of many types e.g a tree, an acyclic graph or a directed graph (see figure 2.3). Besides the structure formed by links, each link may have its own semantics indicated by its type. Some type of links are jump links, annotations, cross references or bookmarks (see section 2.3).

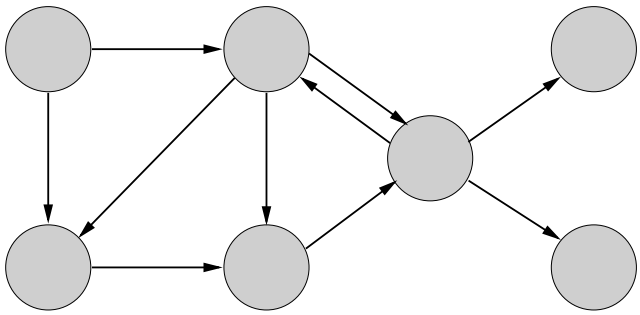

Figure 2.3: Directed graph of hypertext documents

2.2 A Formal Definition of the Link

This section presents a formal mathematical approach to define links published by Liam Quin [Qui98].

Def. 1.1: A *link* is an assertion that a relation R holds between two sets of resources.

A link can also be defined with a function:

Def. 1.2: A *linking function* is a function $f_R : S_s \rightarrow S_t$ where R is the relationship being asserted and S_s and S_t (the domain and of the link function, respectively) are the two sets of resources between which the relationship may be said to exist. S_s is informally referred to as the source of the link f and S_t as the target.

Def. 1.3: The *inverse linking function* f^{-1} is the function that maps the codomain S_t of f back into its domain S_s.

Two types of inverse functions are of particular interest, the computed and the explicit inverse function :

Def. 1.4: A *computed inverse* f^{-1} of a linking function $f_R : S_s \rightarrow S_t$ is the function obtained by evaluating f and then stating that f^{-1} is $S_t \rightarrow S_s$.

Def. 1.5: An *explicit inverse* f^{-1} of a linking function $f_R : S_1 \rightarrow S_2$ is a function $f^3 : S_s \rightarrow S_4$, with the constraint that S_2 and S_3 be either identical or have a non-empty intersection P; P may be a proper subset of S_2 since the inverse mapping may not be defined for all members of the result set of f.

An example of a computed inverse function would be the back navigation via a history buffer of an application such as web browsers (provided mostly be a “back” button). Explicit inverse functions are mostly supplied (e.g. bidirectional links) rather than computed.

Generally, link functions are not injective because the asserted relationship can be one-to-one, one-to-many, many-to-one, and many-to-many or in other words, the formal definition presented in this section imposes no restriction on the cardinality of the resource sets. An extended link as described in chapter 3 is a link which can have more than one local and remote resource.

Link functions are also not necessarily surjective because there might be resources in the codomain of the link (e.g all addressable resources in the World Wide Web) function which cannot be target of a link. The link function can also be non-deterministic, i.e. each time the function is evaluated it yields a different resource set.

In theory, computation of the inverse function is always possible. In practice, computing the inverse could be quite expensive or impossible. Finding the inverse is equivalent to the question:"What is the set of all World Wide Web documents pointing to a particular page". Some pages might be not accessible due to restricted access permissions or a server containing a particular page is not reachable.

2.3 The Origin of Hyperlinks

In 1945, Vannevar Bush published the article *What We May Think* [Bus45] where he described a library which nowadays would be called a knowledge management system or digital library . In his article Bush described a device called *memex* :

> Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, "memex" will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.

What distinguishes Bush's concept form other storage devices is the associative structure of memex which is closely related to the human mind.

> The human mind [...] operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain.

Hyperlinks are a way to allow readers to follow at least the associations of the author of a document.

The term Hypertext was created by Ted Nelson who can be regarded as one of the pioneers of hypertext systems in 1965. As early as 1967 he described the requirements of a hypertext systems called Xanadu which he later published in his book *Literary Machines* [Nel92]. One of Ted Nelson's definition of hypertext is:

> By hypertext I mean non-sequential writing–text that branches and allows choices to the reader, best read at an interactive screen. As popularly conceived, this is a series of text chunks connected by links which offer the reader different pathways.

Non-sequential writing gives authors a better possibility to express what they mean. The concept of links plays a central role in realizing non-sequential text. Nelson defines a link as a

> a simple connection between parts of text or other material. Links are part of the document and created by individuals.

Whether Ted Nelson also coined the term hyperlink is not clear. Nelson himself does not lay claim to the creation of this term. A Hypertext system must support links of any types, depending on the writer‘s need. The link types explicitly mentioned in [Nel92] are:

- bookmarks
- commentaries (annotations)
- place markers
- footnotes
- hypertext jumps
- marginal notes

Another type of link called transclusion [Nel95] was introduced by Ted Nelson. Transclusion allows to embed documents or part of documents into other documents without copying them. It allows a document appear in multiple places. Transclusion might be compared to a pointer which connects the original document to all places that use it. It avoids redundant storage of documents. In contrast to copying a document, a document included via transclusion stays up to date. Changes of the original document are reflected through transclusion. Transclusion also includes authentication to ensure that authors get credited for their work.

2.4 Hypertext and the World Wide Web

Many of Ted Nelson's and Vannevar Bush's visions made their way into the World Wide Web (WWW) which was first conceived by Robert Caillian and Tim Berners-Lee in 1989 [BLC90]. The WWW soon became a playground for hypertext and information retrieval concepts. However many of these concepts were rapidly picked up by the W3 community before developing an overall concept [RCH94]. The intention of Tim Berners-Lee was

- to provide a simple protocol for requesting human readable information stored in remote systems that is accessible using networks.
- to ease the exchange of documents independent of the display technology used.
- to provide a protocol which allows information exchange independent of the platform of the information supplier as well as the information customer use.
- to provide search options, to allow information to be automatically searched for by keywords.
- to support navigation between documents by following hyperlinks.
- to provide and maintain collections of documents into which users could place documents of their own.
- to allow documents or collections of documents managed by individuals to be hyperlinked to other documents or collections of documents.
- to use public domain software free of charge wherever possible.
- to interface to existing proprietary systems.

Information exchange takes place between the client and the server, no communication between servers was intended which makes automatic link updates a difficult task. Currently, maintaining links, that is to guarantee that destination anchors point to the correct location, is up to volunteers. As a consequence, many dangling links are around the World Wide Web.

2.4.1 Link Consistency

The World Wide Web can be regarded as an Internet-wide distributed information system that operates on a client/server basis [RCH94]. The Hypertext Transfer Protocol (HTTP) manages the flow of information from the server to the client. It was designed as an information retrieval protocol perfectly suited to read information stored in a certain location in a distributed environment. Things quickly become complicated when documents are moved, deleted or modified. Consider the following example: a research institution made a set of documents available on the World Wide Web. Those documents are heavily interlinked and referenced from many people around the world. As the number of documents increases, a reorganization of the web site is unavoidable. The site administrator now faces the problem of keeping the links up to date. Currently, there is no mechanism to keep links made by external authors consistent. Though arduous, it is possible to update links that have both source and destination anchors in the domain of this Web-Site.

One of the major problems of the World Wide Web is that links are embedded in documents [ea97]. The disadvantages of this concept are [And96]:

- Linking from read-only documents is not possible. Either there is no permission to modify the document or the medium the document is stored on is read-only (e.g. CD-ROM).
- Links are unidirectional which makes link maintenance and link maps difficult to implement.
- It's tough to link from arbitrary multimedia documents which may be composed of e.g video streams.
- It's difficult to add additional features such as link access rights and automatic link generation.

External links , stored in a external link database , decouples the link and the document the link belongs to. The advantages of external link databases are:

- The implementation of bidirectional links is less complicated thus link maintenance can be achieved.

- No modification of the document the link is originating from is required.
- Personal link databases are possible (see section 2.4.2)
- Adding access rights to links is less complicated.

The creation and maintaining of external link databases requires some additional work: a database is needed to store the links and an effective synchronization mechanism to guarantee link consistency between the databases is required [Kap95]. Additionally, links have to be included in documents before handing the documents over to the browser.

An example for a hypertext system which uses an external link database is the Hyperwave Information Server [Mau96] . Hyperwave provides bidirectional point-to-point links with individual access permissions and guaranteed link consistency. The Dinopolis middleware system described in chapter 5 also uses an external link database to implement its rich link functionality.

2.4.2 Personalized Links

Personalized links enable customization of the link structure on a per-user base. Consider the link structure outlined in figure 2.4: imagine it represents the documents of a knowledge base. Each document contains lots of references to documents with similar topics, not all of them are of interest to the user. The knowledge base is presented to all users in the same way. As the number of documents increases, it is likely that users loose track of the fairly complex document structure, in other words, they get "lost in hyperspace".

Figure 2.5 shows the same documents as figure 2.4 but with a personal view on the document structure accomplished through personal links . Only links relevant for the user are displayed. There are two possibilities to achieve a personalized view:

- Users themselves decide which links should be displayed.
- The hypertext system computes links according to the users' knowledge, preferences, or according to other characteristic properties. For example introductory explanations may be added for novice users, advanced details for experts.

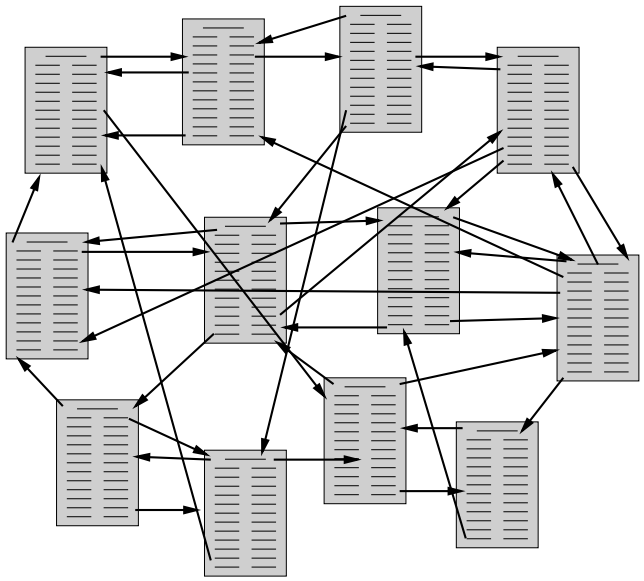

Figure 2.4: Link structure common to all users

Hypertext systems which perform link adaption or content adaption are called adaptive hypertext systems [BHW99].

Customization was not part of the initial design of the Word Wide Web. Primarily commercial web sites were the driving forces to make the World Wide Web more personal [Goo98]. Many web sites create their pages dynamically on the server depending on the users' preferences or surfing habits. The techniques that are in use today to make web content more dynamical are CGI-Scripts (Common Gateway Interface) [CR98], ASP (Advanced Server Pages) [Mue99] and Java servlets [Sag98] . All these techniques work in a similar way: Web pages are composed on the server side and are sent to the client. The information necessary to personalize web pages is either stored on the web server which requires authentication of the user to the server, or placed in cookies (small pieces of data) on the client host or a combination of both (e.g authentication data could be stored in cookies, customization data on the client). Mostly, web servers only customize the content of web pages whereas the overall link structure remains the same. The limitations of HTML makes it hard to supply different link databases to one document as links are embedded in the documents themselves.

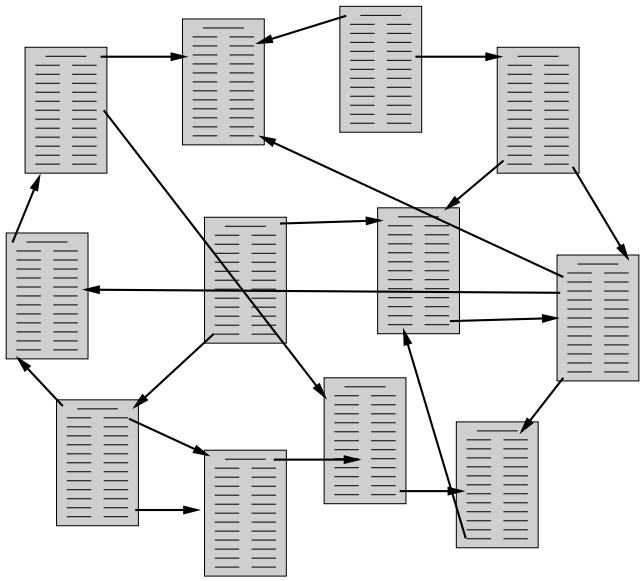

Figure 2.5: Personalized link structure

2.4.3 Dynamic Links

Many highly interlinked WWW applications such as digital libraries retrieve the content from a database and create all necessary links on the fly. Mostly this requires that authors write their documents following a predefined schema. The information of the link structure is a property of a program or script that generates the final pages. Links where the source and destination anchors are computed are called dynamic links [AV94] . Dynamic links are generated with an implicit link function as described in section 2.2. The counterpart of a dynamic link is a declared link where users choose the two endpoints of a link and then tell the Hypermedia system to create a link.

Early attempts of publishing journals on the WWW mostly leave the content untouched and added only simple navigation. As publishing in the WWW became a matter of business, more sophisticated techniques were used. An explicit meta-data database was maintained which allowed programs or scripts to impose a link structure beyond simple inner document navigation [CH]. The ACM digital library [1]

[1] `http://www.acm.org/dl/`

provides links between articles of the same subject and to all articles published by the same author.

2.4.4 Typed Links

Semantically typed links help authors to organize their information more effectively and give users important hints about the purpose of links. Semantically typed links also ease automatic information retrieval which is important for search engines [ASR94].

Wang and Rada [WR98] describe hypertext systems with a semantic net: a semantic net is a directed graph in which concepts are represented as nodes and relations between nodes are represented as links. They distinguish three types of semantic-based hypertext systems:

- An unstructured hypertext system can have arbitrary node and link types, the use of which is free from constraints. The preservation of any patterns depends wholly on voluntary actions of the authors.
- A semistructured hypertext system provides a set of recommended link types, but relies on users' voluntary actions to implement the constraints.
- A structured hypertext system has rules for structural and relational constraints, and these constraints are enforced by the hypertext system.

In theory, the World Wide Web may be regarded as a semistructured hypertext system. In 1996 M. Maloney [MQ96] proposed an extension to the HTML 2.0 specification [BLC] concerning the HTML `<a>` and `<link>` tags. With some minor changes, his proposal became a W3C recommendation [Gro99a]. According to the W3C recommendation, both the `<a>` and the `<Link>` tags have a `rel` and a `rev` attribute which specify the forward and reverse link types, respectively. In the initial proposal, there are several categories of link types:

- Browser defined links (home, back, forward) which are used by user agents (WWW clients) to traverse the recent history.

- Navigational node links (`contents`, `index`, `navigator`) which assist users through a closed document set (comparable to the linear text, see section 2.1). The navigator type points to a navigational aid which may consist of a whole or partial "table of contents", a list of related documents, an indication of the current document's location within a document hierarchy, or any other information which may be useful to the user.
- Hierarchy links (`child`, `parent`, `sibling`, `top`) may be used by a user agent to visualize the hierarchical structure of documents.
- Sequence links (`begin`, `end`, `next`, `previous`) offer the author of a document to specify a sequence the users should follow.
- Link types to indicate related documents such as `citation`, `biblioentry`, `glossary`, and `definitions`.
- Link types to indicate meta documents. There are classes of information which are not intrinsic to a document, but for which a clear and unambiguous association is often useful or even necessary such as `author`, `publisher`, `copyright`, `disclaimer`, `editor` etc.
- Some other link types like `lang` which indicate the language of the target document, `bookmarks` which provide key entry points to documents or `translation` which point to a translation of the same document.

Appendix A lists all link types which made their way into the W3C recommendation. Unfortunately, the widely used browsers Netscape Communicator and Internet Explorer only support a small fraction of the specified link types. One reason might be that the specification makes no suggestion of the implementation [Goo98]. For example Web browsers could render semantic link types in a pop-up window or in a separate frame. Web-based editors could make it easy for authors to assign semantic types to links e.g by providing a pop-up window or an input field showing the possible link types. The XML linking language XLink gives much more possibilities to assign meta data to hyperlinks. Although still a candidate recommendation, its likely that XLink will become the new standard as soon as browsers implement the full XML functionality.

2.4.5 Addressing in the WWW

The primary way to identify resources in the World Wide Web is to describe their location by the use of Uniform Resource Locators (URL) which are per definition not stable [BLFIM98].

Another problem is identification of resources within an HTML document. The source anchor of a link in HTML is defined by the `<a>` tag, the destination anchor of a link is specified by the `name` or `id` attribute:

```
<h1>Table of Contents</h1>
<p><a href="#section1">Introduction</a><br>
<a href="#section2">HTML 4.01</a><br>
...the rest of the table of contents...
...begin of content...
<h2 id="section1">Introduction</h2>
...section 1...
<h2 id="section2">HTML 4.01</h2>
...section 2...
```

The "#"indicates a fragment identifier which is used to address elements within an HTML document. The first problem is that only elements with an `name` or `id` attribute can be addressed by the fragment identifier and consequently only these elements can be addressed. Non-element selections such as a phrase in a paragraph or a word in a phrase are not possible. The second problem is that elements without a `name` or `id` attribute can not be addressed at all. This means the author of a document decides which elements can be a destination anchor for a link or in other words the author can control what readers can refer to [J.D99]. While hypertext should enhance the capabilities of the traditional paper based documents, this is clearly a restriction. XLink together with XPath and XPointer (see chapter 3) eliminate these limitations.

Chapter 3

Linkage in XML Documents

The previous chapter outlined some "missing links" concerning the capabilities of hyperlinks in HTML. The World Wide Web consortium reacted with some new standards centered around the eXtensible Markup Language (XML) . In contrary to HTML, XML defines neither a tag set nor semantics. XML is a meta language for describing markup languages. A tag set in XML is defined by Document Type Definitions (DTD) [Gro98] or in the future by so called XML schemas [Gro00a]. Although, XML does not define a tag set, the tag-set which will be used in the World Wide Web will be a fixed tag set defined in DTDs or in XML schemas. Which tag set will be used will be strongly influenced by the predominant browser vendors.

XLink specifies a way how to indicate that a certain tag is a link. The XLink specification also provides link classes that have proven useful but it permits authors to add arbitrary meta-data to a link. HTML, on the other hand, assigns a fixed behavior to its linking elements. Three specifications form the heart of the XML linking language. XPath, XPointer and XLink. While XPath and XPointer specify how elements and data are addressed within an XML document, XLink specifies how links have to be embedded into XML documents. This chapter deals with the three building blocks of the XML linking language. Their capabilities and usage are shown as well as some problems of theses specifications are discussed.

3.1 Clarification of Terms

3.1.1 Resource

Generally spoken, a *resource* is anything that has identity [BLFIM98]. This might be an electronic document, an image, a video, a device such as a printer or a human being. In the context of the World Wide Web, a resource is is any addressable unit of information or service [BLFIM98] or anything which can have a Uniform Resource Locator (URI).

According to XLink, if a link associates resources, those resources are said to participate in the link. The XLink specification differs between *local resources* and *remote resources* . A *local resource* is an XML element that participates a link but represents the link itself or is a child element of the link. A *remote resource* is a resource that participates a link but is addressed through a URI, no matter if the resource is in the same document as the link or even in the same linking element. These terms are somewhat misleading and should only be used in conjunction with XLink.

3.1.2 Identifier

An identifier is an object that can act as a reference to something which has identity [BLFIM98]. In case of a URI, it is a reference to a resource and consists of a series of characters.

3.1.3 URI, URL, PURL, and URN

The purpose of the URI is to identify resources. The term "uniform"applies to the syntax of the URI which is defined in a way to allow new types of resource locators to be introduced which follow the same syntax. One such type is the Uniform Resource Locator (URL) [TLMM94] which refers to a subset of URIs that identify the resources through the primary access method rather then identifying the resource by name. The most popular access method is the Hypertext Transfer Protocol (HTTP). A URL specifying HTTP access might look like the following:

http://www.iicm.edu

The URL above follows the URI specification, it is an URI schema used to access HTTP services.

Uniform Resource Names (URN) [SM94] provide a globally unique, persistent identifier for a resource. A URN is a name of global scope which does not imply a location. A URL may be used to locate a resource identified by a URN. Such a resource might reside on different locations thus can have more URLs mapped to it. A resource might move to another location without changing its URN whereas its URL will change.

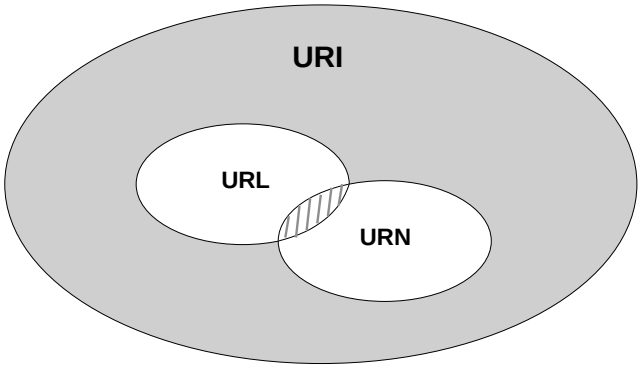

Figure 3.1: Connection between URI, URL, and URN

Figure 3.1 shows the connection between URIs,URLs, and URNs. The outer ellipse shows the Uniform Resource Identifiers. Both URLs and URNs, outlined by the two small inner ellipses follow the specification of URI. URLs and URNs are only two possible types of URNs. The shaded area consists of all URNs which have a URL attached to it. Clearly a mechanism is required which translates URNs to URLs.

3.1.4 Arcs and Traversal

Those three terms are defined in the XLink specification and should only be used in the context of XLink.

The term *traversal* applies to the action of following or using a link for any purpose. The source of the traversal is called the *starting resource* , the destination of the

traversal is called the *ending resource* . Traversal is more than following a link from the user's perspective when navigating through a set of documents. Applications may traverse links transparently to the user when composing distributed documents.

An *arc* gives information how a link should be traversed. This includes the direction of the link as well as application behavior information.

3.2 The Capabilities of XLink

The introduction to the XLink Working draft [Gro00b] describes Xlink as a language which

> *allows elements to be inserted into XML documents in order to create and describe links between resources. For XLink purposes, a link is an explicit relationship between two or more resources or portions of resources.*

The Xlink working group considers not only links between documents but between resources. That means XLink is specified to support linking between various pieces of information, for example sound and movie files, information retrieved from databases, query results etc. An XLink defines an *explicite relationship* between resources.

Xlink allows XML documents to

- define unidirectional and bidirectional links.
- assert linking relationships among two or more resources.
- associate metadata with a link.
- attach links to read-only documents.
- express links that reside in a location separate from the linked resources e.g in a separate XML document or in a database.

Although the XLink itself must appear inside an XML document, the resources associated with the XLink can be of arbitrary type, not just XML encoded ones.

The XLink specification does not determine how links have to be rendered or what action has to follow when a link is activated through a mouse click, for example. The XLink model only describes the structure of links. Higher level applications which process XML documents containing XLinks have to decide on their own how to present XLinks to the user and how to react on link activation (the specification proposes some behavior depending on the type of the link, see section 3.4.1 and section 3.4.3).

3.3 XPath and XPointer

This section describes two XML specifications which improve the capabilities of defining links in XML documents. In short, XLink governs how links are inserted into XML documents, XPointer provides location specification where exactly the link starts and where it ends. XPointer is based on XPath which defines how to address parts of an XML document. Both XPath and XPointer reached the status of a W3C recommendation [Con99], [Con00b].

3.3.1 XPath

XPath defines expressions over the hierarchical structure of XML documents (not the unparsed XML stream). XPath models an XML document as a tree which can have different types of nodes. The root node is, as its name suggests, the root of the tree (the element node of the document element is a child of the root node). Other node types are element nodes , attribute nodes , processing instruction nodes , comment nodes and text nodes (representing the CDATA sections of an XML document). Additionally, there exist namespace nodes which provide information about the namespaces of an element and its attributes.

XPath is used to locate any element or attribute in an XML document. One important expression in XPath is the location expression which consists of the following three parts:

1. An axis, which specifies the tree relationship between the nodes selected by the location step and the context node.

2. A node test which specifies the node type and expanded-name of the nodes selected by the location step.

3. Zero or more predicates which further refine the set of nodes selected by the location step.

Syntactically, a location path expression looks as the following:

```
axis::node-test[predicate]*
```

E.g a child axis would select all child nodes of the context node . The nodeset computed by the axis is then further evaluated by the node-test which yields another nodeset. This nodeset is then filtered by a predicate . For each node in the node set the predicate expression is evaluated with that node as the context node. If the predicate expression evaluates to true, this node is included in the final node set otherwise it is not included. The semantics of the predicate expression depends on the axis specified. Predicate expressions will be explained later in more detail.

The following XML document describes a book with one chapter containing two sections. One section consists of some text together with an image, the other section contains two paragraphs.

```
<book>
  <chapter name="the chapter heading">
     <section name="Section one">>
        The content of section one also contains an image
        <image title="example" href="example.gif"/>
     </section>
     <section name="Section two">
       <p>fist paragraph</p>
       <p>second paragraph</p>
     </section>
  </chapter>
</book>
```

The location path expressions below assume that the element describing the chapter is the context node. A list of all available axes is given in appendix B.

- `child::section[position()=1]` selects section one: The axis specifying the tree relationship is `child`. The node test selects all child nodes named `section`, in this example `section one` and `section two`. The position predicate selects the first of the previously selected children, `section one`.
- `child::section[attribute::name="section one"]` again selects `section one` but this time an attribute is specified as predicate.
- `attribute::name` selects the name attribute of the context node, in this case the name attribute of the chapter node.
- `child::*` will select all children of the context node which are the two section nodes, `section one` and `section two`.
- `descendant::p` selects the two paragraphs: The descendant axis selects all children recursively, the node test selects all paragraph elements.
- `/descendant::section/attribute::name="section one"` selects `section one`. This example shows a combination of location paths: The very first "/" selects the root node as the context node. The root node is the root of the document tree, the book node in the XML document above is a child of the root node. The first axis selects all children of the root node with node name `section`. From these selected nodes, all nodes with the specified attribute value are selected.

An expression in XPath, like the predicate expression, can consist of location steps , function calls, union operators, node sets, relational expressions, and relations grouped together with "or" and "and" operators. There is a minimal set of functions which all XPath implementations like XPointer must provide. Among these functions are node set functions like `position()`, which returns the position of the context node, string functions used to manipulate strings such as `substring(string, start, length)` which returns a substring, boolean functions, and number functions. XPointer enlarges the minimal set of functions (see section 3.3.2). For a complete list of all functions please consult the XPath specification [Con99].

A few examples of valid predicate functions:

- `[position() <= 5]`: evaluates to `true` for the first five nodes in the node set.
- `[position() = last()]` : evaluates to `true` for the last node. The function last() returns the context size of the expression evaluation context, that is the number of nodes in the node set handed over to the predicate expression.
- `[substring(string(),3,4) = "ctio"]` : evaluates to true for each node, which has a substring "ctio" starting at position 3 and length 4 (e.g section)

3.3.2 XPointer

XPath specifies how nodes in an XML document can be located. There is no way defined by XPath to locate text within an element or to define ranges within a document. Besides all the XPath functionality described in section 3.3.1, XPointer allows the specification of `points` and `ranges`. The terms point and range are defined as in the DOM Level 2 specification [con00a]: A point is defined as a node plus an index. The node containing the point is called container node. A range identifies a range of content in a document, document fragment or an attribute. The boundary of a range is defined by a start and an end point. A range is continuous i.e. it contains all the XML structure and content between the start and end point, including both start and end point.

Whereas XPath operates on nodes and node sets, XPointer operates on nodes, points, and ranges and sets of these. Therefore some new terminology is introduced in the XPointer specification: a location is either a node, a range, or an XPath node. A location set can contain nodes, ranges, and points. What was denoted context node in XPath is denoted context location in XPointer.

Defining XPointers

As XPointer is based on XPath, XPointer uses location paths to select locations. Each location-step of the location path selects a location relative to some other well defined location in the document, the so-called location context. The syntax for the location path is exactly the same as defined in the XPath specification (see section 3.3.1):

```
axis::node-test[predicate]*
```

XPointers as well as XPath are not limited to location paths. They can use functions which return location sets :

- The function `id()` is defined in the XPath specification and selects the element in the XML document that has an `id` type attribute with a specified value (see [Con99] for an exact definition of the parameters).
- `here()`: This function is only available within XPointer and returns the node that directly contains the XPointer. Mostly, the returned node will be of type `text` because XPointers mostly occur in CDATA [Gro98] sections of an XML document.
- The function `origin()`: is useful in conjunction with XLink applications. It returns the node of the starting resource of a traversal (see section 3.1.4).

The disadvantage of the `id()` function is that it can only be used if the desired element of the target document has an `id` attribute defined. If no `id` attribute is present and there is no way to modify the target document, the location path has to be used.

Defining Points and Ranges

Sometimes it is necessary to refer to a particular point in a document which does not directly belong to a markup element. Pointing to a certain location in a huge CDATA section is one application of the XPointer's point capability. In conjunction with XLink, ranges allow the implementation of transclusion by providing the possibility of defining exactly what has to be transcluded. Pointing into the text of bad designed markup in XML documents is another application. Consider the following `<date>` element:

```
<date>01 Jan 2000</date>
```

A well designed markup would supply distinct elements for the day, the month, and the year. There is no way in XPath to locate the the first two characters within the `<date>` element as XPath operates on nodes. The underlying data model in XPath for the `<date>` is outlined in figure 3.2.

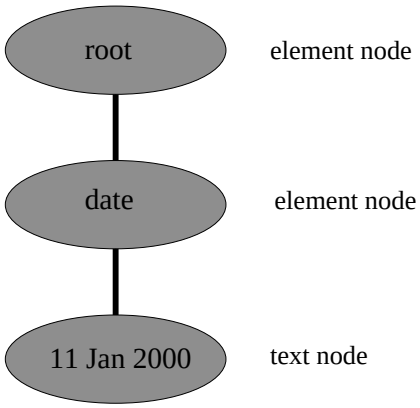

Figure 3.2: XPath Data Model for the <date>tag

The tree consists of three nodes (a root node, an element node and a text node) and there are 16 points in this example:

- The point before the Root node.
- The point before the Date element node.
- The point before 01 in the text node.
- The point between 0 and 1 in the text point.
- ....
- The point after the Date element node.
- The point after the Root element.

If the container node containing the point is of type text, comment, or processing instruction (i.e. a node which can not have child nodes), then the index is between the characters, starting with zero, otherwise the index refers to the point before the

child element. A point is selected using an XPath expression for selecting a node and suffixing it with `/point()[position()=n]`, where n is the index. The following XPointer selects the "d"in date:

```
/child::date/child::text()/point()[position()=0]
```

The XML document below allows more sophisticated examples:

```
<book>
  <chapter name="the chapter heading">
     <section name="Section one">>
       <p>A quotation from Ted Nelson:.....</p>
     </section>
     <section name="Section two">
        ....
     </section>
  </chapter>
</book>
```

The following XPointer points to the paragraph of section one and then selects the "A"in the phrase "A quotation...":

```
/descendant::section[position()=1]/child::p/child::text()/
  point()[position()=2]
```

In some applications it might be useful to select a range rather than a particular point. A typical range an application could handle is the selection of a region by users with a mouse. The boundary points of such a region generally do not exactly match an element node. A region can start in the middle of a text node, span some element nodes, and end up again in a text node. Points allow to cover such regions. Ranges are defined by appending

```
/range-to(end-point)
```

to the location path specifying the start point. The following example defines a range containing the quotation of Ted Nelson in section one of the above XML document:

```
/descendant::p[position()=1]/child::text()/point()[position()=30]
  /range-to(/descendant::p[position()=1]/child::text()/
     point[position()=39]
```

The following range spans section one and section two:

```
/child::section[position()=1]/range-to(/child::section[position()=2])
```

Another useful function concerning ranges are string ranges. String ranges allow to search for a particular string in a location set. For example the following XPointer locates the quotation of Ted Nelson without providing the position within the text node:

```
/child::/string-range(/child::section[position()=1],
   "A quotation from Ted Nelson:")
```

String ranges are an important feature concerning link consistency, as will be discussed in the next section.

3.3.3 Discussion of XPath and XPointer

According to the XPointer specification, XPointer is not designed to be a query language. A location path returning an empty node set is therefore considered to be an error whereas returning a disjoint node set is allowed. The examples given in section 3.3.2 are in fact queries of an XML document. It would be reasonable to define XPointer as a query language and to name it differently. From the point of a query, an empty query bears information and is not considered an error. An application which makes a query can interpret an empty query as an error. XLink can be regarded as an application of XPointer. It should be defined in XLink how to tread XPointers which yield empty results.

Link Consistency

XPointer allows a fine grained location of elements and text within elements. Whereas in HTML it is necessary to supply an id-value to each tag to make it a candidate for destination anchors, XPointer provides axes to locate elements without id-values. This is especially useful if the author of a document did not take into account that parts of the document will be destination anchors of links.

Ranges defined in text nodes are fragile concerning changes of the content. Generally, editing will destroy the content of the range. It's much more complicated to keep links stable which point into text nodes than links which refer to whole documents. This also applies to links to element nodes which have no identifiers. Rearranging element nodes will change their position in the document tree. Axis like following and preceeding will return a different node set. As a consequence, one should assign identifiers to elements and attributes especially if link consistency is desired. For text nodes, the string ranges should be preferred over defining text fragments with location points. As long as the content of the range does not change, editing the text surrounding the range does not effect the XPointer specifying the range. The disadvantage of string ranges is that a string matching algorithm has to be invoked which might consume considerable computing power if the text fragments are large.

MIME Type Support

Defining ranges in streaming data such as audio and video is another useful application. However, references to such data is strongly dependent on the media format. The XPointer specification focuses on XML documents and as a fragment identifier language for `text/xml` and `application/xml` [WIM98]. It's possible to point into CDATA sections, but CDATA sections are not the right place for binary data due to the end delimiter of CDATA sections which can occur in any position in binary data. A possibility to refer to binary data in an XML document is using unparsed external entities or processing instructions [Gro98]. In the XPointer specification there is no proposal how to deal with external entities. XPointer specifications will have to treat external entities differently from other XML elements. External entities are represented as element nodes in the XPath data model. The content of

an external entity is a URL, but what is to be located by an XPointer application usually is not a the URL itself, it is the location the URL refers to. As external entities need special treatment, it would have been a good idea to assign a particular node to external entities.

According to the XPointer specification, different schemes for XPointers can be used to refer to different media types. However the only scheme allowed at the time of this writing is XPointer and users who want to define or use new schemes are invited to check the XPointer errata document which does not yet exist or is not publicly available.

3.4 XLink Concepts

In contrary to HTML, where a link is defined by special tags, an XLink is defined by the use of XLink attributes within arbitrary elements. The attributes must belong to the XLink namespace and follow the constraints defined in the XLink specification [Gro00b]. As a consequence, every XML element may serve as an XLink. An XLink element has to define a type attribute which can have one of the following values:

- `simple:` Defines a simple link which is similar to standard HTML links.
- `extended:` Defines an extended link . An extended link associates an arbitrary number of resources. The participating resources may be any combination of local and remote.
- `locator:` Indicates remote resources of an extended link .
- `resource:` Indicates the local resources of an extended link .
- `arc:` Indicates rules for traversing among participating resources of an extended link .
- `title:` Associates a title to an extended link .
- `none:` Indicates that an element has no XLink specified meaning.

Depending on the type, each XLink element may have other attributes such as from, to, label, role. Please refer to the XLink specification [Gro00b] for allowed combinations of attributes. The types listed above might suggest that there exist eight different link types in XLink, but in reality there are only two, namely `simple` and `extended` links. While simple links may be semantically regarded as a subset of extended links, their structure differs from extended links. The main purpose of simple links is to provide a similar construct to the HTML link tags like `<a>` and `<img>`. Simple links therefore will not be considered any further in this thesis. The locator, arc, title, and resource link types have only XLink specific meaning if they appear as child elements of an extended link.

3.4.1 Extended Links

Extended links provide substantially more capabilities than HTML links: This includes multi-directional links between resources, grouped links, and meta-data assigned to links. An extended link consists of a set of resources and a set of connections between them. Each resource may be either a destination anchor or a source anchor or both. An extended link which only contains remote resources is called an out-of-line link . The following example specifies an extended link with one local resource and four remote resources. It represents a link of a software site which refers to two mirror sites (see figure 3.3).

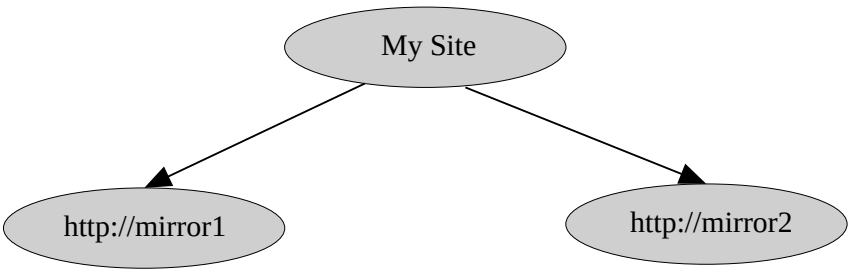

Figure 3.3: An extended link with one local and two remote resources

A child element of type `locator` describes the remote resource, a child element of type `resource` describes the local resource which is confusing. A different naming would make the intended behavior more apparent. The following document type

definition defines the extended link outlined in figure 3.3.

```
<!ELEMENT mirror_link (server, mirror*, traversal*) >
<!ATTLIST mirror_link
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
  xlink:type (extended) #FIXED "extended"
  xlink:title CDATA #IMPLIED
  xlink:role  CDATA #IMPLIED
>

<!ELEMENT server (#PCDATA)>
<!ATTLIST server
  xlink:type (resource) #FIXED "resource"
  xlink:title CDATA #IMPLIED
  xlink:label CDATA #IMPLIED
>

<!ELEMENT mirror (#PCDATA)>
<!ATTLIST mirror
  xlink:type (locator) #FIXED "locator"
  xlink:href CDATA #REQUIRED
  xlink:title CDATA #IMPLIED
  xlink:label CDATA #REQUIRED
  xlink:role  CDATA #IMPLIED
>

<!ELEMENT traversal
  xlink:type (arc) #FIXED "arc"
  xlink:from  CDATA #REQUIRED
  xlink:to    CDATA #REQUIRED
  xlink:show  CDATA #IMPLIED
  xlink:actuate CDATA #IMPLIED
>
```

The extended link has three child elements of type `resource`, `locator`, and `arc`. An `arc` element indicates rules for traversal of the participating resources of an extended link. In the DTD above, the `<traversal>` element is of type `arc` and may have four attributes. The `from` attribute defines which resource the `arc` comes from, the `to` attribute defines the destination of the `arc`. Labels must be supplied to locate the corresponding resources. The `show` and `actuate` attributes of the arc element are so called behavior attributes. Applications should (not must!) interpret the behavior attributes according to the XLink specification. For example if the value for the `show` attribute is `replace`, the application should load the destination resource in

the same window or frame. The value `onRequest` for the actuate attribute tells the application that the application should traverse the link only if a certain event occurs, e.g. clicks on the graphical link representation by users. Possible values for the `show` attribute are `new`, `replace`, `embed`, `other`, and `none`, possible values for the `actuate` attribute are `onLoad`, `onRequest`, `other`, and `none`. For a detailed description how applications should interpret these values please consult the XLink specification [Gro00b].

The resource and locator typed elements may have the semantic attributes named `title` and `role` . The value of the `role` attribute must be a URI that references some resource which describes the intended property. There is no description in the XLink specification how this information has to be supplied. The title attribute describes the meaning of a link or resource in a human readable fashion. The following fragment shows how an extended link described in the preceeding DTD can be used in an XML document. Figure 3.3 illustrates the extended link described by the XML fragment:

```
<mirror_link xlink:type="extended" xlink:title="Mirror Sites">

   <server xlink:type="resource" xlink:label=''my_site''
       My Site
   </server>

   <mirror
     xlink:type="locator"
     xlink:title="Mirror1"
     xlink:href="http://mirror1"
     xlink:label=''mirror1''
   />

   <mirror
     xlink:type="locator"
     xlink:title="Mirror2"
     xlink:href="http://mirror2"
     xlink:label=''mirror2''
   />

   <traversal
     xlink:type="arc"
     xlink:from="my_site"
     xlink:to="mirror1"
```

```
    xlink:show="replace"
    xlink:actuate="onRequest"
  />

  <traversal
    xlink:type="arc"
    xlink:from="my_site"
    xlink:to="mirror2"
    xlink:show="replace"
    xlink:actuate="onRequest"
  />

</mirror_link>
```

3.4.2 External Link Databases

Link databases ease the management of links as discussed in section 5.3.3. Out-of-line links, that are links which only have remote resources, can be stored in link databases also called linkbases . According to the XLink specification, the linkbase must be a document conforming to the XML specification. The following example shows the out-of-line link corresponding to figure 3.3:

```
<mirror_link xlink:type="extended" xlink:title="Mirror Sites">
 <server xlink:type="locator" xlink:label=''my_site''
    My Site
 </server>

 <mirror  xlink:type="locator" xlink:title="Mirror1"
          xlink:href="http://mirror1" xlink:label="mirror1"
 />

 <mirror  xlink:type="locator" xlink:title="Mirror2"
          xlink:href="http://mirror2" xlink:label="mirror2"
 />

 <traversal xlink:type="arc" xlink:from="my_site"
            xlink:to="mirror1"  xlink:show="replace"
            xlink:actuate="onRequest"
 />

 <traversal xlink:type="arc" xlink:from="my_site"
            xlink:to="mirror2" xlink:show="replace"
```

```
            xlink:actuate="onRequest"
 />

</mirror_link>
```

The difference to the previous example is that this extended link has only remote resources denoted by the `locator` type. There are some open questions concerning this example:

1. On loading the XML document, how is the corresponding linkbase located by an application?
2. Where should the link be located in the corresponding XML document ?

The currently defined way to locate a linkbase is to include an extended link into the document which refers to the linkbase. This, however, requires the author of the linkbase to have write access to the document. It is likely that this will be subject for change in the XLink specification.

XPointer provides a convenient way to locate the element in the document which should represent the link. The specification is superficial of how this has to be defined. Again it is proposed to include the information about the location of the element in the document itself, not in the document of the linkbase. This is somewhat similar to style sheets and HTML documents where one part of the style information is placed in the style sheet, another part is placed in the document itself. It seems that concerning out-of-line links, the same mistake is made again.

The following example shows an external linkbase. The first XML document given shows some text and locates the link base:

```
<?xml version="1.0">
<linkbase xlink:href="glossary_linkbase"/>
<example>
  <body>
     <h1>Hyperlink Fundamentals</h1>
     <p>.....hyperlinks......hyperlinks...</p>
  </body>
</example>
```

The linkbase contains one link which maps every occurrence of the term “hyperlink” to the glossary entry “hyperlink”. The glossary is also an XML document (not shown here).

```
<?xml version="1.0">
<linkbase>
  <link_to_glossary xlink:type="extended"
    <source xlink:type="locator"
            xlink:title="Glossary Entry"
            xlink:label="hyperlink_source"
            xlink:href="#/::/string-range(/::text(), hyperlink)"
    />

    <to_hyperlink_entry xlink:type="locator"
          xlink:label="hyperlink_glossary"
          xlink:href="glossary.xml#/id(hyperlink)"
          xlink:label="hyperlink"
    />

    <traversal xlink:type="arc"
               xlink:from="hyperlink_source"
               xlink:to="hyperlink_glossary"
    />
  </link_to_glossary>
</linkbase>
```

3.4.3 Discussion of XLink

Simple Links

Currently, XPath is a W3C recommendation, XPointer and XLink are W3C candidate recommendations. The XLink requirements document was released in February 1999 [Gro99b]. One of the design goals was that XLink has to be prepared quickly. This design goal has not been met. Simple links were introduced hoping that applications would use them before the more powerful and complex extended links. As simple links just provide minor enhancements to HTML links (they can have semantic and behavior attributes), why should one use them? If the XLink specification will be accepted by developers, simple links will play a minor role as they can be represented by extended links too. However, it is likely that simple links will remain part of the specification.

Behavior Attributes

The XLink specification does not say anything about the presentation of links. This follows the XML credo of separating structure and presentation and is a major enhancement to HTML. The behavior attributes which might be attached to both source and destination anchors of a link are controversial. One might argue that how a link behaves is intrinsic to the link and is part of the document structure. A link which should be automatically embedded imposes a different structure than a link which should be opened in a separate window on demand. On the other hand the behavior of a link also depends on the viewer. A link with an attribute of the value `embed` might on some systems embed the referred document, on other systems this might be unwanted due to bandwidth limitations. Another point is that the specification explicitly mentioned that applications should (not must!) follow the recommended behavior. It might well happen that the predominant browsers interpret the behavior attribute differently. One way would be to exclude the behavior attributes from the XLink specification and to put them into stylesheets. Viewer dependent stylesheets could then optimize link presentation and link behavior according to their needs.

Another point to consider is that the behavior attributes might be well suited for the World Wide Web at this time. But XML is intended to be a format for data exchange for many areas of application. As the name states it is extensible and should be well suited for future applications. XLink should follow the same philosophy. Of course it is possible to assign a value of `other` or `non` to the behavior attributes to indicate the the intended behavior is not covered by the specification. But why then define the behavior attribute as part of the XLink specification at all?

Linkbases

One strong feature of XLink, the linkbase , is not well defined in the XLink specification yet. To locate a linkbase, a location specifier has to be included into the XML document. If access permissions deny editing of the document, it is impossible to associate a linkbase with this document. Different linkbases to one document (see section 2.4.4) would require changing the document each time a new linkbase

should be loaded. While the specification respects the need of linkbases, the way applications or hypermedia systems should access them is not well defined.

The Impact of XLink on the World Wide Web

In contrary to HTML, where special tags define a link, in XLink every object is a candidate for a link. Attributes according to the XLink specification define whether an element supplies XLink functionality. Besides navigation, links will become more and more a tool to define the structure of documents as each element defining the structure can also serve as a link. Though it is not the task of a link service (see chapter 4) to construct documents, link services will be tightly coupled with document processors.

Linkbases will play a minor role concerning the creation of dynamic links (see section 2.4.3) which are mostly created to link the results of database queries. Links created this way require a highly standardized document format like e.g. journals and magazines provide. Linkbases are useful in the case of unstructured data where automatic generation of links via a program or a script is difficult. Linkbases also ease the implementation of consistent and personalized links.

The ability to assign meta data to links will increase the usability of links in the World Wide Web. Although the specification of XLink does not describe how links have to be rendered in applications such as browsers, the meta data will enable applications to render links according to their intention: cross references, links within documents, footnotes, or annotations can be rendered differently and give users a clue which semantic was intended by the author of a link.

The XML linking language allows a fine grained placement of source and destination anchors. This has an impact on navigation as well on the composition of documents. Precise navigation is possible as text within an element can be addressed. XLink allows the specification of ranges within a document which makes transclusion feasible. It is likely that transclusion will occur more frequently when composing a document.

The range of new features also has one drawback: The XML linking language is more complex than the link facilities provided by HTML. To understand XML linking,

it is not enough to understand XLink, one also has to study XPointer and XPath. One of the reasons of success of HTML was its simplicity which allows people to write documents without the need of any special tool. To exhaust the features of XLink without a special tool will be quite cumbersome. However, at the time of writing this thesis, there do not exist any tools that go beyond the generation of simple links.

Chapter 4

Distributed Link Services (DLS)

The previous chapter described XLink which defines how links have to be defined in documents. In addition to links embedded in documents, XLink allows out-of-line links which can be stored in linkbases, separated from the documents. Out-of-line links ease to maintain link consistency because links can also be defined and changed without having to revise the concerned documents. This implies that links can be defined for read-only documents (see section 5.3.3 and section 2.4.2 for the benefits of linkbases). The program dealing with linkbases is called a "link service". A link service treats a link as an independent object. It may be distributed across databases which in turn may be distributed across various servers. In this case, a distributed link service (DLS) maintains the link base. The kind of DLS described in this section is not a link service which provides link collections of a specific subject to users like search engines do. The DLS treated in this section deals with links which are stored in a linkbase separated form the resources they belong to.

The issues discussed in this section are based on experiences of various research activities on electronic libraries [CRH] and hypertext systems [CHH98, Pea89, RS97, Mau96] and the Dinopolis project [Dal99]. The DLS of the Dinopolis system will be discussed in chapter 5.

4.1 Fundamentals of Link Services

Figure 4.1 illustrates a document server which outsourced link management to a link server. The links managed by the link server are stored in a database of its own, separated, but not independent of the document database. The two databases are not independent because changes of the document location and content requires synchronization with the link database.

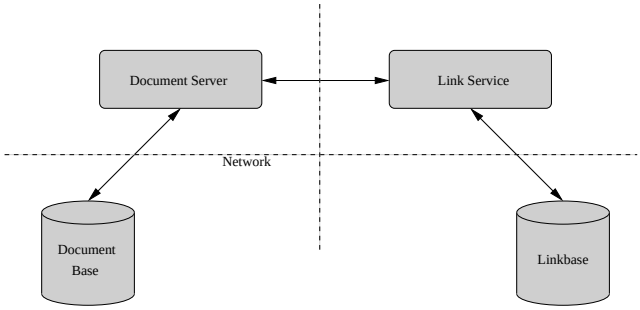

Figure 4.1: A simple link service

4.1.1 Link Maintenance

The synchronization between the document server and the link server is necessary to keep the link consistent. E.g. if one of the source/destination anchors of a link is deleted, the link becomes invalid. If a document or part of a document is moved to a different location, the source/destination anchors of the concerned links change. There are several ways a link service can detect changes of anchors:

Garbage Collection

The link service provides an link garbage collection mechanism for discovering dangling links : the garbage collector periodically goes through all the links in the database and checks whether the links are valid. The advantage of this approach is that it works well in an heterogenous environment because the link service in this case queries the managing application of the destination anchor of the link. The disadvantages are:

1. Dangling links are not discovered immediately. It can still happen that a user follows an invalid link before the garbage collector has detected it.
2. The garbage collector has to check all links over and over again, even if the links did not change. In a huge link database, the time the garbage collector needs to check all links might become considerably long.
3. The garbage collector detects dangling links but does not avoid them.

Closed hypertext systems usually do not check links which point outside the domain of the system. A garbage collector is the only way to detect dangling links in a heterogenous environment as long as no standard protocol for link maintenance exists and there is no practical method to keep links up to date, if the resource the link points to changes its location.

Implicit Detection of Dangling Links

In this case, a user tries to follow a dangling link. The document server tries to fetch the document the link points to and discovers that the link is dangling . The document server informs the link service of the invalid link which in turn deletes the link. The disadvantage of this method is that users have to detect dangling links which is clearly not the intention of a link service.

Event Driven Link Maintenance

Closed hypertext systems have more sophisticated possibilities to maintain their links. The Dinopolis system described in chapter 5 uses an event mechanism to keep the links consistent and to avoid dangling links. Every time, a resource is moved or deleted, the link service is informed about the changes and updates its link database accordingly (see section 5.3) .

4.1.2 Other Issues of Link Services

Response Time

A mechanism has to be found to optimize the update of the database if the number of involved links is large. Consider the case of moving a heavily referenced document to another location. This might require the update of thousands of links. In case of a distributed link service, these links may be spread accross different linkbases, some might have a bad response time due to network overload or even might be temporarily unreachable. However, this must not affect the move operation. The Dinopolis system addresses this issue and uses a forwarding mechanism. For details on the forwarding mechanism please see section 5.3.

Standardized Protocol

In an open hypermedia environment, a standardized and open protocol is necessary to provide sophisticated link management. A proprietary protocol will make data exchange between link services of different vendors a difficult task. The internet domain name service is an example of a service using a standardized protocol which perfectly works over years in the heterogenous environment of the World Wide Web. NNTP or SMTP are other examples. While XLink is close to become a standard, the standardization of a protocol for a distributed link service is still far away. As a consequence, link management currently is a subject of closed hypertext systems which all use different and proprietary protocols.

4.2 Communication with the Client Application

Client applications such as browsers should be independent of the link service. While this is not necessarily important for a closed hypertext system which provides special user agents, this is clearly an issue of an open hypertext systems with millions of users like the World Wide Web. Changes of the DLS could affect millions of clients which would then have to be updated. The prerequisite to make client independent from the DLS is that a standard for link definition like XLink exists and is implemented

both by the client and by the DLS. Currently, most hypertext systems which offer advanced link capabilities suffer from the lack of a standard. As a consequence, the representation of out-of-line links is proprietary and are closed to other hypertext systems.

Figure 4.2 and 4.3 outline how a client application can have access to links stored in a linkbase. In figure 4.2 the client first requests the document and then loads all links belonging to the document. The location of the linkbase might be provided by the user's preferences or, as described in section 3.4.2, might be part of the document. This approach has several advantages:

- The server providing the document is not burdened to also provide the correct links from the linkbase.
- The client application can freely choose a linkbase, e.g. a private linkbase stored on the client side.
- Depending on the user's actions, the client application can load links on demand, it does not have to fetch all links belonging to a document in advance.
- Users can subscribe to third party linkbases .

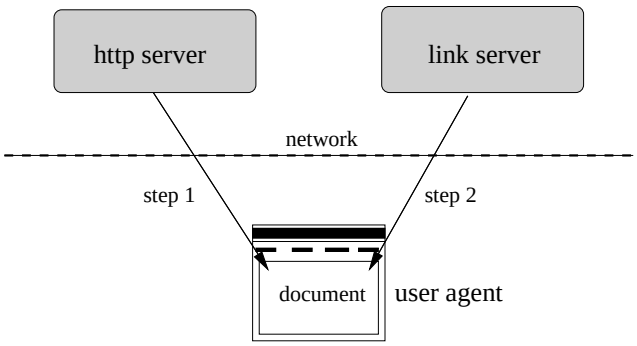

Figure 4.2: The user agent is aware of a linkbase

In figure 4.3, the client connects to the HTTP server as usual and the server fetches all required links from the link server. In this approach, the client need not be able to communicate with the linkbase server, no additional protocol must be implemented

on the client side. Furthermore, content servers which require authentication can also provide private linkbases.

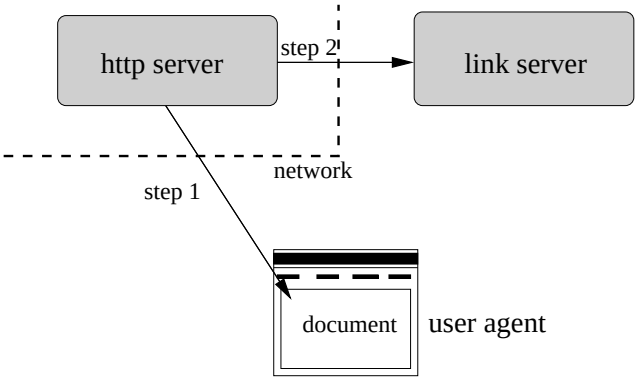

Figure 4.3: The user agent is completely uncoupled from the linkbase

It is likely that both approaches will coexist. While the first approach is more flexible for the user, the second approach requires no change on the client application. Certain links could be processed by the HTTP server and might be hidden from the client application. Regarding Xlink, a client application which fully implements the XLink standard, has to know how to deal whith out-of-line links which have to be placed in a valid XML document. An XLink enabled browser is able to handle the method outlined in figure 4.2.

Chapter 5

The DLS of the Dinopolis Project

This chapter first explains the general concepts of the Dinopolis middleware system and provides the necessary background to understand the requirements on the DLS which are then elaborated in detail. The implementation section focuses on the link forwarding mechanism which dramatically increases performance if a huge number of links have to be updated.

5.1 The Dinopolis Project

The development of Dinopolis started at the IICM in 1996. The initial goal was to create a messaging system. Applications could insert events in a central message queue and listen to that queue to retrieve messages of other applications. While this concept worked well for small systems, it did not scale on large systems as the amount of messages increased. The experiences with this first version of Dinopolis led to a complete redesign with new goals in mind [CHKZ00]. The new design goes far beyond a simple messaging system. In contrary to other hypermedia systems, the Dinopolis system focuses on incorporating already existing systems and to provide a single point of access to users. Figure 5.1 shows the three tier architecture [Sad97] of the Dinopolis system. The advantage of this approach is that client application

programmers access the underlying servers with a uniform application programming interface (API) .

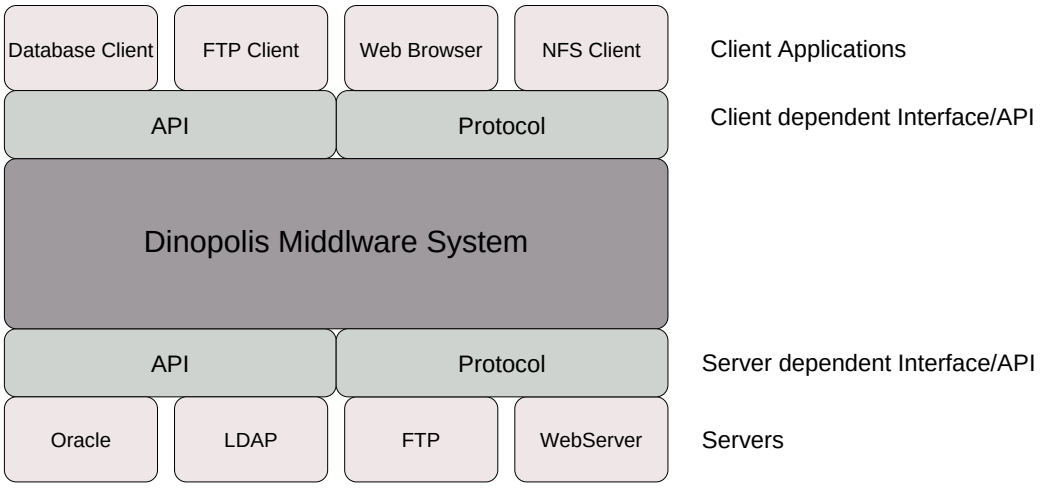

Figure 5.1: The middleware architecture of Dinopolis

Without the middleware layer, programmers are faced with different APIs depending on the system they want to access. Other significant advantages of the three tier architecture are:

- changes of the server dependent protocols and APIs do not affect the client applications.
- client applications are independent of network and service infrastructure decisions which gives system managers more flexibility [Ber99].
- additional capabilities can be added to the underlying systems (e.g networking, user management,...).
- the middleware layer allows cooperation between different, initially isolated systems.

The Dinopolis system exploits the advantages of a middleware architecture . The middleware layer is a data- and communication-model that allows embedding of various system without cutting down their functionality. It provides unique object

addressing and a sophisticated link service to the client applications. Open and well defined interfaces supporting user management and security are provided that can be implemented depending on the users' needs. Dinopolis is designed to be an open system, hence it is based on widely accepted standards.

For a detailed description of the design of Dinopolis please read [Dal99]. The user management is described in [Hau99]. Since 1999, the Dinopolis system is developed under the GNU public License, the developing process can be traced on the Dinopolis Web Server[1]. Additionally, detailed technical documentation and all related work is provided on this site.

5.1.1 Clarification of Terms

The terms defined in this section are commonly used in conjunction with the Dinopolis project.

- *embedded system:* a system which is accessed by client applications via the Dinopolis middleware layer is called an embedded system .
- *relation:* in the dinopolis system, a hyperlink defines a relationship between arbitrary resources. To emphasize this model, the term relation is used instead of the term hyperlink.
- *Dinopolis object:* a Dinopolis object is an encapsulation of a resource of arbitrary type provided by an embedded system. The content of a Dinopolis object is described by the use of XML. Besides the content of the resource, a Dinopolis object provides all methods to access and manipulate the resource. Every Dinopolis object has a unique address and can therefore be the source or target of relations.
- *Dinopolis application:* a Dinopolis application accesses the embedded systems via the Dinopolis middleware layer by using the Dinopolis API .
- *External Access Gateway (EXAG):* the purpose of an EXAG is to provide alternative APIs along with the Dinopolis API which allow applications to

[1] `http:/www.dinopolis.org`

make use of the Dinopolis system without supporting its API. An example would be an HTTP EXAG which enables WWW browsers to access embedded systems and make use of the Dinopolis' features.

5.1.2 An Exemplary Use Case

Before digging into the structure of the Dinopolis middleware system, a concrete area of applications is shown which exploits the power of the Dinopolis architecture. Consider an application which manages medical documents as shown in figure 5.2.

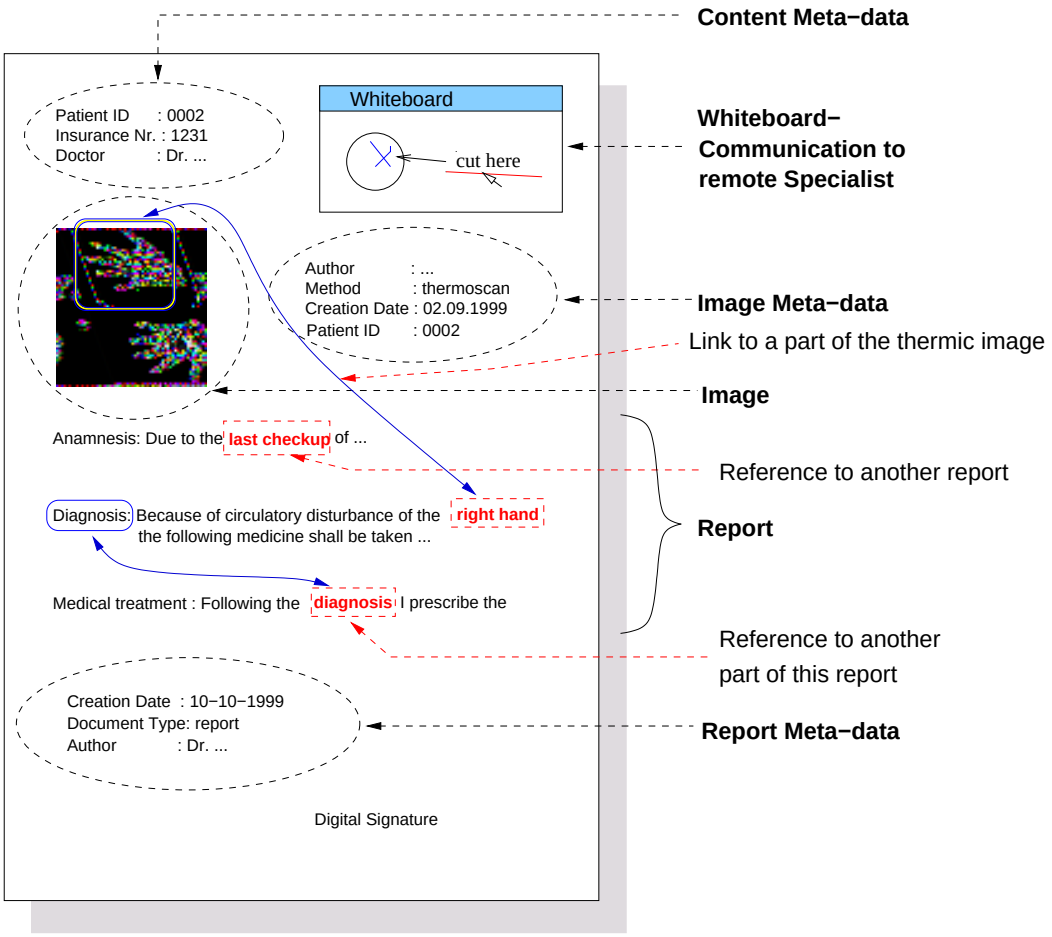

Figure 5.2: A medical document composed of various parts

The document is composed of a medical report, an image showing a thermic diagno-

sis, and a whiteboard for communication with a remote specialist. The whiteboard indicates that documents managed by Dinopolis also can include active content. The various parts the medical document is composed of, are distributed among various systems: the medical report including its meta-data are stored on the local filesystem, whereas the thermic image and its meta-data reside in a Hyperwave information server. The meta-data describing the overall content, the digital signature, and all relations are kept in an Oracle database. The user management of the application uses the X.509 certificates stored in an LDAP server.

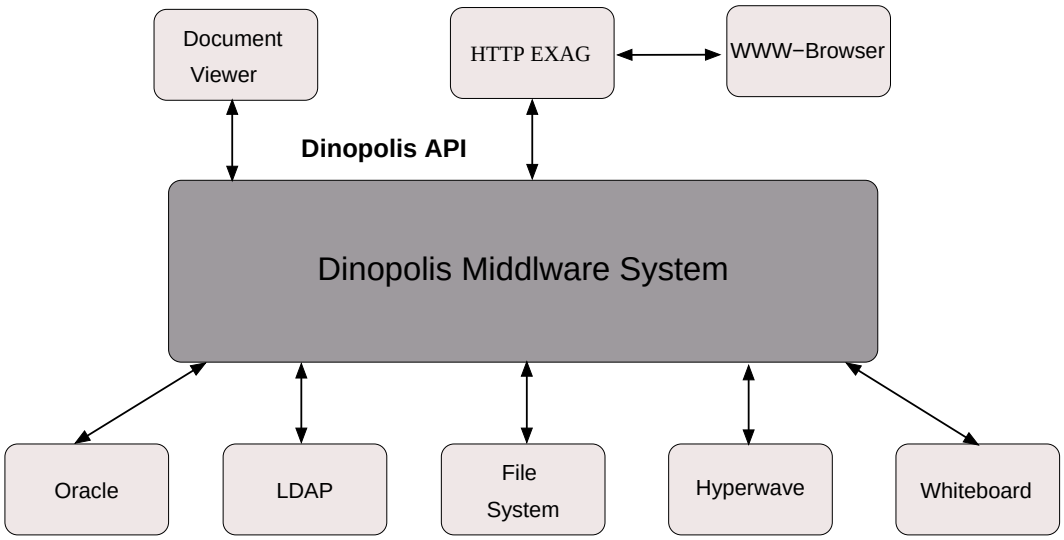

Figure 5.3: An example of a Dinopolis system integrating a variety of server systems and providing the content to different clients

Figure 5.3 shows the configuration of the Dinopolis system which handles the kind of medical documents described above. The document viewer access all the servers through the middleware layer by using the Dinopolis API . The WWW browser accesses the servers through the HTTP EXAG which in turn makes use of the Dinopolis API. Both applications make full use of the features of the Dinopolis system (e.g. user management) although the WWW browser is not aware of the Dinopolis system. This example outlines one of the strengths of a middleware architecture: Applications can take advantage of the middleware API which provides uniform access to the embedded servers. Conventional corporate intranet solutions

also provide a single point of access, but this solution is mostly based on servlets or CGI scripts that access the various servers directly, hence have to deal with different protocols and APIs.

The Dinopolis system has many different areas of applications. The power of Dinopolis in distributed teaching environments is examined in [DHK+00b, DHK+00a]. Further examples are described in the Dinopolis user requirements document [IIC99].

5.1.3 The Dinopolis Architecture

The developers of Dinopolis followed four main design goals [CHKZ00]:

- Dinopolis has a extensible and modular architecture . It has to be extensible to be able to easily react on new technologies. The modular design eases code maintenance. The base system providing the data- and communication-model is a slim piece of software rather then a huge monolith.
- Dinopolis is designed to be open to industry standards. Proprietary protocols are only used where absolute necessary and are hidden to client applications.
- Dinopolis systems can be distributed across a network, fully transparent to the client applications accessing the underlying systems. All Dinopolis objects can be accessed by a unique handle .
- Dinopolis is designed to be platform independent.

Dinopolis is written in pure Java. Java was preferred to other programming languages because

1. Java allows to write platform independent code.
2. Java supports dynamic classloading.

Dynamic classloading is the key for developing a slim core Dinopolis system. All additional functionality required to embed external systems is grouped around the core system and is loaded by the use of factories [GHJV98]. The layered design is

outlined in figure 5.4. Each module in the layer model has a well defined interface to its adjacent modules. If the implementation of one layer changes, other layers are not affected as long as the interfaces remain untouched. Thus the definition of the interfaces is a crucial part in the design of the system.

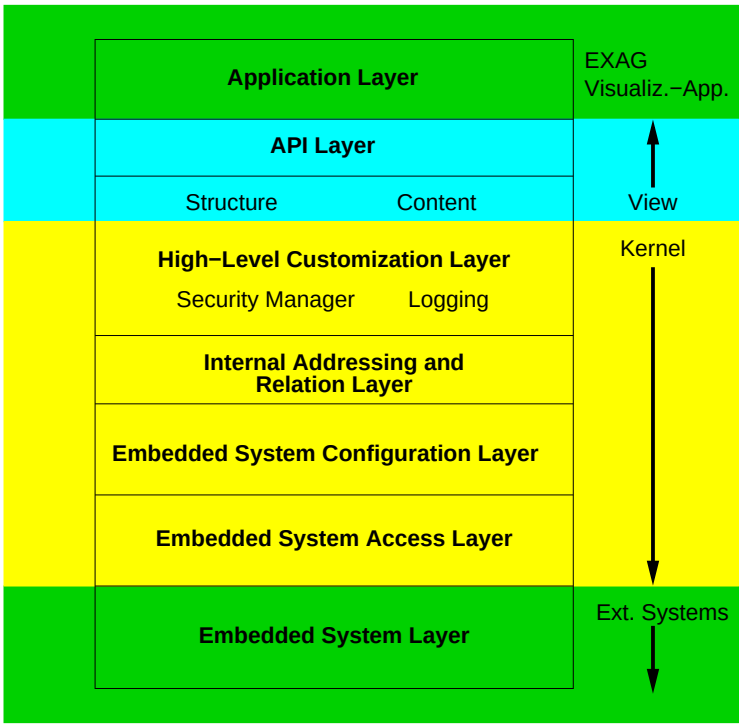

Figure 5.4: The layered architecture of the Dinopolis system

Embedded System Layer

All external systems are part of the embedded system layer. The only requirement on the external systems is that they must provide an API or a protocol in order to access them.

Embedded System Access Layer

Each embedded system has a low-level driver (also called "system embedder") which is part of the Dinopolis kernel and loaded via the use of factories. This layer provides

an abstraction of external systems to the layers above. The task of the low-level drivers is to map the different APIs or protocols of the external systems to a common API which is used by the upper layer to access the external systems. This also includes mapping the address model of the external system to the hierarchical addressing scheme used by Dinopolis and to provide all relations according to the Dinopolis relation model.

Embedded System Configuration Layer

One of the strengths of the Dinopolis system is to improve the capabilities of external systems by combining them. This layer provides the mechanism necessary to combine systems and to present combined systems as a single system to the above layers.

Internal Addressing and Relation Layer

This layer provides a virtual addressing and relation layer to all Dinopolis applications and EXAGs. The addressing scheme is hierarchical and its purpose is to provide unique handles to access all Dinopolis objects. Navigation in the object space is achieved through relations. Whereas the data model of the address space is a distributed tree and hence hierarchical, relations provide arbitrary relationships between Dinopolis objects.

High-Level Customization Layer

The log- and security-manager are parts of this layer. They have access to all the communication between the addressing and relation layer and the API layer. The implementation of the security manager and log manager is not part of the Dinopolis base system. Interfaces for both the security manger and the log manager are provided which can be implemented according to the users' needs. This allows not only to build a flexible security manager, but also to incorporate the existing security models of embedded systems.

API Layer

The API layer is single point of access for all Dinopolis applications and EXAGs. The functionality of the API comprises administration of the Dinopolis system, access to Dinopolis objects, and mechanisms to manipulate these objects. Additionally, the API provides different views to the virtual addressing and relation space. Views allow personalization of the information system (see also personalized links, section 2.4.2).

Application Layer

All Dinopolis applications as well as EXAGs reside in this layer. EXAGs follows the philosophy of Dinopolis to be an open system thus allowing access not only to Dinopolis applications but to a variety of existing and future applications which do not support the Dinopolis API.

5.2 Requirements on the Dinopolis Link Service

The requirements on the Dinopolis link service can be partially deduced from the properties modern hypermedia systems have to provide (see chapter 2) and from the special architecture of the Dinopolis system outlined in the previous section. The ability to embed external systems which provide their own relations means that the DLS is not responsible to manage all links of a Dinopolis system. This leads to the definition of immanent and explicit relations:

- *immanent relations* are provided by external systems. From the point of view of the Dinopolis system, these relations are persistent (even if they are dynamically created, see section 2.4.3).
- *explicit relations* are relations attached explicitly to a Dinopolis object and the embedded systems the object belongs to are not capable to handle the type of the attached relation. These relations have to be managed by the Dinopolis DLS.

As an example consider a hierarchical filesystem which per-se has parent-children relationships because of its storage structure. The system embedder of the filesystem is responsible to provide this kind of relation. If a user annotates a file, it is the task of the DLS to manage this type of relation because a filesystem generally is not able to handle annotations.

The requirements presented below affect both the functionality of the DLS as well as the representation of relations.

R 1: Consistency of Relations

Modern hypermedia systems must support consistent relations to avoid dangling links and to ease the administration of documents. If a relation between two or more Dinopolis objects is defined, it has to be valid and point to the correct destination even if the location of any involved Dinopolis object changes. If one endpoint of a relation is deleted, all relations belonging to this object have to be removed. If a Dinopolis object is moved, all involved relations have to be updated accordingly. In case of a copy operation, things are a little bit more complicate if the copied Dinopolis object is a container object (i.e is composed of objects). Several issues have to be considered:

- all relations within the container object must still be valid which requires an update.
- implicit relations will change depending on the type of the embedded systems affected.
- the treatment of all explicit relations cannot be determined by the system. In some cases, the application (which in turn may ask its user) must decide how to deal with these relations.

Consistency of relations is not required if the source or destination of a relation is beyond the scope of the Dinopolis system. This applies to relations concerning systems which are neither embedded nor support an open protocol to maintain links. An example would be a relation pointing to an ordinary web server. In this case, an

implicit detection of dangling links is required (see section 4.1.1)

R 2: Relations are Bidirectional

All implicit and explicit relations are bidirectional. Bidirectional relations are required because:

- link consistency without bidirectional links is hard to achieve. Consider the following case: a unidirectional relation between two objects is defined, say from object A to object B. If object B is moved, it is hard to compute the inverse linking function (see section 2.2) of the relation. The only possibility is an exhaustive search over all objects which is clearly not efficient.
- relations have properties imposed by embedded systems or by applications. Providing only unidirectional relations would make the embedding of external systems which are based on bidirectional links a burden.
- navigation in the Dinopolis system is done via relations. Bidirectional relations ease forwards and backwards navigation.
- from the authors' point of view, it might be interesting which other users reference a certain document.

In case of implicit relations, which are provided by the embedded systems, only relations of Dinopolis objects which are already part of the address space have to be considered. If the embedded system only maintains unidirectional relations and is not capable to transform them into bidirectional relations, the Dinopolis DLS has to transform these relations as far as possible.

R 3: Relations have Meta-Data

The meta-data of a relation contain the semantic information embedded systems and applications need to interpret and to render the relation accordingly. Since the Dinopolis system is a middleware system and it can not be predicted which systems will be embedded in the future, it is not possible to define a fixed set of meta-data.

Meta-data are also required to achieve personal relations which form the basis of the "View" concept of the Dinopolis system. Depending on the type of the relation, a view decides to show or to hide a relation to an application as outlined in figure 5.5.

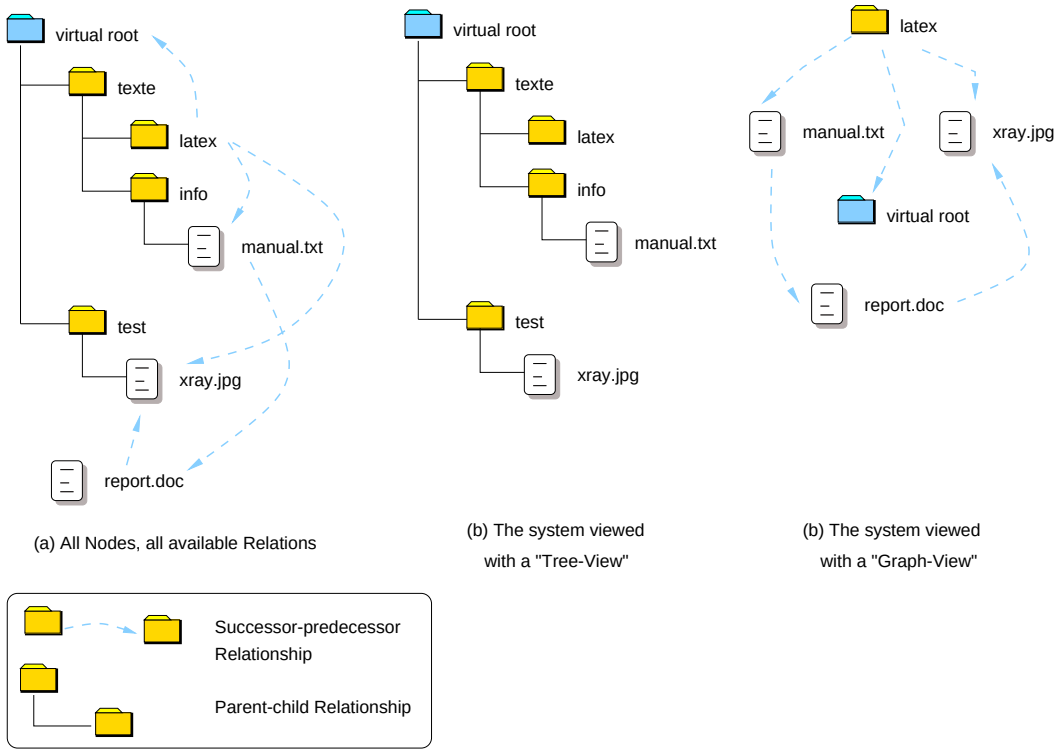

Figure 5.5: Different views depending on the relations' type

R 4: Relation Management has to be efficient

As relations have to be kept consistent, an efficient relation maintenance technique is required. A move or delete operation of a Dinopolis object which is referenced by millions of other objects must be carried out in an acceptable period of time. These operations might involve connections to remote hosts that may be temporarily unreachable. Appropriate algorithms have to be implemented which lead to an acceptable response time.

R 5: Support of external Linkbases

The Dinopolis DLS must be able to incorporate external linkbases from third parties. This applies to all linkbases which follow an open standard. The support to incorporate external linkbases is required because:

- special purpose linkbases on special topics may be provided.
- linkbases will be delivered along with a set of documents, e.g on a multimedia CD-ROM all links may be stored in a linkbase.

UR 6: Support for multidimensional Relations

Multidimensional relations (one-to-many, many-to-one and many-to-many) have to be supported because:

- embedded systems might support multidimensional relations
- it is convenient for authors to define multidimensional links (see the mirror-site example in section 3.4.1.

R 7: Relations can be defined between Relations

At the first glance, it might not make much sense to define a relation that references another relation. But consider the case outlined in figure 5.6: Imagine that a page shall be referenced which in turn keeps a reference to the most important link page to a special topic. The authors of this page always update the link as soon as they find a more important link page. As the important link page should be referenced directly, a relation pointing to the relation that references the link page is the perfect solution.

R 8: Standardized Representation of Relations

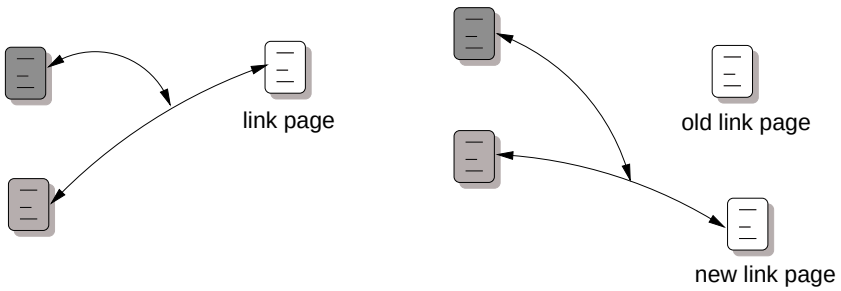

Figure 5.6: A relation automatically follows another relation

A widely accepted standardized representation of relations has the following advantages:

1. clients that adhere to the standard can access the representation efficiently because no transformations are necessary.
2. incorporating linkbases based on the standard is possible without transformation.
3. tools from third parities based on the standard can be used.

5.3 Implementation

This section describes some implementation aspects of the Dinopolis DLS. First the model of relations in Dinopolis and its representation through XLink is shown . The last two sections focus on the techniques used to keep relations consistent and on the communication between link services.

5.3.1 The Relation Model of Dinopolis

A relation in the Dinopolis system consists of a relation object and two or more so called “pin” objects. There is a bidirectional relationship between the pin object and the relation object. Pin container objects are attached to Dinopolis objects which hold all pins belonging to the object. Figure 5.7 shows relations between three

Dinopolis objects (the small white rectangles within the pin container symbolize the pin objects).

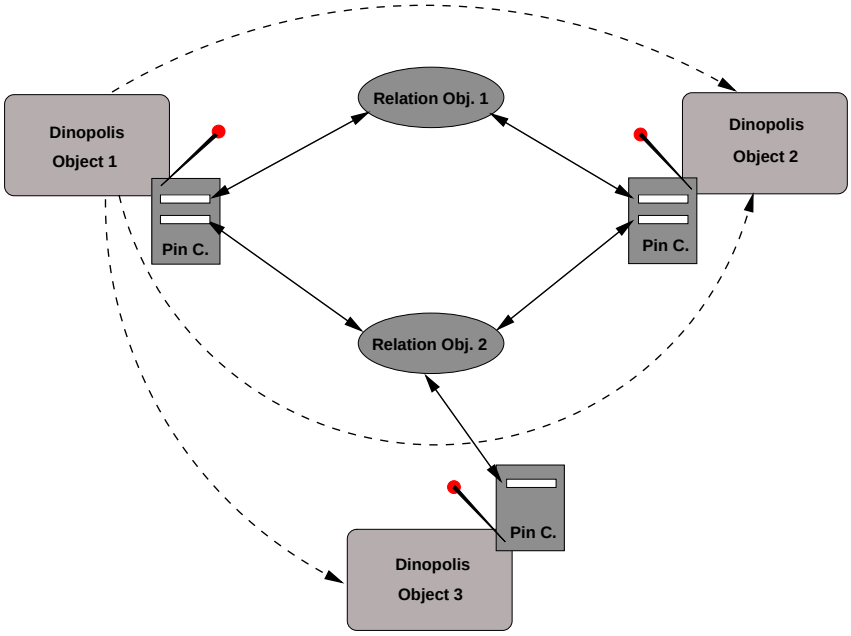

Figure 5.7: Example of relations between three Dinopolis objects

Object 1 and object 2 have two pins in their pin container, object 3 has only one pin. Although there exist only two relation objects, three bidirectional relations are defined:

1. between object 1 and object 2.
2. between object 1 and object 3.
3. between object 2 and object 3.

The relation object and both pin objects can have their own set of meta-data. The relation object 2 in figure 5.7 is a one-to-many or a many-to-one relation depending on the type of the relation. The dashed lines in figure 5.7 show a possible behavior of the relations according to their meta-data. In this case, the relation object 2 represents a one-to-many relation. There are two relations between object 1 and object

2, both pointing to the same object. This is not unusual because both relations can have a different set of meta data, e.g. it is possible that two different personalized linkbases created these relations.

Please note that a Dinopolis object not necessarily represents an entire document. It represents whatever the embedded system provides. A Dinopolis object therefore can also represent a phrase, a word, or a certain character of a document. Hence the language describing the link must be capable of addressing these items (compare to XPath and XPointer in section 3.3).

A relation object itself is a Dinopolis object. According to R 7 , a relation can also be the target and the source of another relation and the addressing model of the Dinopolis system requires every addressable object to be a Dinopolis object.

5.3.2 Representation by the Use of XLink

XLink was chosen as the language to represent relations for a variety of reasons:

1. it is close to become a standard (R 8).
2. it supports bidirectional links (R 2).
3. it provides out-of-line links which allow the creation of external linkbases.
4. arbitrary meta-data can be assigned to an XLink (R 3).
5. Xlink supports one-to-many, many-to-one, one-to-one, and many-to-many relationships (R 5).
6. XPointer, which is part of XLink, allows fine grained addressing into XML documents and supports string ranges (see section 3.3.3 for a discussion of the importance of string ranges concerning link consistency).
7. XLink is part of the XML specifications and the content of Dinopolis objects is described by the use of XML.

XLink was described and discussed in detail in chapter 3. The following example shows an extended link which is generated out of a link base (the DTD defining this

XLink is not shown here, see section 3.4.1 for an example of a DTD). According to the XLink specification, linkbases must be XML conformant. Usually, they will be stored in a database and the result of the database query will be transformed into XML by the link service. For this example, let us assume that the Dinopolis object 1 from figure 5.7 represents a file called file1.xml, the Dinopolis object 2 represents file2.xml, and the Dinopolis object 3 represents file3.xml. The extended link describing the relation object 2 according to the dashed lines in figure 5.7 will look like the following XML fragment:

```
<file_link xlink:type="extended" xlink:title="File Link">
  <source
     xlink:type="locator"
     xlink:label="file1"
     xlink:href="http://root/file1.xml"
  />

  <target xlink:type="locator"
    xlink:title="file2.xml"
    xlink:href="http://root/file2.xml"
    xlink:label="file2"
  />

  <target  xlink:type="locator"
     xlink:title="file3.xml"
     xlink:href="http://root/file3.xml"
     xlink:label="file3"
  />

  <traversal xlink:type="arc"
       xlink:from="file1"
       xlink:to="file2"
       xlink:show="replace"
       xlink:actuate="onRequest"
  />

  <traversal xlink:type="arc"
       xlink:from="file1"
       xlink:to="file3"
       xlink:show="replace"
       xlink:actuate="onRequest"
  />

</file_link>
```

The attributes `xlink:type`, `xlink:title`, `xlink:from`, `xlink:to`, `xlink:actuate`, and `xlink:show` constitute the meta-data of the relation. They may be part of the relation object itself or distributed among the relation object and the pin objects according to the user's preferences. As the pin object is directly attached to a Dinopolis object, some attributes (e.g `xlink:title`) might be well placed at the pin object because the relation object needs not be queried if the user does not follow the link. Nevertheless, the document viewer can access attributes that allow to render the relation accordingly. This can avoid unnecessary network connections (the pin objects and relation object might belong to different servers, see section 5.3.4).

Although the XLink specification proposes some attributes, the set of attributes can be extended. The drawback is that not all clients will understand an extended set of meta-data. However, some embedded system and some special purpose clients will create their own set of attributes.

Retrieving Relations from XML documents

The way how applications can access the external linkbase is described in section 4.2. Another question is how relations can be retrieved from XML documents by the DLS and then be stored in an external linkbase. As XML is an open standard, various free tools exists to extract information from XML document. An open source project called XLinkFilter[2] is currently under development and used by the Dinopolis project. The XLink filter operates together with a SAX[3] compatible parser and allows to retrieve links from an XML document. Figure 5.8 illustrates the communication between the SAX parser, the XFilter, an application, and the link service .

The XLinkFilter processes all SAX events and passes all XLink specific information to the link service. All SAX events are forwarded to the application which can do further processing of the XML document. The location filter that also listens to all SAX events provides information about the target and source anchors of inline links. It returns XPointers that describe the position of the source and target

[2] `http://www.simonstl.com/projects/xlinkfilter/`
[3] `http://www.megginson.com/SAX/`

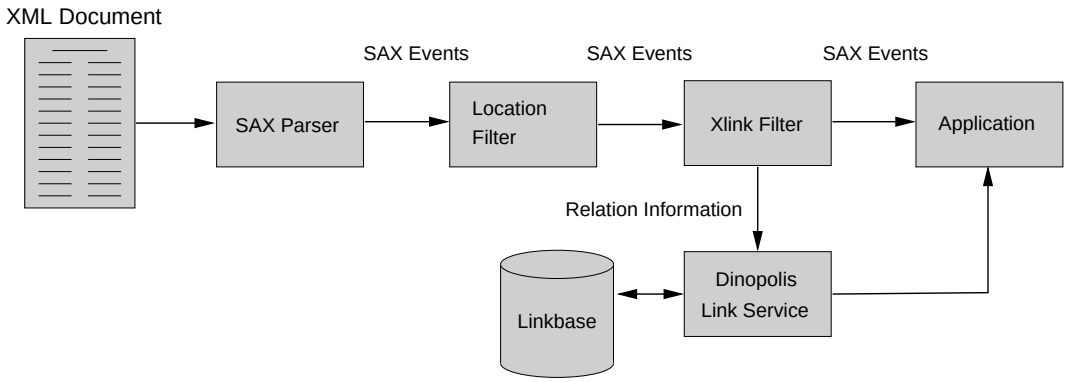

Figure 5.8: How to retrieve relations out of a XML document

anchors. The location filter is still under development and supports only primitive XPointer functionality. The advantage of the design outlined in figure 5.8 is that minor changes of the XLink specification do not affect the application or link service. Only the code of the XLink filter and the location filter has to be rewritten.

5.3.3 Consistency of Relations

One of the most important requirements on the Dinopolis system is to guarantee relation consistency. Under normal operation (the connection to the link service does not fail) a relation under the control of the DLS should never point to the wrong location. The bidirectionality of relations in the Dinopolis system is one prerequisite to allow an efficient link maintenance algorithm. For example if a Dinopolis object referenced by a relation is deleted, the relation must be removed too. This is only possible if the system is able to detect that the deleted object is referenced by a relation. The steps required to update the relations after a move operation are illustrated in figure 5.9 and figure 5.10.

Figure 5.9a shows a Dinopolis object located on server B that defines two relationships to another Dinopolis object (not shown in the figure) located on server A. Figure 5.9b shows the first step where the object is moved to the server C: A new Dinopolis object residing on server B is created by the link service that acts as a forwarder. These forwarder objects are stored by the link service. The address of the

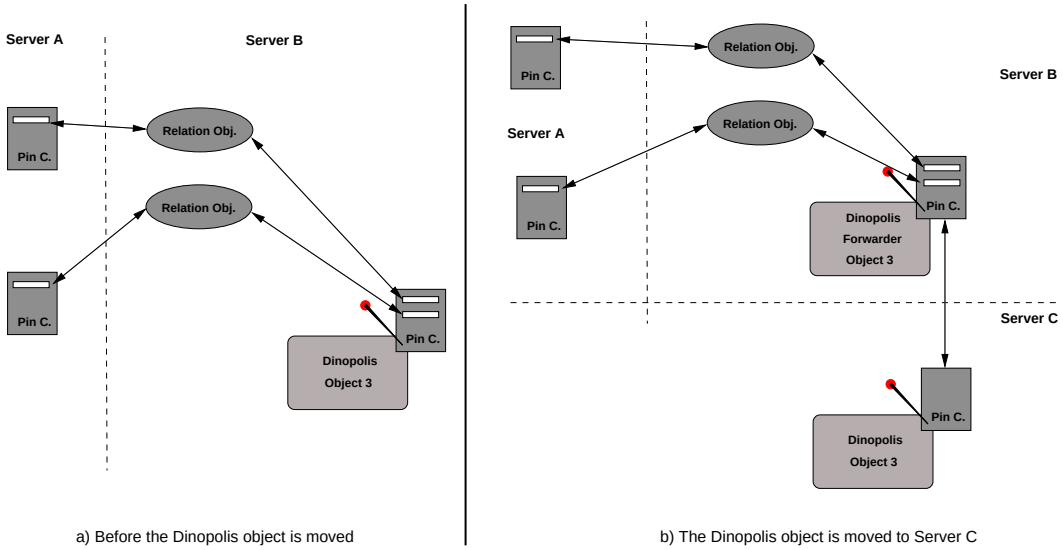

Figure 5.9: The move operation of a Dinopolis object with relations

forwarder object is the same as the address of the old Dinopolis object previously moved to server C. All new relations that are created after the move operation do not affect the forwarder but are attached directly to the Dinopolis object on server C. The advantage of this technique is that the links must not be updated immediately. While updating the two relations shown in 5.9 would not be time consuming, updating a million relations would dramatically increase the duration of the move operation.

The next step do be done by the link service is to gradually update all relations as shown in figure 5.10a. One relation has already been dragged to the Dinopolis object on server C, the other relation still has to be forwarded. Figure 5.10b shows the end of the process, both relations are attached to the object on server C.

If many consecutive move operations occur and a large number of relations has to be updated, the chain of forwarder objects can be considerable long, spread over different link services. Under these circumstances, following a relation might require some network connections to collect all links belonging to appropriate Dinopolis object. Figure 5.11 shows a chain of forwarders, each of them has relations and belongs to a different network.

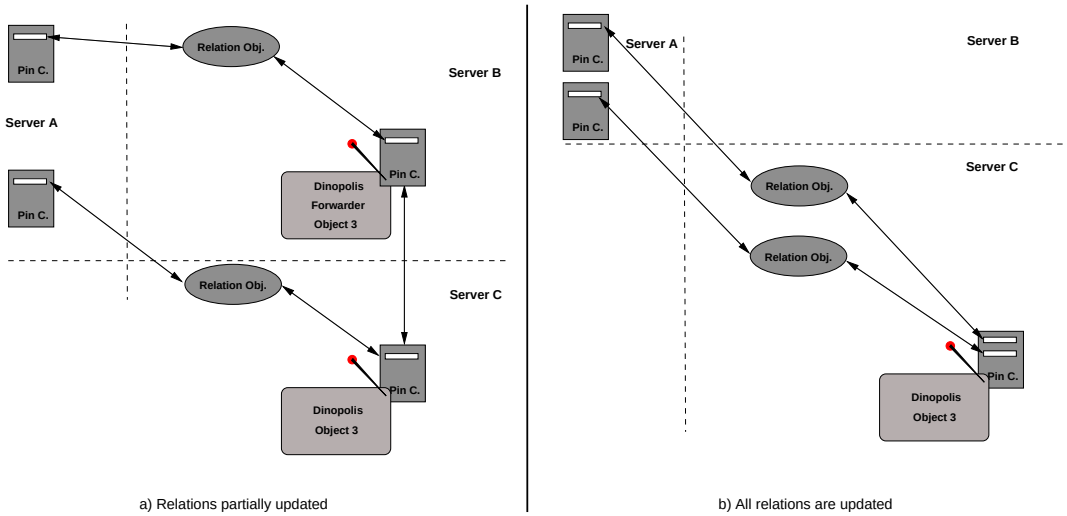

Figure 5.10: Gradually updating the relations

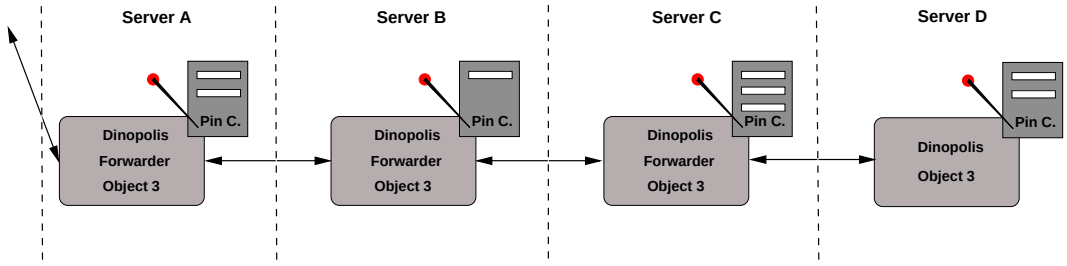

Figure 5.11: A chain of forwarders, each of them with relations

If one of the affected Dinopolis systems is out of operation or not reachable due to network problems, the relation can not be followed at all. Thus, a long chain of forwarders has to be avoided whenever possible. The link service's garbage collection mechanism is responsible to keep the chain short. A special process goes through all objects which act as forwarders and drags the pins to the most recent location of the Dinopolis object. To find this location, the garbage collector must also follow the chain of forwarders.

5.3.4 Other Implementation Issues

The Dinopolis distributed link service takes advantage of the architecture of the Dinopolis middleware:

- the abstraction of the access to embedded system decouples the link service from any special data management system. Whether the linkbase is stored in a filesystem or in a database is of no interest to the link service. As a consequence, the storage system can be exchanged without affecting the link service.
- two or more Dinopolis system can be connected ("merged" in the Dinopolis terminology) to form one virtual Dinopolis system. All merged Dinopolis systems share one global address tree, which is distributed over the participating systems. Access to the address tree and thus to the Dinopolis objects is network transparent.
- To keep track of the changes of Dinopolis objects, the Dinopolis system has a built-in notification mechanism (see below).

Event Driven Link Maintenance

The Dinopolis DLS uses an event driven mechanism to detect whether a Dinopolis object is moved, copied or deleted. It follows the observer design pattern [GHJV98]: If one object changes it's state, the observer (in this case the DLS) is notified and can react accordingly. Each operation carried out on a Dinopolis object may be the target of an observer.

The DLS only maintains explicit relations, the maintenance of implicit relations is up to the appropriate embedded system (see section 5.2). In case of a move operation, the forwarding mechanism (see section 5.3.3) is activated. A delete operation requires the affected relations to be deleted. Regarding the copy operation, the DLS cannot figure out how to handle all the involved relations (see R 1):

- all relations which only affect the copied object and its contained objects have to be created. This requires no update of any existing relations.

- all explicit relations that also affect other Dinopolis objects beside the copied objects have to be updated according to the users' preferences. Depending on the users' choice, some explicit relations will not be created, others will be created. As relations are bidirectional (with some exceptions, see R 1), this will result in new pin objects in the pin containers of other Dinopolis objects that take part in a relationship of the copied object.

Linkbases delivered on read-only media like a CD-ROMs impose some unsolved problems on the DLS. Although external linkbases allow to define relations on read-only documents, the maintenance of bidirectional relations is difficult. The reason is that the CD-ROM can be moved to another location and the system is not able to figure out the new location. Furthermore, new bidirectional relations will change the linkbase of the CD-ROM. Changing the linkbase on the CD-ROM itself clearly is not possible, but it can be copied to the database of the responsible DLS. However, it would be necessary to move the changed linkbase together with the CD-ROM to the new location. This contradicts the way users customary handle media like CD-ROMs: They simply eject and move the CD-ROM to another location by hand. While it is possible for the Dinopolis system to record the ejection of the CD-ROM, it is in most cases not able to detect its new location.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

Many ideas concerning hypermedia and hyperlinks goes back to the early sixties. Among those ideas were link consistency, personalized, dynamic, and typed links. The concept of hyperlinks was also one of the driving forces of the World Wide Web which became popular around the nineties. However, hyperlinks in the World Wide Web were limited in their capabilities, link consistency and personalized hyperlinks were not part of the initial concept. Only a small set of link types was defined but most of them not supported by the predominant browsers. The commercialization of the WWW led to the demand of a more sophisticated link management as proposed by Ted Nelson about forty years ago.

Four major components are necessary to realize a link management that meets the commercial requirements:

1. a link service which provides link consistency, personalized dynamic, and typed links.
2. a flexible and standardized markup language which enables link services to describe hyperlinks.
3. browsers which adhere to the standardized markup language.

4. an standardized protocol that allows link services to exchange link information.

The World Wide Web Consortium reacted on the limitations of HTML and defined a new linking language called XLink which currently undergoes the standardization process. XLink, based upon two further specifications, XPath and XPointer, supports:

- bidirectional links.
- out-of-line links (external linkbases).
- arbitrary meta-data attached to links.
- definition of linking relationships among two ore more resources.
- fine grained addressing, including ranges and string ranges.

Besides some weak points of the specification discussed in this thesis, these features allow a link service to provide an advanced link management to hypermedia systems. The DLS of the Dinopolis middleware system uses XLink to describe its relations. The DLS supports different views (equivalent to personalized links) and implements an efficient link update mechanism. The architecture of the Dinopolis system also requires support for attaching arbitrary meta-data to relations as it is designed to embed different external systems. The DLS enhances the link management of embedded system in that applications can define relationships on objects even if the concerned embedded system can not handle this kind of relationship. Furthermore, it provides consistency of relations to embedded system which have no built in link management.

The Dinopolis system is designed to be open to other applications and hypermedia system, it is designed to be used in an heterogenous environment like the WWW. As an open system, it suffers from the lack of a standardized communication protocol between link services of different vendors. As a consequence, consistency can only be guaranteed to relations which are under the control of the Dinopolis system. It is not likely that a communication protocol will be standardized in the near future.

Another point to consider is how far the predominant browser vendors will support XLink. XLink provides more possibilities and hence is more complicated to understand and to implement. Currently, there are only implementations supporting a small fraction of the power of XLink and XPointer. Due to its architecture, Dinopolis may circumvent this problem: It may use XLink as the internal representation only and present its relations in whatever markup language supported by the browsers through external access gateways. If XLink will be accepted, this is also a way to walk around some weak points of the XLink specification.

6.2 Further Work

Currently, the next version of the Dinopolis middleware system is developed that will support stable handles to Dinopolis objects. Stable handles will ease the maintenance of relations by the link service: In case of a move operation, no forward mechanism has to be implemented by the link service since once a handle is acquired, the Dinopolis object can be accessed through this handle independent of its location. A simpler forward mechanism as described in this thesis will then be implemented by a lookup service.

The Dinopolis project uses as much open source tools as possible. Currently, only the basic XPointer functionality is supported by the available open source tools as XPointer is still a candidate recommendation. External linkbases are weak defined in the specification (which is also a candidate recommendation). As soon as both specifications reach the state of a recommendation, the full XPointer and XLink functionality will be supported by the Dinopolis system.

Appendix A

Recommended Link Types in HTML

User agents, search engines, etc. may interpret these link types in a variety of ways. For example, user agents may provide access to linked documents through a navigation bar. Unfortunatly, as has already be mentioned, more or less none of the features described here are supported by current browsers.

- **Alternate:** Designates substitute versions for the document in which the link occurs. When used together with the lang attribute, it implies a translated version of the document. When used together with the media attribute, it implies a version designed for a different medium (or media).
- **Stylesheet:** Refers to an external style sheet. See the section on external style sheets for details. This is used together with the link type "Alternate"for user-selectable alternate style sheets.
- **Start:** Refers to the first document in a collection of documents. This link type tells search engines which document is considered by the author to be the starting point of the collection.
- **Next:** Refers to the next document in a linear sequence of documents. User agents may choose to preload the "next"document, to reduce the perceived load time.

- **Prev:** Refers to the previous document in an ordered series of documents. Some user agents also support the synonym "Previous".
- **Contents:**Refers to a document serving as a table of contents. Some user agents also support the synonym ToC (from "Table of Contents").
- **Index:** Refers to a document providing an index for the current document.
- **Glossary:** Refers to a document providing a glossary of terms that pertain to the current document.
- **Copyright:** Refers to a copyright statement for the current document.
- **Chapter:** Refers to a document serving as a chapter in a collection of documents.
- **Section:**Refers to a document serving as a section in a collection of documents.
- **Subsection:** Refers to a document serving as a subsection in a collection of documents.
- **Appendix:** Refers to a document serving as an appendix in a collection of documents.
- **Help:** Refers to a document offering help (more information, links to other sources information, etc.)
- **Bookmark:** Refers to a bookmark. A bookmark is a link to a key entry point within an extended document. The title attribute may be used, for example, to label the bookmark. Note that several bookmarks may be defined in each document.

Appendix B

Available Axis in XPath and XPointer

The following list shows all axis available in XPath and XPointer. In this description, the term context node is used according to the XPath specification. The XPointer uses the term location context to cover nodes, points, and ranges.

- **ancestor:** selects the parent of the context node, the parent of the parent of the context node and so forth recursivly down the root.
- **ancestor-or-self:** selects the ancestors of the context node and the context node itself.
- **attribute:** selects the attribute(s) of the context node
- **child:** selects the immediate child/children of the context node.
- **descendant:** selects the children of the context node, the children of the children of the context node and so forth.
- **descendant-or-self:**selects the context node itself and its descendants.
- **following:** selects all nodes that start at the end of the context node.
- **following-sibling:** selects all nodes that start ag the end of the context node and have the same parent.

- **parent:** selects the parent of the context node which is unique.
- **preceding:** selects all nodes that start before the beginning of the context node.
- **preceding-sibling:** selects all nodes that start before the context node and have the same parents.
- **self:** selects the context node itself.

Bibliography

[And96] Keith Andrews. Applying hypermedia research to the world wide web. *Position Paper for the workshop "Hypermedia Research and the World-Wide Web*, March 1996. available online http://hyperg.iicm.tu-graz.ac.at/apphrweb.

[ASR94] Bernd Amann, Michel Scholl, and Antoine Rizk. Querying typed hypertexts in multicard/o2. *Proceedings of the 1994 ACM European conference on Hypermedia technology*, September 1994.

[AV94] Helen Ashman and Janet Verbyla. Dynamic link management via the functional model of the link. February 1994. Proccedings of the Basque International Workshop on Information Technology, France.

[Ber99] Cliff Berg. The state of Java application middleware, Part 1. *Javaworld*, March 1999. available online http://www.javaworld.com/javaworld/jw-03-1999/jw-03-middleware.html.

[BHW99] Paul De Bra, Geer-Jan Houben, and Hongjing Wu. Aham: A dexter-based reference model for adaptive hypermedia. *Proceedings of the tenth ACM Conference on Hypertext and hypermedia*, February 1999.

[BLC] T. Berners-Lee and D. Connolly. Hypertext markup language - 2.0. *RFC 1866*. available online http://www.cis.ohio-state.edu/htbin/rfc/rfc1866.html.

[BLC90] Tim Berners-Lee and Robert Cailliau. Worldwideweb: Proposal for a hypertext project. Technical report, CERN, 1990.

[BLFIM98] T. Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform Resource Identifiers (URI). RFC 2396, August 1998. available online http://www.faqs.org/rfcs/rfc2396.html.

[Bos97] Jon Bosak. Xml, java and the future of the web. Technical report, Sun Microsystems, 1997. available online: http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm.

[Bus45] Vannevar Bush. What we may think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[CH] Leslie Carr and Wendy Hall. The implications of xml for open hypermedia. Technical report. Proceeding to the4 th Workshop on Open Hypermedia Systems at Hypertext, Pittsburgh.

[CHH98] L. A. Carr, W. Hall, and S. Hitchcock. Link services or link agents ? Proceedings of the ninth ACM conference on Hypertext and hypermedia, 1998.

[CHKZ00] Dallermassl C., Haub H., Maurer H.and Schmaranz K., and Zambelli. Dinopolis - a leading edge application framework for the internet and intranets. Conference proceeding WebNet 2000, October 2000.

[Con99] World Wide Web Consortium. Xml path language (xpath) version 1.0, November 1999. available online at http://www.w3.org/TR/xpath.

[con00a] World Wide Web consortium. Document object model (dom) level 2 traversal and range specification version 1.0, September 2000. available online at http://www.w3.org/TR/2000/PR-DOM-Level-2-Traversal-Range-20000927/.

[Con00b] World Wide Web Consortium. Xml pointer language (xpointer) version 1.0, June 2000. available online at http://www.w3.org/TR/xptr.

[CR98] Ken A L Coar and D.R.T. Robinson. Common gateway interface version 1.1. *Internet Draft*, 1998. available online at http://Web.Golux.Com/coar/cgi/draft-coar-cgi-v11-00.html.

[CRH] Leslie Carr, David De Roure, and Wendy Hall. The distributed link service: A tool for publishers, authors and readers. Technical report, Department of Electronics and Computer Science, University of Southampton.

[Dal99] Christof Dallermassl. Aspects of integration of heterogenous server systems in intranets - the java approach. Master's thesis, IICM, Graz University of Technology, 1999.

[DHK+00a] Christof Dallermassl, Heimo Haub, Harald Krottmaier, Klaus Schmaranz, and Philipp Zambelli. Adaptive learning environment framework. submitted to, June 2000.

[DHK+00b] Christof Dallermassl, Heimo Haub, Harald Krottmaier, Klaus Schmaranz, and Philipp Zambelli. Using highly sophisticated middleware for building arbitrarily distributed teaching environments. June 2000.

[ea97] Michael Biber et al. Fourth generation hypermedia: Some missing links for the world wide web. *International Journal on Human Computer Studies, Academic Press*, 47:31–65, 1997.

[GHJV98] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. 1998. ISBN 0-201-63361-2.

[Goo98] Danny Goodman. *Dynamic HTML - The Definitive Reference*. O'Reilly, 1998.

[Gro98] W3C XML Working Group. Extensible markup language (xml) 1.0. Technical report, W3C, February 1998. available online: http://www.w3.org/TR/REC-xml.

[Gro99a] W3C HTML Working Group. Html 4.01 specification. Technical report, W3C, December 1999. available online: http://www.w3.org/TR/html4/.

[Gro99b] X3C XML Linking Working Group. Xml xlink requirements version 1.0. Technical report, W3C, February 1999. available online: http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm.

[Gro00a] The XML Working Group. Xml schema part 0: Primer. W3C Working Draft, September 2000. available online at http://www.w3.org/TR/xmlschema-0/.

[Gro00b] X3C XML Linking Working Group. Xml linking language version 1.0. Technical report, W3C, July 2000. available online: ttp://www.w3.org/TR/NOTE-xlink-req/.

[Hau99] Heimo Haub. Aspects of access management in heterogeneous distributed object systems. Master's thesis, IICM, Graz University of Technology, 1999.

[IIC99] Dino Team IICM. Dino V4 Software Requirements. Technical report, IICM, Graz University of Technology, 1999.

[J.D99] Steven J.DeRose. Xml linking. *ACM Computing Surveys*, 31, December 1999.

[Kap95] Frank Kappe. Maintaining link consistency in distributed hyperwebs. *Proceedings of INET'95, Honolulu, Hawaii*, June 1995. available online: http://www.isoc.org/HMP/PAPER/073/abst.html.

[Mau96] Hermann Maurer. *HyperWave - The Next Generation Web Solution*. Addison Wesley, 1996.

[MQ96] Murray Maloney and Liam Quin. Hypertext links in html. 1996. Internet Draft, currently expired, available online http://ww.ics.uci.edu/~ejw/authoring/draft-ietf-html-relrev-00.txt.

[Mue99] Russell Mueller. Introduction to asp. 1999. available online at http://www.devshed.com/Server_Side/ASP/Introduction/.

[Nel92] Theodor Holm Nelson. *Literary Machines 93.1*. Mindful Press, 1992.

[Nel95] Theodor Holm Nelson. The heart of connection: hypermedia unified by transclusion. *ACM*, 38(8):31–33, August 1995.

[Nie95] Jakob Nielsen. *Multimedia and Hypertext: The Internet and Beyond*. AP Professional, Boston, MA, 1995.

[Pea89] Amy Pearl. Sun's link service: A protocol for open linking. November 1989. available online at http://www.w3.org/Conferences/WWW4/Papers/178/.

[Qui98] Liam R. E. Quin. Links in xml: Detection, representation and presentation. Technical report, November 1998. available online at http://www.groveware.com/~lee/papers/markup98conference/links.html.

[Rag98] Dave Raggett. Raggett on html 4. Technical report, 1998.

[RCH94] Liam Relihan, Tony Cahill, and Michael G. Hinchey. Untangling the world wide web. *ACM Conference proceedings on Technical communications at the great divide*, October 1994.

[RS97] Antoine Rizk and Dale Sutcliffe. Distributed link service in the aquarelle project. pages 208–209, 1997.

[Sad97] Darleen Sadoski. Client/Server Software Architectures–an Overview. Technical report, Carnegie Mellon, Software Engineering Institute, 1997. available online http://www.sei.cmu.edu/str/descriptions/clientserver_body.html.

[Sag98] Ajit Sagar. Servlets & friends. *Java Developer's Journal*, 1998.

[SM94] K. Sollins and L. Masinter. Functional requirements for uniform resource names. RFC 1737, December 1994. available online http://www.faqs.org/rfcs/rfc1737.html.

[SW88] John Smith and Stephen Weiss. An overview of hypertext. Technical report, CACM, July 1988.

[TD96] Klaus Tochtermann and Gisbert Dittrich. The dortmund family of hypermedia models - concepts and their application. *J.UCS*, 2:34–56, 1996.

[TLMM94] T.Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators. RFC 1738, December 1994. available online at http://www.faqs.org/rfcs/rfc1738.html.

[WIM98] E. Whitehead, UC Irvine, and M. Murata. Xml media types, July 1998. available online at http:///www.faqs.org/rfcs/rfc2376.html.

[WR98] Weigang Wang and Roy Rada. Structured hypertext with domain semantics. *ACM Transactions on Information Systems*, 16:372–412, 1998.

[Zam00] Philipp Zambelli. Shared bookmarks - building an application on a distributed object system. Master's thesis, January 2000. available online http://www.dinopolis.org/documentation/misc/theses/.

Index

V

X