Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Enriching the XLink Standard with RSL Metamodel Features

Master thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

David Sverdlov

Promoter: Prof. Dr. Beat Signer
Advisor: Ahmed O.A. Tayeh

Academic year 2015-2016

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN

Enriching the XLink Standard with RSL Metamodel Features

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

David Sverdlov

Promotor:	Prof. Dr. Beat Signer
Begeleider:	Ahmed O.A. Tayeh

Academiejaar 2015-2016

Abstract

Digital documents do not exist in isolation but rather share connections with other documents. Based on this fact and the hyperlink concept, various existing digital document formats support limited features of linking (i.e. unidirectional and embedded hyperlinks) to other documents. Moreover, various hypermedia models and applications have been proposed to enable the linking between documents and to overcome the limitations of the linking features offered by existing document formats. The Resource Selector Link (RSL) metamodel and the XML Linking Language (XLink) standard are well known, recent link models that support various linking features. The XLink standard that is recommended by the World Wide Web Consortium (W3C) has been designed to support only the advanced linking features in XML documents such as bi- and multi-directional hyperlinks. In contrast to XLink, the RSL metamodel is flexible and extensible, which allows for support of linking to most existing and emerging document formats. Moreover, RSL supports other important hypermedia features such as the management of user access rights, supporting overlapping hyperlinks and context resolvers.

In this thesis, we investigate the possibilities to enrich the XLink standard with important features of the RSL metamodel. Our investigation leads to an enhanced XLink standard that does not invalidate its original W3C specification and allows the linking across existing as well as emerging document formats. Our enhanced XLink standard further supports advanced hyperlink features such as user access rights and the resolution of overlapping hyperlinks. As a proof of concept, we use the enhanced XLink standard in an existing cross-document link service that supports the linking across existing as well as emerging document formats.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

My greatest gratitude goes towards my promoter Prof. Dr. Beat Signer and my supervisor Ahmed O. A. Tayeh for their patience and guidance on this long road. There have been several obstructions along the way, but Ahmed always believed in me and kept pushing me to work hard and keep seeing the end goal. Even while ill or on vacation, he has been with me until the end. I wish him all the best with his family, his research and any challenges he might be faced with.

With the endless amount of support she has given me, I can safely say that my mother, Alinoë Van Looveren, has been an essential contributor to the completion of this thesis. She is hands down the most supportive, intelligent, forgiving and loving person I have ever had the fortune of knowing. I am extremely grateful to her, my stepfather and my siblings for all their encouragements and musical talents that often accompanied me while working on this thesis.

My dear friends, Quentin, Mathijs, Kenny, Elke, Jo, Nick, Toon and Inge also receive an acknowledgement for all the support, suggestions and most importantly, the occasional distractions they have provided me with. I am very grateful to have met these amazing people and to be able to call them my friends.

Eric Gijbels is also an important person who must be mentioned in this acknowledgement. For it was with his input, his level-headed pushing, that this thesis took a turn in the right direction when it really needed it. The same goes for the conversations I have had with Jacqueline. It has been really helpfull being able to talk about anything and always receive solid advice.

For their time and effort put into proofreading, spell-checking and answering annoying questions, I extend my gratitude to Ahmed, Rani, Mathijs, Elke, Quentin and Audrey.

To any other family members, friends or acquaintances that have shown interest or taken any part in this thesis and have not yet been mentioned, thank you. This has been quite an interesting and educational experience, especially since it happened during a time when there was a lot going on in my life. The completion of this thesis finalises my student career, which I will remember fondly, and opens the doors for new adventures.

Contents

1 Introduction

1.1 Context

Digital documents seldomly exist on their own, separated from each other. Most digital documents share direct or indirect associations with other documents. An association in this context can be defined as a relationship between different elements that share some sort of connection. An example of a direct association could thus be an actual hyperlink or a reference to another document, while an indirect association could just refer to the similarity of content. The existence of these associations, or *links* as they are called, is essential for maintaining structure and creating connections between all digital documents. This does not only apply to local resources stored on a personal computer but also to the immense amount of documents on the Web. Without these links on the Internet, every web page would exist in isolation and users would be unable to navigate from one page to another without manually entering the address of the next page.

Not all of these direct and indirect associations, however, can be fully realised through current linking possibilities. While most linking solutions used today provide the basic means to create and visualise hyperlinks, many limitations still exist. Current methods of linking, for example, allow the owner of a website to create standard hyperlinks on their web pages via the HyperText Markup Language (HTML). For creating links in the Microsoft Word desktop application, users can click on the 'hyperlink' button to open an interface that assists them in creating a link to a different section, web page or another document entirely. Links that are stored in the source document, such as the HTML and Word hyperlinks we just mentioned, are called *embedded* links. A link that only allows navigation from one point to another, but not the other way around, is said to be *unidirectional*. Considering the World Wide Web (WWW) can be seen as an immense connected network of documents and web pages, all connected via hyperlinks, it is interesting to see the widespread use unidirectional and embedded links have seen. Since the first version of the Hypertext Markup Language, it has been possible to create these links via the anchor tag ('<*a*>'). In Figure 1.1, an example is presented of a hyperlink on a web page that sends the user to the VUB home page when clicked.

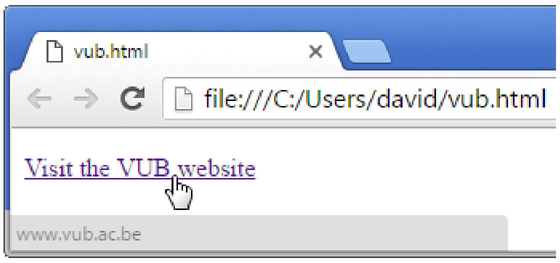


Figure 1.1: An HTML hyperlink on a web page referring to a different website

HTML hyperlinks have the ability to link to entire web pages and third-party resources. Via anchors ('#'), they can even address parts of web pages, but these anchors are limited to HTML documents only. They cannot be used to link to specific parts of other document types. For instance, it is currently not possible on the Web to create a hyperlink leading to the third page of a PDF document.

Document types with some forms of integrated linking, such as Microsoft Word or PDF documents, provide different ways of defining a hyperlink to a website or to a part of the document itself. Unfortunately, these forms of integrated linking still lack a feature allowing the user to link to parts of other document types. Standard hyperlinks are limited to connecting two entities and they can also only be traversed in one direction: from the source of the link to the target destination.

While this is a fairly common limitation of document formats and programs today, more advanced document linking and annotating solutions are being developed. There have been several attempts to create applications that offer linking and annotation properties, yet none of them had significant success.

In an effort to define a standard for linking in XML documents, the World Wide

Web Consortium[1] (W3C) created the XML Linking standard (XLink) [8]. XLink overcomes the shortcomings of existing linking solutions by offering advanced hyperlinks such as bi- and multidirectional hyperlinks. With XLink hyperlinks, multiple snippets of XML documents can be linked with each other. This can be seen in Figure 1.2 and Figure 1.3.

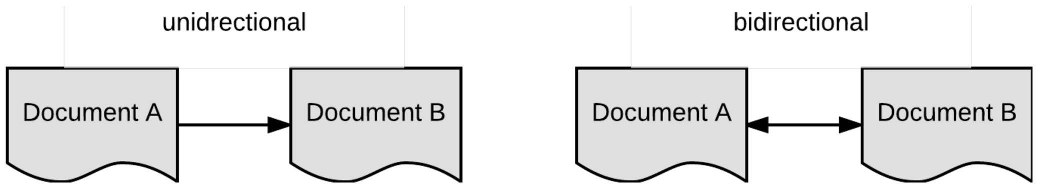


Figure 1.2: Uni- and bidirectional links

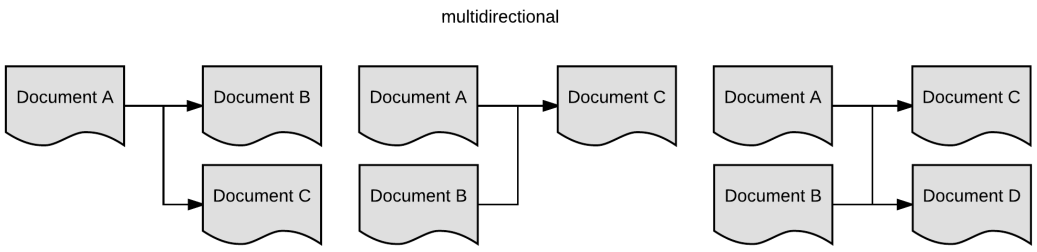


Figure 1.3: Examples of a multi-target link, a multi-source link and a multi-source, multi-target link

Since the XLink standard is limited to solely linking in XML documents, Signer and Tayeh have presented an application to create and browse links between different types of documents [28]. At the core of this application lies the Resource Selector Link (RSL) metamodel [27], a flexible linking model which supports several advanced linking features, such as bi- and multidirectional links and user rights management. The RSL metamodel specifies three main concepts (referred to as *entities*): resources, selectors and links. While a resource is a reference to a (local or remote) document, a selector can be specified to address a part of a resource. The link concept in this metamodel connects two entities with each other. An RSL link can then, for example, connect two selectors with each other or a selector and a resource or a link and a resource, etc. When linking different parts of different documents, we speak of *cross-document linking*. The three components of this extensible model can also be associated with properties, which allows applications to add custom features to the model.

[1]https://www.w3.org/

1.2 Problem Statement

The XLink standard possesses several interesting and powerful aspects, such as the ability to create complex link structures and to separate links from content via link databases (or *linkbases*). However, an unavoidable disadvantage of the XLink standard is that cross-document linking is restricted to XML-based documents only. The standard makes use of XPointer expressions to navigate through XML-based document trees and select certain parts. Thus, parts of document types that are not tree-structured cannot be addressed by the XLink standard. This limiting factor could be one of the reasons XLink has not attained wide-spread use and visibility by various linking solutions. Since the XLink recommendation has been designed with a focus on the linking essentials, it lacks some features that other hypermedia models do have. In the RSL metamodel, for example, there is support for access rights management. This makes it possible to define content ownership and to specify the (in)accessibility of different items to different users. Another feature provided by the RSL model is *context resolvers*. An application can evaluate context resolvers, which are associated with RSL entities (links, resources and selectors), and depending on the context, the entity will be shown or hidden. For example, a link could contain a context resolver that only makes it visible/accessible on week days or during the day time. Overlapping selectors are best explained by imagining a link that selects a full paragraph and another link that selects just one word inside that previous paragraph. This example is illustrated in Figure 1.4. In HTML web pages and many other document types, overlapping selectors are not allowed because the applications do not know which selector the user desired to use (when clicked on the overlapping selection). The RSL metamodel provides a solution to this problem by placing each selection on a different layer. With these features, in combination with its properties, the RSL metamodel can be considered to be a sufficiently flexible and extensible linking model. However, the RSL metamodel is not a recommended standard, whereas XLink is. Standards are important and beneficial in many ways. They have been incrementally designed to ensure that the best practices for Web development are implemented. Interoperability between different devices, backwards compatibility with previous versions also classify as perks of the usage of standards. Since a linking standard already exists, it is worthwhile investigating how XLink could be enhanced to possess more features than it initially does and how to support several key features that are beneficial for cross-document linking. The functionalities that we would like to enhance XLink with are the ones observed (and previously mentioned) in the RSL metamodel, namely resolving overlapping selectors, providing user rights and access rights management, being able to create links from and to links, and supporting context resolvers.

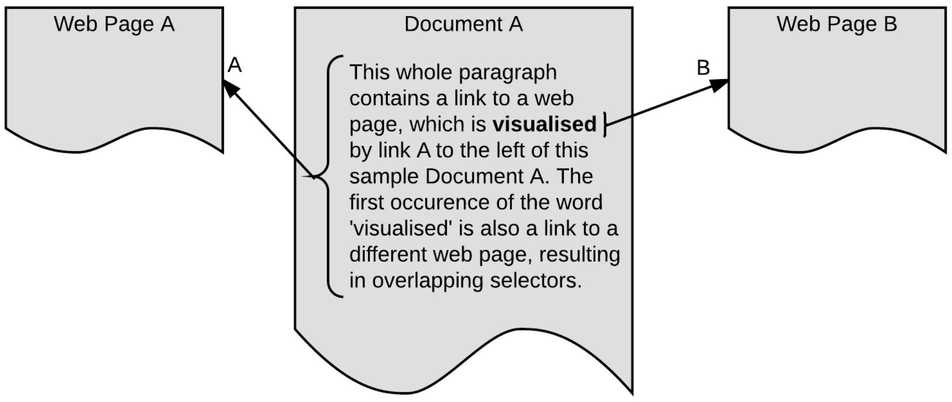

Web Page A
Document A
Web Page B
A
B
This whole paragraph contains a link to a web page, which is **visualised** by link A to the left of this sample Document A. The first occurence of the word 'visualised' is also a link to a different web page, resulting in overlapping selectors.

Figure 1.4: Document A containing two link sources (a word and a paragraph) that overlap each other

1.3 Objectives

In this thesis we investigate how the XLink standard can be enhanced for cross-document linking for any document type, including those that are not XML-based. In order to do this, we provide a comparison between the XLink standard and the RSL metamodel which shows the strengths and weaknesses of each of these linking models. We provide a functional one-to-one mapping between the XLink and RSL linking models and evaluate a proof of concept implementation in the cross-document linking and browsing solution, presented in [27]. Support for various popular document types will be implemented to show the effectiveness of the extensible plug-in architecture.

1.4 Thesis Structure

In Chapter 2, a detailed background of linking in documents and the two linking models is presented, followed by an extensive comparison of the XLink standard and the RSL metamodel. The comparison depicts a clear idea of which features we want to enrich the XLink standard with. Chapter 3 evaluates several strategies of enhancing the XLink standard and Chapter 4 revolves around the implementation of this enhanced XLink standard in a link service. Chapter 5 presents an evaluation of the enhanced XLink method, conclusions and future work.

2 Background

In this chapter, an overview of the history of linking on the Web and in document formats is discussed. The advantages and limitations of various linking strategies are evaluated, focussing on two linking models, namely the RSL metamodel and the XLink standard. An in-depth comparison of these two models is also presented.

2.1 History of Linking Documents

Around the Second World War, an enormous amount of scientific research was being conducted and information started sprouting all over. For researchers, this resulted in increasing difficulty to effectively store, locate, organize and share their work, as well as finding articles of colleagues relevant to their research fields. In 1945, the Atlantic Monthly published a paper written by Vannevar Bush about a vision of a device that would be able to store and organize this enormous flow of information [4]. In this paper, titled *As We May Think*, Bush described a mechanical device designed for digital document management that would connect information in a way very much like that of the human mind: via associations. The device, named "Memex", can be seen in Figure 2.1. The Memex resembles a regular desk, equipped with several distinctive items. Two touch-sensitive screens can be observed in the centre, a glass plate on the left and a panel with buttons on the right. The two screens allow users to view the contents stored on the machine and make annotations to them.

The described links, or *trails* as they were called in the paper, served the purpose of connecting information between documents with each other. Trails could be created by using the screens to view documents and a stylus or keys to mark the information to be linked. Trails could be given certain tags, which in turn could be used to retrieve links by entering them via the key pad. The microfilm, on which the documents were stored, had the added benefit of being removable: as such, these microfilms could be shared amongst colleagues.

The vision of the Memex shows a remarkable resemblance to the desktop computers we currently use in our everyday life, which is astonishing considering the technology to realise it did not even exist back then. The notion of creating trails between documents to link information has led to the rise of several linking applications. Bush's vision is often considered the first step towards hypermedia systems and projects including Project Xanadu [21], Sun's Link Service [24] and Chimera [1].

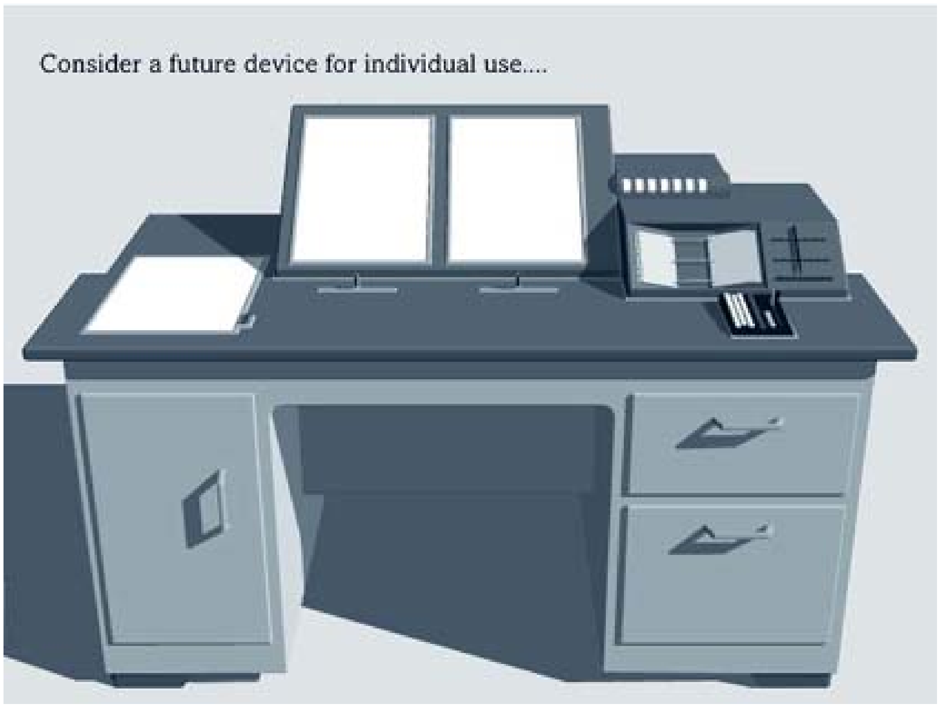


Figure 2.1: The Memex as described by Vannevar Bush in the article "As We May Think"

Project Xanadu, founded in 1965 by Ted Nelson, can be seen as one of the first hypertext projects. Nelson, who evidently also coined the term *hypertext*, wanted to break away from the idea that computer systems should be made to simulate physical paper as best as possible [20]. By attempting to mimic real paper, applications would also inherit the limitations of paper documents. One of the limitations of physical documents is that a piece of paper provides a fixed, two-dimensional area for static content, while a computer system could present data in a completely different way. Books contain a sequential order of pages, but information could also be presented in a non-sequential way or in a way that makes better use of the three-dimensional space. A computer could present information in a 3D environment or on an endless canvas. According to the team behind Project Xanadu, the possibilities for a fully potential hypermedia system should not be limited by trying to mimic the real world. With no other systems supporting linking in documents, Project Xanadu introduced a new linking concept called *transclusion*. Transclusion is the term for the procedure of including parts of documents in other documents.

> “A link connects two things which are different. A transclusion connects two things which are the same.” — Ted Nelson.

This concept is illustrated in Figure 2.2, where document A contains another document (document B). Since document B is transcluded inside the other document, any changes made to document B would be directly visible in document A.

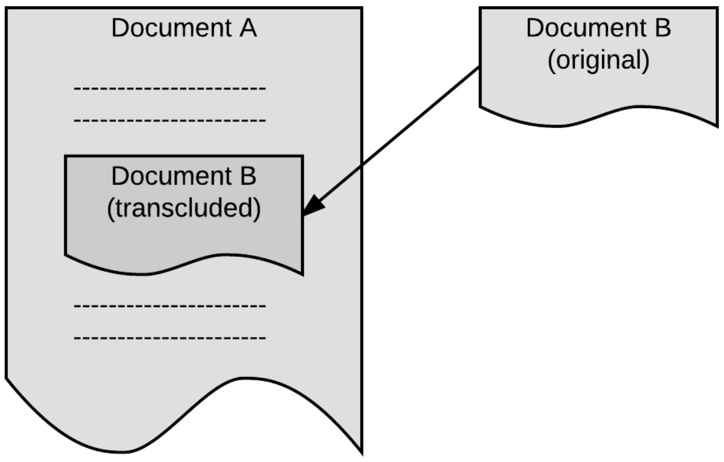


Figure 2.2: Example of a transclusion where document B is directly embedded in another document, document A

Inspired by the associative trails presented in the Memex, Project Xanadu implemented bidirectional links between documents. As we have mentioned earlier, bidirectional links are links that allow users to navigate from one point of a hyperlink to another and back. This is an important feature as in this matter, link integrity can be maintained in a setting where documents are constantly changing. Suppose one has two documents (document A and document B) with a *unidirectional* link from document A to document B and document B is removed, the link would be left dangling. It is only logical then that these links are called *dangling* links. This concept is illustrated in Figure 2.3. If the link would have been *bidirectional*, then document B's embedded link could be followed to find and update/remove document A's link so it is not left dangling.

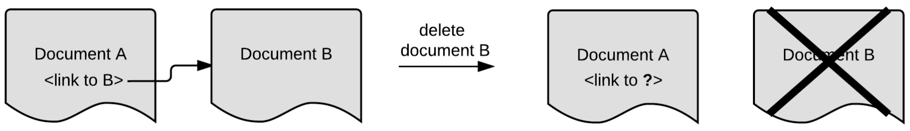


Figure 2.3: Example of the dangling links problem with embedded links

Whereas the idea for Project Xanadu was founded in 1965, an implementation has only been introduced in 1998 under the name Unadax Green[1]. This application, consisting of a front-end and a back-end, could track versions and changes of a document and visualise these in a side-by-side comparison window.

During the seventies, popular document types were not as rich as they are today but rather plain and simple. In other words, they did not have any linking features. Text files just stored the text content without any way of linking related content.The first notion of linking in document formats was introduced by markup languages Generalized Markup Language (GML) and Scribe [25]. A markup language allows documents to include structural and/or representational instructions for document visualising applications. These instructions, specified in tags, allow documents to specify their own formatting, which will be interpreted by appropriate viewing programs. The kind of linking done by Scribe was not yet about creating hyperlinks to other documents but rather about providing structural links that were used to reference local sections and footnotes. These links are called cross-references. With the actual text of a document being separated from its formatting, the Scribe markup language introduced the notion of separation of

[1]http://udanax.xanadu.com/green/index.html

content and visualisation. Structural tags are used to indicate how content should be structured and can be used to include smaller files in bigger documents. With this principle, main documents can divide content into different sections and then include those. This technique is still used a lot in HTML, where other pages, scripts and stylesheets are included in a main document. The Standard Generalised Markup Language (SGML)[2], was the first markup language to interpret a tag (`<link>`) that defines a hyperlink to another resource. An example snippet of an SGML document containing a link can be seen in Listing 2.1.

```
<article>
<header>
<title>Small SGML document
<authors>David Sverdlov
<abstract>Abstract of a small SGML document
<body>
<block>This is a small block of text containing a link element, pointing to the
     <link url="http://vub.ac.be">VUB</link> website.
</article>
```

Listing 2.1: SGML example of a small document

These links, however, are unidirectional and embedded, which means that they originate in the document they are defined in (the source) and point to a location (the target) that is unaware of any incoming links. We have previously discussed the drawbacks of links with these properties. Another issue with these documents is that ownership is usually required to create a link in a document, since the link would become embedded in the document itself. Most pages in the early Web 1.0 were static pages with predefined content that could not be edited. In the Web 2.0, where dynamically generated content became the new way of the Web, many more possibilities have emerged for users to add their own input, which could include links [23]. As such, users are still unable to post a link to their personal website on just any web page, unless there is a dedicated space for users to dynamically change content, like a forum or a comment section.

Several open hypermedia systems like Chimera [1], Sun's Link Service [24] and Intermedia [32] started managing links externally to solve the limitations of having them embedded in the documents themselves. An advantage of managing links separately from content is the complete link overview. This means that an application knows every participating link of a document, inbound or outbound, and retrieve the associated resources. An outbound link is a link in a document that targets another document, while an inbound link is a link coming from another document. If links were only stored embedded in the documents themselves, an application would have to open a file, grab the links, look up the target documents

[2]https://www.w3.org/MarkUp/SGML/

for their links, etc. Furthermore, managing the links externally allows us to create links from documents to which we do not have the access rights. After all, if documents cannot be accessed, embedded links cannot be inserted. Integrity of links can also not be guaranteed when links are embedded in the documents themselves, since when a document would be removed, all links targeting that document would be left *dangling*, as can be seen in Figure 2.3. If document B would be removed, the embedded link in document A would be left dangling.

The first worldwide hypermedia system was founded by Tim Berners-Lee and is now known as the world wide web (WWW) [2]. The WWW introduced web pages, written in the HyperText Markup Language (HTML), which could display elements like text, images and embedded hyperlinks. These hyperlinks can define direct associations to web pages or parts of web pages. These links only provide a simple way of linking documents or media types (e.g. images) and there are several drawbacks to this technique. Unfortunately, the WWW neglected most of the more advanced features for linking documents. Firstly, HTML links are unidirectional. When a user clicks on a unidirectional hyperlink on a web page, they will be directed towards the target but cannot click a link to go back to the source of the link. However, most browsers do keep a list of visited web pages and allow users to navigate back to previous pages. While this list allows users to navigate links backwards, a website still has no way of knowing how many links are targeting it or where they are located. Secondly, hyperlinks are embedded in the web pages. They are bound to it and users do not have the required rights to edit them (for adding their own links, for example). Hyperlinks defined in HTML can only have one target document and can only be edited by the author of the source document. It is not possible, for example, to make a link from a source element to more than one target location. In HTML, an element can only be the starting source of one link. Unidirectional links prevent the creation of more complex linking structures.

In order to improve the linking in XML documents, the W3C published the XML Linking Language (or XLink) [9]. The XLink specification provides a structure to separate the links from content again. Other hypermedia models such as the RSL metamodel found their way to multiple hypermedia applications.

More and more sophisticated document formats started emerging, with more features and possibilities with every new format. Document formats such as OOXML and later PDF became popular working formats for text processing but they also worked with embedded, unidirectional links. Aside from the limitations coming from the links, these document formats also did not support any form of version management or access rights management. The next section will review various linking mechanisms in different document formats and see what is available today.

2.2 Linking in Document Formats

Most of the formats that are popular today still only provide limited ways for creating links between different parts of documents, to parts of other resources or to entire documents. There are plenty of document formats that can support the linking to web pages and these links can be presented in various ways, e.g. as a footnote, as a clickable image in a sideshow, as a hyperlink on a web page, etc. Table 2.1 lists various document types and their linking mechanisms, which will be discussed further on.

Format	Hyperlink Type	Supported Target Resources
HTML	Unidirectional	web resources, entire third-party documents
LaTeX	Unidirectional	web resources, entire third-party documents
PDF	Unidirectional	web resources, entire third-party documents, parts of PDF documents
XML	Uni-, bi- and multidirectional	web resources, entire third-party documents, parts of XML-based documents
DocBook	Unidirectional	web resources, entire third-party documents, parts of other DocBook documents
OOXML	Unidirectional	web resources, parts of other OOXML documents

Table 2.1: Comparing links in different file types, based on [28]

The most used form of linking is seen in pages on the Web. Web pages can link internally by targeting different sections of the same website or link externally to other websites and third-party resources.

```
<html>
  <body>
    <p> Come visit the <a href="http://www.vub.ac.be">VUB</a> </p>
  </body>
</html>
```

Listing 2.2: Simple link in HTML

In Listing 2.2, an example of a basic hyperlink of a web page is shown where the user will be directed towards the VUB website when they click on the word ‘VUB’. The behaviour of the hyperlink can be controlled to a certain degree, such as forcing the website to be loaded in the current window or to open in a new tab (depending on browser settings). The example links in Listing 2.3 show how this can be achieved.

```
<html>
  <body>
    <a href="http://www.vub.ac.be"> Open link here </a>
    <a href="http://www.vub.ac.be" target="_blank"> Open link in new tab/window
     </a>
  </body>
</html>
```

Listing 2.3: A link that opens in the current tab and one that opens a new tab/window

We have already mentioned that these links exhibit rather simple behaviour and they are constrained by several limitations. A possible solution for the limitation brought forth by unidirectional links would be to create extra links, going from the target back to the source of the link. This, however, is not always possible. For example, if you want to create a bidirectional link between your website and another, you need the access rights of the other page in order to create the link back to your website. A different problem rises when attempting to create multi-source hyperlinks in HTML. One could create an identical link at every source location, but this would create a problem when they need to edit the link (e.g. to change the destination). That problem being that one would have to manually find and edit every link, which is not feasible in the slightest.

While PDF documents can contain links to websites, they can also contain cross-reference links for navigating in the document itself. For example, PDF documents provide the structure to allow users to jump to a specific page or section by clicking on the title in the table of contents or to navigate to a reference by clicking on the citation or to jump to a footnote of a page by clicking on the number of the footnote. These links can be visualised in different ways by various PDF viewers. For example they can be made noticeable by changing the cursor icon while the user hovers above it or by surrounding clickable content by coloured borders.

Aside from the aforementioned document formats, there are many more types of document formats which provide linking features, whether or not it is internal or remote linking. In Table 2.1, the linking mechanisms of different file formats and their abilities is portrayed [28]. What can be deduced from this table, is that every evaluated document format can link to web resources and to entire party documents but in a limited, unidirectional way. XML documents are the only document type that also support bi- and multidirectional links (via XLink). The problem with unidirectional links is that targeted resources or documents have no way of knowing that they are being linked to and thus have no way to link back to the target structures. Another important limitation of embedded links in HTML pages is that only the author of the document can manage the links. It is not possible to create custom links in a web resource one does not have permissions for. This is the problem created by embedding links in documents themselves, since they might only be accessible to a very select group of people. While most of the aforementioned file formats (except OOXML) can also link to entire third-party documents, cross-document links to specific parts (or selections as we will call them) are not supported by their current linking mechanisms. A PDF document, for example, cannot link to an image in a Word document and a Word document cannot create a particular link to a selection of text in a TXT file.

2.2.1 Resource Selector Link Metamodel

The Resource Selector Link metamodel allows hypermedia systems to handle extensibility and store data in a structured way. An application implementing the RSL metamodel can easily extend itself with new resource types by adding the corresponding plugins. The RSL metamodel consists of three main parts; resources, selectors and links. Their components are grouped as subclasses of a more general term *entity*. This allows us to specify that a link can have multiple target- and source entities, which in turn can again be links. The RSL metamodel has been implemented in several hypermedia applications, such as the iServer [26] and the open cross-document linking service [28].

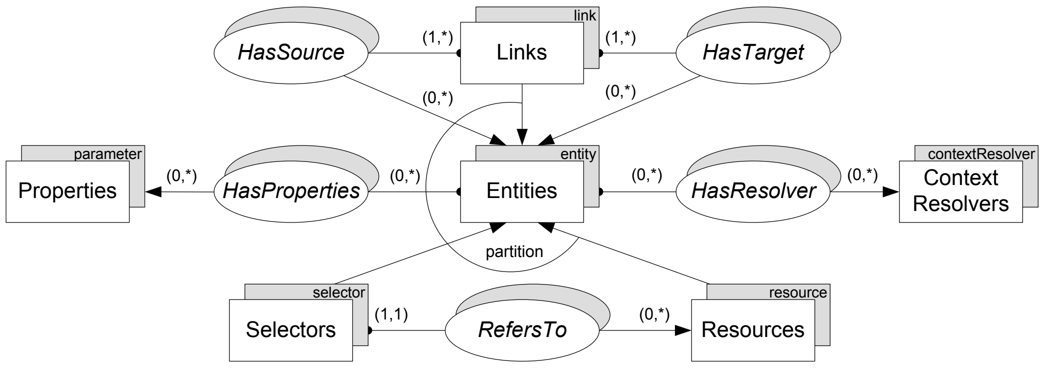


Figure 2.4: RSL metamodel core components, based on [27]

Resources are an abstract concept and represent an existing media type in the system. This could, for example, be a PDF document, an image or a video. Since we sometimes only want to link to a part of a resource, the notion of selectors has been introduced. A selector belongs to only one resource (but a resource can have multiple selectors) and addresses a part of the resource. There are different kinds of behaviour amongst selectors, according to the resource type. A selector for a text file could simply be a start- and end-index, while a selector for an image could be a rectangle, placed at a certain position in the image. Lastly, there is the third kind of entities; links, which create associations between entities. We can link two resources, two selectors, a resource and a selector, two links, a link and a resource, and so on. The RSL metamodel states that a link can have multiple sources and multiple targets, allowing links with more than two participants.

This metamodel brings a lot of powerful features to Table 2.1: *context resolvers* are one of them. A context resolver is a function that can be associated with an entity (resource, selector or link) and returns a boolean value which indicates whether or not the entity should be visible. A link can therefore have a different context resolver on each of its targets, which will be shown depending on the right circumstances. This allows us to control the link behaviour in different situations.

Properties are also an important factor of the RSL metamodel. As one can see on the left hand side in Figure 2.4, an entity can have any number of properties. Rather than having pre-defined (and thus fixed/static) properties in the metamodel, RSL entities can have properties associated with them in the form of key/value pairs. This concept contributes to the flexibility of RSL.

The metamodel is also able to manage user rights. For this, it differentiates between creators, users that have access and users that do not have access. Since this feature is defined at the entity level, which can be seen in Figure 2.5, an application can assign user rights to resources, selectors and links. Each entity is modelled to be created by only one individual. RSL classifies users as individuals or groups, whereby a group can consist of individuals or even other groups. The relationships between users and entities can be denoted by `CreatedBy`, `AccessibleTo` and `InaccessibleTo`.

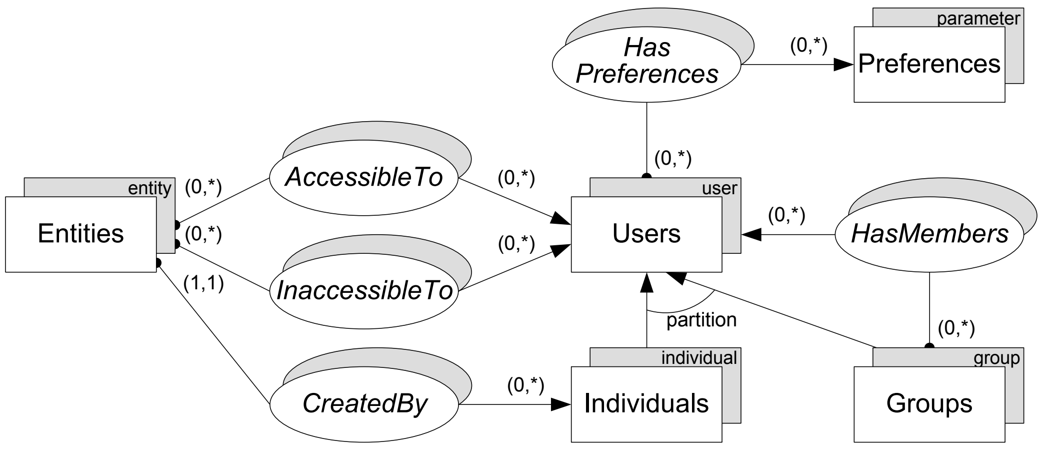


Figure 2.5: User rights- and access management, based on [27]

Sometimes two selectors of a same resource overlap, for example when a first selection spans over a paragraph and a second selection specifies only one word within that paragraph. This would be a problem when a user clicks on that word within the paragraph. In order to resolve which link to follow, the RSL metamodel introduces layers. Each selector is placed on a different layer and a resource can have zero to many layers, which can be activated and deactivated to view the different selectors. An application visualising a resource can thus get all the layers and provide the user with opens to filter or switch between different layers in case they overlap.

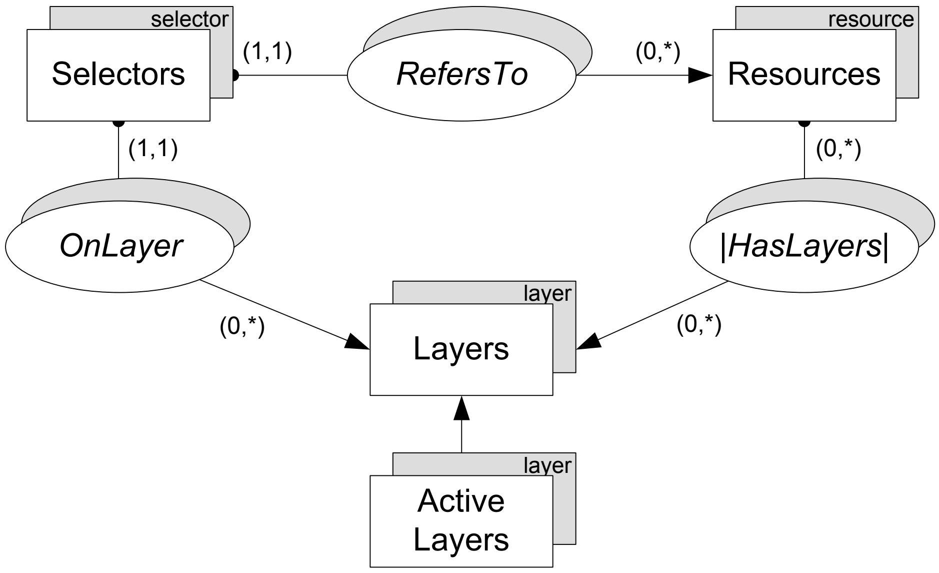


Figure 2.6: Participation of layers in the RSL metamodel, based on [27]

2.2.1.1 RSL Implementations

An open cross-document linking and browsing service has been presented [28]. The link service has been presented as a solution to the limitations of contemporary open hypermedia systems and integrated document linking possibilities. The application has been designed to easily support current document types, as well as be extensible for emerging ones. In order to support modularisation of the two plugins responsible for supporting various document types, the link service makes use of the Open Service Gateway initiative (OSGi) framework [11]. This Java framework allows modularisation and dynamic extensibility, so that functionality can be added to a system without needing to redeploy it. The OSGi is also used in the Eclipse Integrated Development Environment (IDE) to install new software modules without the need to redistribute a new version of the application.

"The modularity layer of OSGi provides a mechanism for dividing a system into independent modules, known as bundles, that are independently packaged and deployed and have independent lifecycles" — OSGi and Equinox: Creating Highly Modular Java Systems [19]

The link service can open documents of supported types by either local visual plugins or third party applications (such as the Google Chrome[3] browser, for example). Links can be created via the user interface and they are stored in a local database. The external managing of links solves the limitations of embedded links, discussed earlier in this chapter. The application makes use of an RSL metamodel implementation and this general structure can be seen in Figure 2.7.

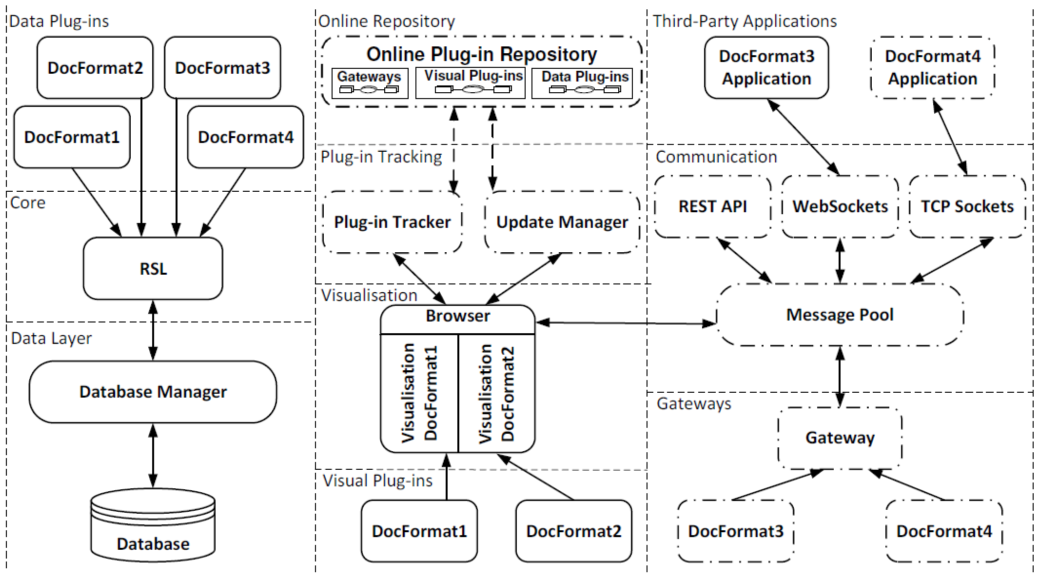


Figure 2.7: Internal structure of the linking service, based on [29]

A screenshot of the link service can be seen in Figure 2.8. In this figure, one can observe three opened documents, namely a text (.txt) file at the top left, a Wikipedia web page (opened via a third-party viewer, Google Chrome) at the bottom left, and a PDF document at the right hand side. The link sources and targets are visualised by yellow highlights. In order to create links, one must first select a source and the application will present a button to confirm the selection. (Not visible in the screenshot.)

[3]https://www.google.com/chrome/

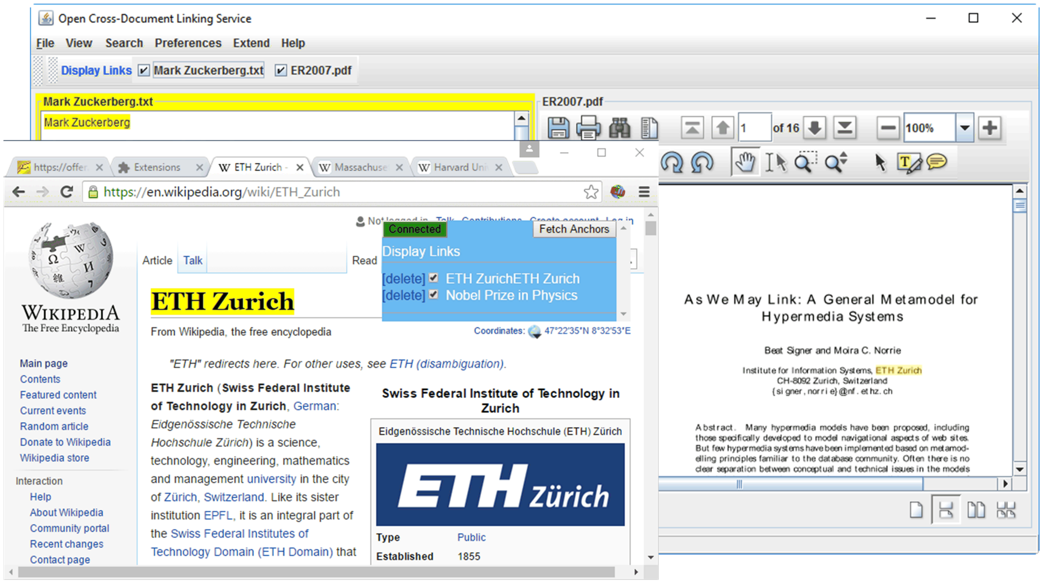

Open Cross-Document Linking Service
File View Search Preferences Extend Help
Display Links Mark Zuckerberg.txt ER2007.pdf
Mark Zuckerberg.txt
Mark Zuckerberg
ER2007.pdf
https://en.wikipedia.org/wiki/ETH_Zurich
Connected
Fetch Anchors
Display Links
[delete] ETH ZurichETH Zurich
[delete] Nobel Prize in Physics
ETH Zurich
As We May Link: A General Metamodel for Hypermedia Systems

Figure 2.8: Screenshot of the open cross-document linking and browsing service

2.2.2 XLink

XLink or the XML Linking Language is a standard defined by the World Web Consortium[4] in an attempt to improve the linking in XML documents. The first version of the XLink standard was released in 2001 [9] and was updated 9 years later to version 1.1 [10], which is also a W3C recommendation and fully backwards compatible with version 1.0. This standard was not intended to be used as a standalone technology but rather as an implementation by other systems. XLink specifies which attributes need to be present in elements to create valid links. The XLink namespace must be present in an XML document that wishes to use XLink attributes. The name is conventionally prefixed by the tag `xlink` and should have the following value: `http://www.w3.org/1999/xlink`

Links formed with valid XLink attributes can be divided into two types: simple and extended links. These types and other XLink topics will be discussed in the following subsections.

[4]https://www.w3.org/

2.2.2.1 XLink Links

A simple XLink link consists of exactly one source and one target. This type of link is unidirectional and can best be compared to an HTML hyperlink, which it was designed to mimic. An example of a simple link embedded in an XML document is provided in Listing 2.4. Notice that an element with the `xlink:type` attribute transforms the element into the specified value of that attribute. A simple XLink link is only valid to the XLink standard if it contains the attribute `xlink:type="simple"`, and an attribute `xlink:href` pointing to the target document. In this case, the type of this XLink element is *simple*, for it is a link without many possibilities. The example shown in Listing 2.4 presents an XML document containing customers, where the second customer element contains a simple link that targets the website of that customer.

```
<?xml version="1.0"?>
<customers xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer id="123">Asus</customer>
  <customer id="456" xlink:type="simple" xlink:href="www.hp.com">HP</customer>
</customers>
```

Listing 2.4: Simple embedded XLink in an XML document

Extended links demonstrate XLink's ability to create more complex links. An element with the extended link status, i.e. attribute `xlink:type="extended"`, is a container for an arbitrary amount of nodes participating in the link. These sources and targets are defined by means of resources, locators and arcs. A resource is a reference to a local file, whereas a locator can reference a remote document. These resources and locators can be seen as nodes in a graph link structure, which can be interlinked via arcs. An arc defines a connection between two labelled locators. It specifies what the source and targets are as well as which traversal behaviour is linked with this connection. The locators are referenced via their `xlink:label` attributes. By defining multiple arcs, an XLink link can form bidirectional, multi-source and multi-target compositions. The target (or source) of an arc can also be another link. A simple link is in fact a special form of an extended link, where there is exactly one arc, coming from the source and going to the target. In Listing 2.5 we show two examples of extended links in an XML file.

```
<link id="123" xlink:type="extended">
    <locator xlink:label="word" xlink:href="word.doc" xlink:type="locator"/>
    <locator xlink:type="locator" xlink:href="pdf.pdf" xlink:label="pdf" />
    <arc xlink:type="arc" xlink:from="word" xlink:to="pdf" />
    <arc xlink:type="arc" xlink:from="pdf" xlink:to="word" />
</link>

<link id="456" xlink:type="extended">
    <locator xlink:href="steve.txt" xlink:label="parent" xlink:type="locator"/>
    <locator xlink:href="sarah.txt" xlink:label="parent" xlink:type="locator"/>
    <locator xlink:href="peter.txt" xlink:label="child" xlink:type="locator"/>
    <locator xlink:href="lilly.txt" xlink:label="child" xlink:type="locator"/>
    <arc xlink:type="arc" xlink:from="parent" xlink:to="child" />
</link>
```

Listing 2.5: Two extended XLink examples

The first link in Listing 2.5 is a single-source, single-target bidirectional link. There are two locators specified and two arcs are used to create the bidirectionality. The second extended link specifies an arc traversal from each parent to each child. Since there are multiple locators with the parent label, each one of them will be taken as a source of the link. Evidentially, this is the same scenario for the (child) targets.

2.2.2.2 XLink Link Elements

An XLink resource is an element that represents a local file and can be interlinked via arc traversal (much like normal locators). Resources are identified via the attribute `xlink:type="resource"`. A resource does not necessarily have to contain any content: it can also be empty and function as a starting resource for a link. The resource element can hold `role`, `title` and `label` attributes. The label attribute is used by arcs to refer to the different resources. Title and role, however, are semantic attributes, meaning that they optionally specify the meaning of a resource in a link. This can be done by specifying a link to a schema.

As mentioned above, an XLink locator object is a representation of a document. Locator elements are recognized via the `xlink:type="locator"` attribute, no matter how the XML element itself is named (e.g. `loc` or `locator`). At the minimum, they require the `xlink:href` attribute to be present, so that the location of the document is known. If the locator points towards an XML-based document, an XPointer expression can be appended to the *href* value to select a particular element inside the resource. An example inspired by a presentation of the W3C is shown in Listing 2.6.

```
<locator xlink:type="locator" xlink:href="http://www.w3.org/#xpointer(id('
    bradford')/li[3])" />
```

Listing 2.6: Locator element referencing a part of a website via an XPointer expression

This locator references the third element of a list with id ‘bradford’ on the website of the W3C. In other cases, a simple pound character (#) can be used to refer to a part of a document, if the id exists. This is how HTML pages can make references to part of their documents.

```
<locator xlink:type="locator" xlink:href="https://en.wikipedia.org/wiki/XLink#
    Extended_links" />
```

Listing 2.7: Locator element referencing a specific section of a website via native HTML referencing

Arcs, in the context of linking with XLink, specify the traversal behaviour between the nodes of a link. These nodes can be either resources or locators. An arc element is considered a valid XLink arc if it contains the ‘from’ and ‘to’ attributes (i.e. `xlink:type="arc"`, `xlink:from` and `xlink:to`). The value for the ‘from’- and ‘to-’ attributes must be labels from resources and/or locators located within the same element that forms the extended link. Extra behaviour can also be specified via the XLink `xlink:show` and `xlink:actuate` attributes. Possible values for these fields are `onLoad`, `onRequest`, `new`, `replace` and `embed`.

2.2.2.3 Linkbases

The XLink standard defines a very practical way for XLink applications to store and access links from external link databases, called *linkbases*. When using linkbases, links are stored externally and there is a healthy separation of links and content. The term healthy separation refers to the various drawbacks related to embedded links, which have been mentioned previously. The linkbase must be an XML file, as stated in the standard. Linkbases make management of links an easier task since the links in this XML file can easily be read, edited, shared, exported and imported. An application retrieving arcs from a linkbase must be cautious for circular link structures. This can occur when a document links to a second one, the second one links to a third one, and the third one links back to the first document. An example of a linkbase can be found in Appendix A.6.

2.2.2.4 XLink Implementations and Applications

In an attempt to provide a layer between a Simple API for XML (SAX) parser and Java applications working with XML documents, XLinkFilter has been developed. It is meant to be an open source project which isolates Java programs from the specific details of the XLink specification. The code is available for download on the XLinkFilter website[5]. Note that the website has last been updated at the end of 1998, three years before XLink became a W3C recommendation. The filter is derived from John Cowan's ParserFilter, which has been removed from their website. An attempt has been made to inquire about this, but was left unanswered.

SXLink is the XLink implementation and Scheme Application Program Interface [15, 16]. It fully supports the XLink interface and provides an API to process multiple XLink documents with the Scheme programming language. The API provides methods for links validation, which could be accurately summarised as traversing the link's nodes (resources, locators) to see if they all still exist. The API can retrieve remote documents through the standard HTTP protocol for this. SXLink can also be used for link resolution. This is the process of changing all XLink links to the outbound format, resulting in a so-called resolved document. This resolution can be done to improve traversal between documents, as the source of every related link can be found in the document itself. Another available SXLink feature is node inclusion, which will substitute the source of a link for the target document. The output documents are represented in the SXML [14] format, representing XML files in terms of symbolic expressions.

A lightweight application called xlinkit has also been developed that generates XLink links from XML resources and a set of rules [22]. The resulting XLink links are put into a linkbase, which can be browsed by certain applications. According to the paper describing xlinkit, the xlinkit website[6] used to contain some examples and demonstrations but this is no longer the case at the time of writing.

Goate is a HTTP proxy designed to handle the advanced XLink linking features in standard HTML browsers [18]. The proxy attempts to map XLink features like bi- and multidirectional links to the simple HTML anchor tags (`<a>`). To circumvent the problem that most documents on the internet do not provide write access, Goate acts as a proxy that transforms documents before they are loaded and shown in the browser. For multi-target links, Goate creates a pop up list with the destinations, which allows users to choose where they want to be directed to. A screenshot of such a pop up list can be seen in Figure 2.9.

[5]http://www.simonstl.com/projects/xlinkfilter/
[6]http://www.xlinkit.com

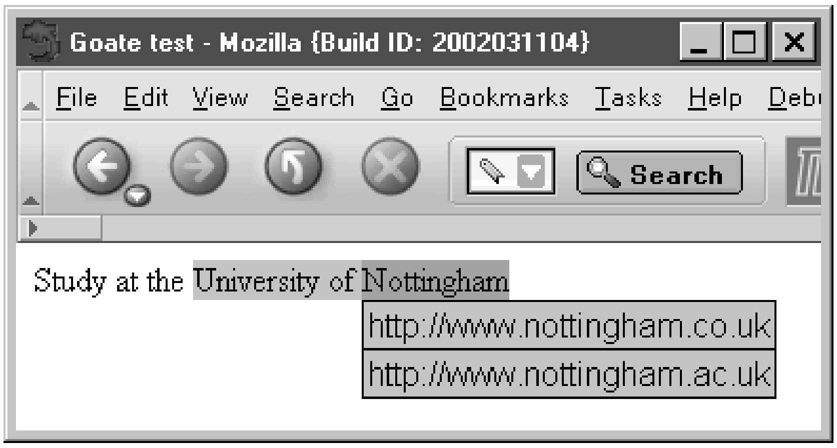
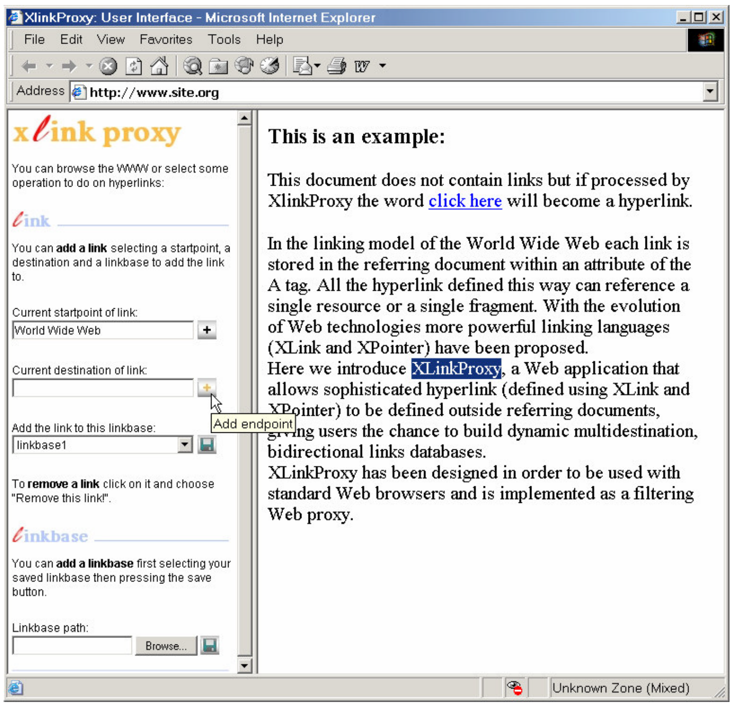

Goate test - Mozilla {Build ID: 2002031104}
File
Edit
View
Search
Go
Bookmarks
Tasks
Help
Search
Study at the University of Nottingham
http://www.nottingham.co.uk
http://www.nottingham.ac.uk

Figure 2.9: Screenshot of the Goate destination pop up lists, based on [17]

Just like Goate, XLinkProxy is a web application that makes use of an HTTP proxy filter to support the functionalities of XLink in web browsers [5]. It uses external linkbases and alters the HTML documents with links while they are being loaded. A screenshot of the XLinkProxy interface can be seen in Figure 2.10. The figure shows an intuitive user interface in the left frame, where a user can add links, remove links and add linkbases.



You can browse the WWW or select some operation to do on hyperlinks:

link

You can **add a link** selecting a startpoint, a destination and a linkbase to add the link to.

Current startpoint of link: World Wide Web

Current destination of link:

Add endpoint

Add the link to this linkbase: linkbase1

To **remove a link** click on it and choose "Remove this link!".

linkbase

You can **add a linkbase** first selecting your saved linkbase then pressing the save button.

Linkbase path: Browse...

This is an example:

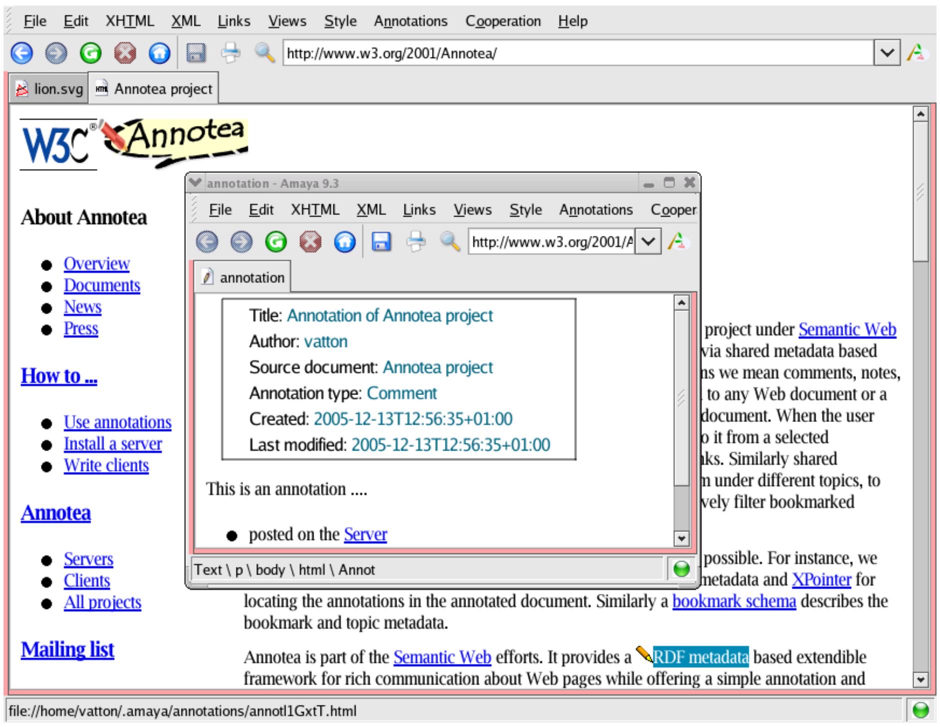
This document does not contain links but if processed by XlinkProxy the word click here will become a hyperlink.

In the linking model of the World Wide Web each link is stored in the referring document within an attribute of the A tag. All the hyperlink defined this way can reference a single resource or a single fragment. With the evolution of Web technologies more powerful linking languages (XLink and XPointer) have been proposed.
Here we introduce XLinkProxy, a Web application that allows sophisticated hyperlink (defined using XLink and XPointer) to be defined outside referring documents, giving users the chance to build dynamic multidestination, bidirectional links databases.
XLinkProxy has been designed in order to be used with standard Web browsers and is implemented as a filtering Web proxy.

Unknown Zone (Mixed)

Figure 2.10: Screenshot of the XLinkProxy interface, figure from [12]

Another big XLink application is the open source tool called the Amaya web editor. The free Amaya browser, made possible by the W3C, allows users to directly annotate and/or edit documents in the browser. The motivation behind its creation was to combine as many standards as they could, including the XLink standard. A screenshot of the tool creating an annotation is provided in Figure 2.11.


File Edit XHTML XML Links Views Style Annotations Cooperation Help
http://www.w3.org/2001/Annotea/
lion.svg
Annotea project
W3C Annotea
annotation - Amaya 9.3
File Edit XHTML XML Links Views Style Annotations Cooper
http://www.w3.org/2001/A
annotation
Text \ p \ body \ html \ Annot
file://home/vatton/.amaya/annotations/annotI1GxtT.html

About Annotea

- Overview
- Documents
- News
- Press

How to ...

- Use annotations
- Install a server
- Write clients

Annotea

- Servers
- Clients
- All projects

Mailing list

Title: Annotation of Annotea project
Author: vatton
Source document: Annotea project
Annotation type: Comment
Created: 2005-12-13T12:56:35+01:00
Last modified: 2005-12-13T12:56:35+01:00

This is an annotation ....

- posted on the Server

project under Semantic Web
via shared metadata based
ns we mean comments, notes,
to any Web document or a
document. When the user
o it from a selected
ks. Similarly shared
n under different topics, to
vely filter bookmarked

possible. For instance, we
metadata and XPointer for
locating the annotations in the annotated document. Similarly a bookmark schema describes the bookmark and topic metadata.

Annotea is part of the Semantic Web efforts. It provides a RDF metadata based extendible framework for rich communication about Web pages while offering a simple annotation and

Figure 2.11: Screenshot of the Amaya browser creating an annotation

2.2.2.5 XLink Limitations

We have already discussed some current limitations of the XLink specification. The most prominent problem being that XLink has been designed to navigate through XML-based documents, i.e. document types that have a coherent tree structure. There is no provided support for other file types, such PDF files, music or videos. In Section 3 we propose a solution to this limitation. XLink does allow overlapping sources yet does not specify any way of displaying this to the viewer or resolving which one is activated first. Unfortunately, XLink has not been able to reach the surface of the Web and remains unknown to many. Nonetheless, there are several notable implementations of XLink, with its biggest documented use by the Extensible Business Reporting Language (XBRL) and the Amaya browser.

2.2.2.6 XPointer

The XPointer [7] standard was designed to address parts of XML-based documents. It uses XPath [6] expressions to navigate in these documents. Elements, or ranges, can be selected via their ID or via relative position towards each other and/or their parents. An example of a simplified linkbase is shown in Listing 2.8. Via the XPointer expression shown in Listing 2.9 an application can retrieve the link element from the linkbase with identification number *1234*.

```
<linkbase>
    <link id="1234" ... />
    <link id="2345" ... />
    <link id="3456" ... />
</linkbase>
```

Listing 2.8: Example of a simplified linkbase layout

```
xpointer(id("1234"))
```

Listing 2.9: XPointer expression grabbing an object via their identification number

In order to navigate in a valid XML-like structured document, one can also use an XPointer expression to follow a certain path of elements. When such paths start with one forward slash (/), the evaluation begins at the root of the document. A double forward slash (//) can start from any point in the document that is still valid for the whole expression. Instead of the name of an element, an index *N* can be given to select the n-th child of an element. Note that some expressions can produce a collection of results rather than a single item.

```
xpointer(/linkbase/link)
xpointer(//arc)
xpointer(/1/1)
```

Listing 2.10: Different XPointer expressions

XPointer is currently not supported by browsers but it can be used via XLink in specific situations. If the target (or locator) of an XLink link points to an XML document, an XPointer expression can be appended to select a part of the document. This is accomplished by adding `#xpointer(*)` to the end of the URL and replacing the asterisk by a valid XPointer expression. An example is given in Listing 2.11, where a hyperlink is made to target the element in a linkbase with id *1234*.

```
<a xlink:type="simple" xlink:href="linkbase.xml#xpointer(id('1234'))"> Link 1234
    </a>
```

Listing 2.11: XPointer expression used to reference a link by their ID

The XPointer specification has a built-in fail-safe mechanism that allows us to pass multiple expressions (which are evaluated from left to right), which will move to the next expression if the first one fails. These expressions are divided by regular spaces. This mechanism could prove to be useful for specifying a backup selection in case the first expression should fail. Suppose a linking application contains a link to a selection of a document. The application could append a fail-safe expression that selects the whole document, in case the first expression could not address the selector. Listing 2.12 demonstrates how a selector could then be structured. The example depicts a scenario of a selector that wants to target the selector of another link. If this XPointer expression should fail, the entire link would then be addressed.

```
<selector  xlink:type="selector"
    xlink:href="linkbase.xml#xpointer(/link[id='1234']/selector[0])
                linkbase.xml#xpointer(/link[id='1234'])" />
```

Listing 2.12: Example of a selector addressing another selector, with a fail-safe

2.3 Comparison of RSL and XLink

In this section, we present a comparison between the RSL metamodel and the XLink standard. The criteria for this comparison are summarised in Table 2.3 and include various features of each of the models. As already discussed in Section 2.2.1, it is clear that the RSL metamodel provides a many features for describing links. It supports advanced links like bidirectional links and links with multiple sources and multiple targets. Since RSL links connect entities, and links themselves are entities, links that connect links with each other are also possible. The concept of these *links over links* is shown in Figure 2.12. The central link object, coloured in blue in the figure, connects two other links (which are coloured in green). One can observe that each link object is an entity (illustrated via the dotted lines), which allows such structures to be formed.

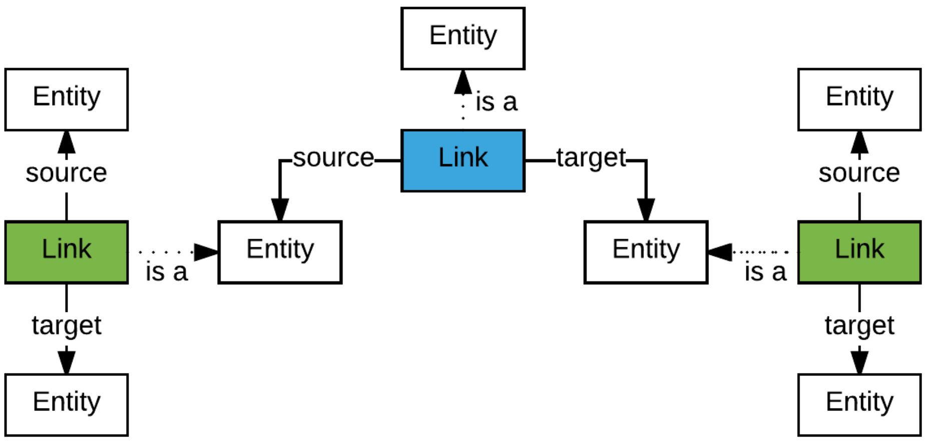


Figure 2.12: Main (blue) RSL link connecting two other links (green)

The same concept for a possible XLink implementation can be seen in Figure 2.13. In the figure three links in a linkbase are visualised. Once more, the colour blue is used to highlight the main link and green the other links. To connect the other links, the locators of the main link can refer to the linkbase file and the ID's of the links inside. The linkbase principle allows for XLink links to be stored out of line and to be shared via the accessible XML files.

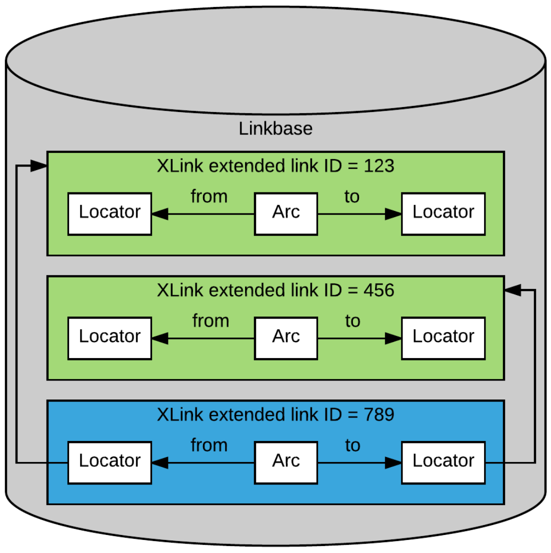

Linkbase
XLink extended link ID = 123
Locator
from
Arc
to
Locator
XLink extended link ID = 456
Locator
from
Arc
to
Locator
XLink extended link ID = 789
Locator
from
Arc
to
Locator

Figure 2.13: Main (blue) XLink link connecting two other links (green)

As the properties can be associated with any entity, seen in Figure 2.4, the RSL model can be easily extended with extra features. If, for example, an application using the RSL metamodel wants to add the features of expiring links (links that will be deleted after a certain period of time), an extra property could be added to the ‘entity’ element, named *expires-at*.

For the powerful W3C recommended linking model XLink, these properties are not part of the supplied set. Despite the big limitation that XLink was created for linking in XML-based documents, its strength partially comes from the ability to create complex linking structures via locators and arcs. The locators and arc elements can be viewed as nodes and arcs of mathematical graphs. As an XLink link structure can contain an arbitrary amount of locators and arcs, the links can have any form from simple single-source and single-target to multi-source and/or multi-target. Even though the RSL metamodel does also allows the creation of bi- and multidirectional links, it does not suffer from the limitation of not being able to support every document type. A large amount of document types cannot describe a part of themselves via XPointer expressions (which are used by XLink). Table 2.2 lists various types of document types and how a corresponding selection can be made.

Document Type	Selector Format
Text	Start- and end- index
Image	X,Y position from top-left corner, width and height of rectangle
Video	Start- and end- time
PDF	Page index, X,Y position, width and height of rectangle
XML	XPointer / XPath expression
Excel	Expression selection various row/column/cell combinations

Table 2.2: Different selection strategies in various document types

Through the user management component supported in the RSL metamodel, a link service implementing RSL can introduce the functionality of user access rights. Via these access rights, users can have ownership of entities (such as links) and an entity can also associate which users can or cannot access it. A scenario is sketched in Figure 2.14 where an individual (Ind. C) created two links. One link that is accessible to anyone in one of the two groups and on that is only accessible to another individual (Ind. D). Individual D in this example does not have access rights to the first link created by individual C. User rights introduce many possibilities in regards to individual users and groups. A system that can distinguish or manage rights differently per person can provide the option to create private, restricted and fully public links. In the XLink standard this functionality is unfortunately not provided. All standard XLink links would thus be available to all users.

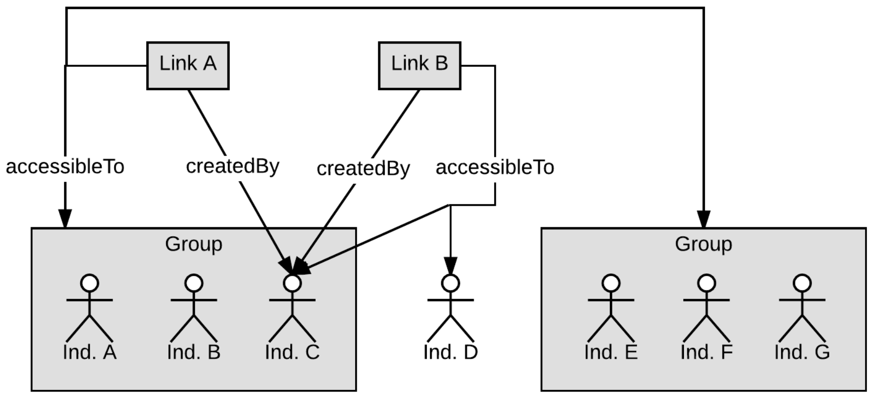


Figure 2.14: Example of links with different user access rights

Both linking models store links externally. In RSL a link simply points to other source- and target entities. Figure 2.13 an XLink linkbase example shows the effective use of a linkbase. Both linking models perform the good behaviour of not embedding links in the documents themselves, but rather storing them separately. The disadvantages of embedded link have already been discussed in Section 2.1 and will not be repeated in this section. For the comparison it suffices to note that both models support external links.

Extensibility for the features (and similar ones) is not specified in the XLink recommendation but attributes and elements may be added to XML elements containing XLink links, as long as they remain valid according to the standard. By defining specific attributes, extra functionality can also be added to XLink links without breaking the important notion of being a standard.

The limitation of XLink only being able to link in XML-based documents explains why it is infrequently used. Furthermore, the lack of special built-in features is another indicator. By making use of the possibility to add elements and attributes to XLink links, the XLink standard might gain a lot more appreciation than it has so far garnered.

An overview of the previously mentioned comparison features can be found in Table 2.3.

Feature	RSL metamodel	XLink standard
Advanced (complex) links	Supported	Supported
Solution for overlapping selectors	Yes, via layers	Supported but no solution provided
Support for every document type	Yes	No, XML-based document types only
Separating links from content	Yes	Yes
Recommended standard	No	Yes
Usage since introduction	Frequent	Seldom
Extensibility	Yes	Yes, via additional attributes

Table 2.3: Comparison of RSL and XLink features

2.4 Summary

Throughout this chapter, we have looked at the history of linking in documents. Early document types started with no linking features, after which more and more features were introduced. We discussed that embedded and unidirectional links are too limited for the realisation of more advanced links, such as bi- and multidirectional, multi-source and multi-target links. Open hypermedia projects have in their own way tried to change the way links are used and created, for example by storing the links outside of the documents themselves. However, this always introduced new limitations of their own. We have seen the emergence of the XLink standard, published by the W3C, and how its limitation (only being able to link to XML-based document types) has prevented the standard from being more widely used. The RSL metamodel has proven to be a powerful, extensible and useful linking model with a lot of extra features. We compared the XLink standard and the RSL metamodel and concluded that the XLink standard has a lot of potential if it were able to link to every document type.

3 An Enhanced XLink Standard

In this chapter we discuss several strategies of how the XLink standard can be enhanced to be used for cross-document linking in all document types. Not only analysing the advantages and disadvantages of each approach, an investigation will also be conducted on how important linking features presented by the RSL metamodel can be included.

3.1 Embedded XLink Links

The first stage of research revolved around inserting the enhanced XLink links in the resources where they originate from. Several document formats already adopted the technique of storing links inside the documents themselves, including Microsoft's Office Open XML (OOXML), the OpenDocument [31] (ODF) ISO standard and DocBook [30]. An example of an embedded link in the DocBook format, provided by DocBook: The Definitive Guide [30], is shown in Listing 3.1.

```
<!DOCTYPE sect1 PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
 "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
<sect1><title>Examples of <sgmltag>Link</sgmltag></title>
<para>
In this sentence <link linkend='nextsect'>this</link> word is
hot and points to the following section.
</para>
<para>
There is also a link to the section called
<quote><link linkend='nextsect' endterm="nextsect.title"/></quote>
in this sentence.
</para>
<sect2 id='nextsect'><title id='nextsect.title'>A Subsection</title>
<para>
This section only exists to be the target of a couple of links.
</para>
</sect2>
</sect1>
```

Listing 3.1: Example of a link in DocBook, based on [30]

Aside from being XML-based document types that do not utilise the XLink standard, the aforementioned document types also only support simple hyperlinks. The beneficial features such as bi- and multidirectional links are completely neglected.

Embedding advanced XLink hyperlinks in XML-based documents would be a great improvement of the way links can currently be created in documents. An example of an extended link in an HTML document is shown in Listing 3.2. In this example, an image is participating in an extended link that targets two websites. If we follow the code from top to bottom, we find an HTML document with an XLink link in it. Inside the link are three locators and one arc. The first locator actually doubles as a standard `img` tag, showing a picture. The following two locators are links to different websites and the arc links the image to the two websites.

```
<html xmlns:xlink="http://www.w3.org/1999/xlink">
<link xlink:type="extended">
 <img xlink:type="locator" xlink:label="F" src="vub.jpg" />
 <loc xlink:type="locator" xlink:label="T" href="www.vub.ac.be" />
 <loc xlink:type="locator" xlink:label="T" href="www.ulb.ac.be" />
 <arc xlink:type="arc" xlink:from="F" xlink:to="T" />
</link>
</html>
```

Listing 3.2: Example of an image on a web page linking to two websites

This method does come with its limitations. As it requires inserting XLink links, the source documents must be XML-based documents and they must be editable by anyone. Thus, it does not provide a valuable solution to the cross-document linking problem (i.e. being able to link between different parts of different documents).

Because the links are embedded, documents containing link targets cannot know which documents are linking to them. The targeted document is unaware of the documents containing links to it. In the example shown in Listing 3.2, the two websites do not know of all the inbound links and can therefore not display any indication that they are being linked to. In order to find all the incoming links, every known document would have to be searched for links, which is practically impossible for obvious reasons. Furthermore, if an application wants to embed links into documents, it will need access rights, which they are usually not granted. The VUB website is a good example of this: one cannot simply edit it to add a link in the code.

Since this technique does not fully support cross-document linking with XLink, we do not go into how other rich linking features (as seen by the RSL metamodel) could be achieved. Instead, the next section entails an overview of storing links in different documents.

3.2 Virtual Documents

To solve the limitations introduced by the first approach, we can extract the links and selectors from the documents themselves and store them in different locations. This can be realised via XLink linkbases, an important part of the XLink standard. In addition to these linkbases, for each linked document we store some additional information like the document's path, filename, as well as its selectors. In this approach, the linking service can resolve a link by accessing the linkbase and reading the virtual documents for the location and selectors of the associated resources. With the file location and selectors known, an application can then open the actual documents and highlight the obtained selectors.

To better explain this process, an example is shown in Figure 3.1, where the virtual documents are coloured in a light-blue color and an example of their contents is given in the yellow note.

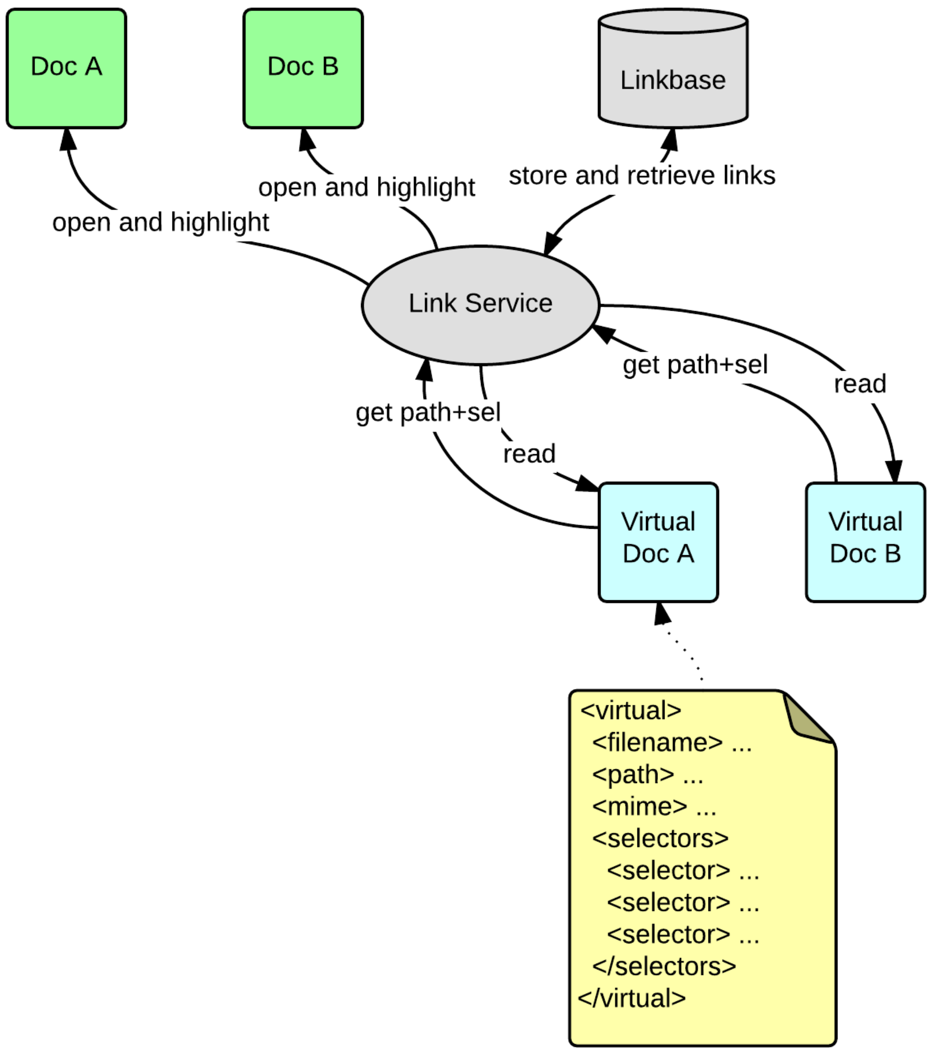


Figure 3.1: Storing information in virtual documents

For every new link or selector an application needs to add information to existing virtual documents or create new ones if they do not already exist. The management of this system can get quite tedious and will require a lot of space when the service is scaled towards a large number of documents and links. Figure 3.1 depicts a lot of interactive steps for simply displaying one XLink.

This approach even works for non-XML-based document types because we store the selector and other data in a virtual document that is an XML file itself. Because of this, a selector can make use of XPointer expressions to grab a certain selector, regardless of the document type it is from. A scenario is presented in Listing 3.3, Listing 3.4, and Figure 3.2, where a linkbase contains a link between two selections of a PDF document. The selectors, which for a PDF document are made up of a page index, an X-position, a Y-position, a width and a height value, are each given an ID when stored in the corresponding virtual document. The *href* attribute of the link locator elements (seen in Listing 3.4) is pointing to the selector information via an XPointer expression.

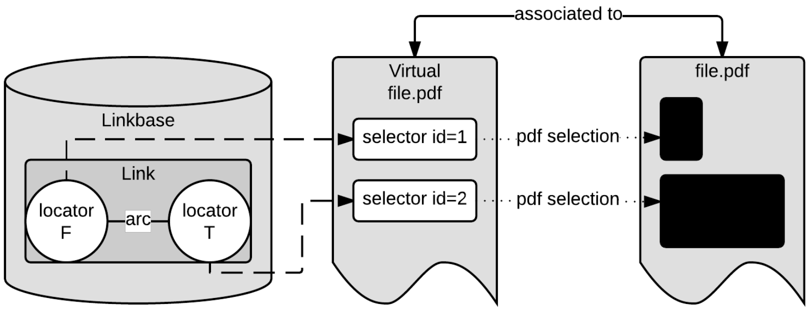


Figure 3.2: Visualisation of the example of a link using virtual documents

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<virtual>
 <id>123</id>
 <path>c:/users/user/documents/file.pdf</path>
 <filename>file.pdf</filename>
 <mime>application/pdf</mime>
 <selectors>
  <selector>
   <id>1</id>
   <page>1</page>
   <x>0</x>
   <y>0</y>
   <width>100</width>
   <height>100</height>
  </selector>
  <selector>
   <id>2</id>
   <page>50</page>
   <x>0</x>
   <y>0</y>
   <width>100</width>
   <height>100</height>
  </selector>
 </selectors>
</virtual>
```

Listing 3.3: How a virtual document could look like

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

 <link xlink:type="extended">
  <locator xlink:type="locator" xlink:label="F" xlink:href="virtual/virtual123.
    xml#xpointer(//selector/id=1)" />
  <locator xlink:type="locator" xlink:label="T" xlink:href="virtual/virtual123.
    xml#xpointer(//selector/id=2)" />
  <arc xlink:type="arc" xlink:from="F" xlink:to="T" />

</linkbase>
```

Listing 3.4: Linkbase link accessing selectors via the virtual document technique

3.3 Links with Extra Attributes

We can eliminate the need to look up a file location from the path in the virtual documents by storing the real location in the links themselves as added metadata. Bullet 4.4 of the XLink standard[1] states that elements with XLink attributes are allowed to add non-XLink attributes, while remaining a valid standard. We use this fact to append all the data from the virtual documents in the elements containing the links in the linkbase. While we can just create attributes to specify the URI, the name and the MIME type, the problems remains of how to specify the selector data, since most file types rely on different selection strategies.

We have investigated whether or not it is possible to specify the selector data via XPointer expressions. Unfortunately, this standard has been specifically designed to navigate through the structure of parsed XML and tree-structured documents. There are no possible arguments that could be passed to an XPointer expression that could make it address parts of documents that are not XML-based. As we cannot extend XPointer for navigating non-XML based document types, this idea was disregarded as a possible solution. Looking into the XML language, a special node type which can hold instructions for an application was found, called Processing Instructions (PI). These special XML nodes can be retrieved via special XQuery [3] and XPath expressions. An XML processor ignores these instructions because they are meant for the application and the processor would not be able to make sense of them. Processing instruction elements start with "<?" and end with the "?>" tag. A processing instruction is made up of a name and a target. Since these instructions must be queried and cannot be retrieved in context, the name should be an unique identifier. This can be achieved by formulating an al-

[1]https://www.w3.org/TR/xlink/#integrating

gorithm to name the processing instructions, like concatenating the link ID, the locator label and a tag (`selection`). We can then retrieve the selector data via XPointer expressions. A link using PI to define selector information could then look like the snippet presented in Listing 3.5.

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

    <link id="WORDtoPDF" xlink:type="extended">
        <locator xlink:type="locator" xlink:href="word.doc" xlink:label="from" />
        <locator xlink:type="locator" xlink:href="file.pdf" xlink:label="to">
            <?WORDtoPDF_to_SELECTOR x=50 y=100 w=200 h=300 ?>
        </locator>
        <arc xlink:type="arc" xlink:from="from" xlink:to="to" />
    </link>
</linkbase>
```

Listing 3.5: Linkbase example

Then, from the application we can query the selector information from the link by checking each locator if they contain a processing instruction. The query for the previously described selector will take the following XPointer expression:

```
processing-instruction()[name()="WORDtoPDF_to_SELECTOR"]
```

Listing 3.6: Processing Instruction example

This method, however, also suffers from a couple of drawbacks, such as the fact that we need to be able to validate the processing instructions for validity and security. While it is possible and good practice to validate an XML file (such as a linkbase) via a schema definition (.XSD file), processing instructions are simply ignored by the validator and can therefore present a risk factor when its code is executed in the application. This leads us to embedding the selector metadata in the link itself as extra attributes. Since the data required by selectors of different document types can vary, the validating XML schema must be able to distinguish these differences. For this, we make use of XML instances, which will specify which attributes are needed, depending on the value of the MIME type.

In order to have a complete mapping of the enhanced XLink standard with the RSL functionality, our proposed solution will need to support the same features. We now discuss those features, which have also been described in chapter 2, followed by a section about the validation schema and its extensibility.

3.3.1 Overlapping Selectors

In the RSL metamodel [27], the concept of layers has been introduced to handle the problem of certain selectors overlapping each other. An example is that a paragraph could be the anchor of one selector and a word inside that paragraph could be the anchor of another selector. If that word would be activated (clicked), we would have to resolve which link the user meant to follow. The XLink standard does allow selectors to span over the same content but provides no solution for resolving the links. We tackle the problem of overlapping links in the same way the RSL metamodel does: by placing each selector on a different *layer*. In a linkbase we can assign a new layer level as an additional attribute to a newly created link or selector. This way an application can retrieve the layer attributes and find an appropriate way to visualise it.

Listing 3.7 shows a link in a linkbase with two locators that overlap. The first locator (of a text file) selects all characters between index 200 and 540, whereas the second locator selects interval 290 to 300, which causes overlap with the first one. Each locator element has received a different value for the layer-attribute so the application can handle it.

```
<linkbase xmlns:xlink="http://www.w3.org/1999/xlink">
<link xlink:type="extended">
 <locator xlink:href="url/file.txt" xlink:label="textSelectParagraph" layer="14">
  <selector xsi:type="text_plain">
   <from>200</from>
   <to>540</to>
  </selector>
 </locator>

 <locator xlink:href="url/file.txt" xlink:label="textSelectWord" layer="15">
  <selector xsi:type="text_plain">
   <from>290</from>
   <to>300</to>
  </selector>
 </locator>
</link>
</linkbase>
```

Listing 3.7: Introducing layers to XLink locators

3.3.2 Links of Links

In the RSL metamodel every link is modelled as an entity. In this matter, a link can have another link as a target or as a source. This feature can also be supported by our solution because the links are stored in a linkbase, which is an XML document and can be pointed at via an XPointer expression. To create a link of links, the locators have to reference the linkbase (and in particular the ID of the wanted link). In the Listing 3.8, we create a link from an image to the ‘word-to-pdf’ link defined in Listing 3.5. Alternatively, links that link one link to another are also possible.

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

    <link id="IMGtoLINK" xlink:type="extended">
        <locator xlink:type="locator" xlink:href="image.jpg" xlink:label="from"
    />
        <locator xlink:type="locator" xlink:href="linkbase.xml#WORDtoPDF" xlink:
    label="to" />
        <arc xlink:type="arc" xlink:from="from" xlink:to="to" />
    </link>

    <link id="LINKtoLINK" xlink:type="extended">
        <locator xlink:type="locator" xlink:href="linkbase.xml#IMGtoLINK" xlink:
    label="from" />
        <locator xlink:type="locator" xlink:href="linkbase.xml#WORDtoPDF" xlink:
    label="to" />
        <arc xlink:type="arc" xlink:from="from" xlink:to="to" />
    </link>

</linkbase>
```

Listing 3.8: Creating links that connect other links

The links in the above linkbase create a structure that can be seen in Figure 3.3.

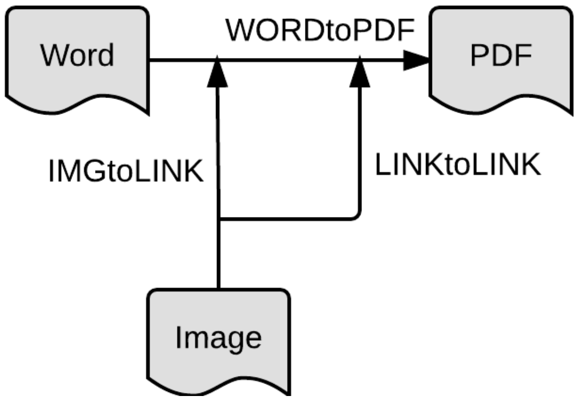


Figure 3.3: Visualisation of a link linking links

3.3.3 Data Ownership and Access Rights Management

The XLink specification lacks any internal support for data ownership management. As we have mentioned before, XLink does allow additional (non-XLink) attributes to be present in elements with XLink attributes, without impairing its standard status. For data ownership, we can manage a database of users and groups (in the same way that RSL handles this). The same is applicable to access rights: these can be specified by defining them in the metadata. Listing 3.9 shows an example linkbase with a public and a semi-private link.

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

 <locator xlink:type="locator" xlink:createdBy="111" accessibleTo="1" />

    <link id="PUBLIC_link" xlink:type="extended" createdBy="127" accessibleTo="1"
    >
     <!-- if 1 were to belong to a public user group -->

        <locator xlink:type="locator" xlink:href="word.doc" xlink:label="from" />
        <locator xlink:type="locator" xlink:href="file.pdf" xlink:label="to" />
        <arc xlink:type="arc" xlink:from="from" xlink:to="to" accessibleTo="" />
    </link>

    <link id="PRIVATE_link" xlink:type="extended" createdBy="127" accessibleTo="
    52">            <!-- 52 might be a special selected users group -->
        <locator xlink:type="locator" xlink:href="word.doc" xlink:label="from" />
        <locator xlink:type="locator" xlink:href="file.pdf" xlink:label="to" />
        <arc xlink:type="arc" xlink:from="from" xlink:to="to" />
    </link>
</linkbase>
```

Listing 3.9: User- and access rights

In Listing 3.9, we see a `createdBy` field which resembles the identifier of the user. The implementation of these identifiers is left to the application using this linking model. They can then also select the users or user group which can access the content of the link. When the user chooses the target group from an interface, the group identifier number is determined and inserted by the system.

3.4 XML Schema

In terms of integrity, we have created one master XML schema (XSD) that defines what valid links must resemble. It specifies exactly how links in the linkbase must be formed, which are valid and which are not. In order to comply to the W3C XLink standard, the links, resources, locators and arcs must provide the required XLink attributes. These are imported in the master schema via the *xlink* namespace (see Section 2.2.2). In addition to the standard attributes, the extra metadata attributes enhancing XLink are also specified in the schema. Links, resources, locators, and arcs can each specify the following attributes: `accessibleTo`, `inaccessibleTo`, `createdBy`. Additionally, the `layer` attribute is also available for selectors. The schema can be used to validate linkbases upon importing/exporting XLink links or to check the integrity of an active linkbase. We dive deeper into the exact creation of this schema in section 4.1.

3.4.1 Selector Plugins

Different file types may have different versions of how they select parts of their documents. Also, emerging document formats would require a way to be able to be integrated into the system. We support this feature in the master schema by having an abstract selector type. This abstract selector validates elements and attributes that are shared for every document type. A new document type plugin schema can then extend this type with its own, custom attributes. We refer to the Appendix A.2 for example plugin schemas. In the linkbase, selectors can specify the type of the selector via `xsi:type`. The name of this selector type would then represent the MIME type of the file type it is meant for, with the limitation that forward slashes are prohibited in XML *CNAME* values. We handle this by substituting the forward slash with an underscore.

3.4.2 Extending the Master Schema

In order to support a new document type selector in the master schema, a type specific schema must be injected into the master schema. This can be done via eXtensible Stylesheets Language Transformations (XSLT) [13]. The master schema will include a schema of document type specific selectors and this schema will be transformed via a XLink plugin tracker (see Chapter 4). The XSLT code for this injection can be seen in Appendix A.4. Removing a plugin from the schema is also possible with a different XSLT code, which can be found in Appendix A.5. Note that the tag `###LOCATION###` is where the application should substitute the location of the plugin schema.

3.4.3 Schema Plugins

The schema plugins must follow certain specifications in order to be valid. First, they must be valid XML schema definitions and secondly, the abstract selector class must be subclassed. For a proof of concept, two representative file types (PDF and TXT) selectors are presented in Appendix A.2. The two schema plugins each specify the MIME type of their respective document format as the name of their type and then extend the `abstractSelector` with their selector attributes. The plugin for text documents specifies two extra attributes, `from` and `to`, which specify a selection in a text document. The PDF selector requires a page index, an X and a Y position and a width and length of the desired rectangle. As we can see from the two examples, creating schemas for new document types is relatively easy and requires little code.

3.5 Summary

In this chapter we have investigated the possibilities for enriching the XLink standard with features of the RSL metamodel. These features include linking functionality such as user rights management, context resolver handling and overlapping selector resolution. In the first approach, we suggested the embedding of XLink links in different document formats. Unfortunately, this resulted in several hard limitations such as requiring the access rights to edit the documents. Apart of the access right, this approach is still limited to XML-based document formats, since only these are able to integrate the XLink links. For these limitations, we explored the idea of creating one XML document per resource, containing the necessary linking information. These XML documents, which we called *virtual* documents, stored information such as the filename, MIME type, location and selectors of the associated resource. In order to create an XLink link between two resources, an application could then refer to the appropriate selector (stored in the virtual XML document) by means of an XPointer expression.

This approach, however, introduced a large amount of steps in the creation of a simple link and the maintenance of these virtual documents would not scale well in an application with an immense amount of links and resources. Further research allowed us to drop the virtual documents, by moving their data into the links as additional attributes. The aforementioned RSL features can also be supported in the links by thoughtfully defining which attributes will be used for what. The final solution then consists of having all the links in linkbases and the extra data, such as the selector information and the RSL features, is embedded in the links themselves via attributes. The validation of these enhanced links can be performed via an XML schema definition, created for the enriched XLink links, which aim to support existing as well as emerging document formats.

4 Implementation

In this chapter we will elaborate on the implementation of the XLink enhancing solution and its integration with the open cross-document link service [28]. The XSD schema that defines and validates our enhanced XLink links will be explained in greater detail, together with the place where document plugin validators are added. Furthermore, the Java implementation for the mapping (via converters) in the application will be discussed.

4.1 Master Schema

The XML schema definition (XSD) document, which in this context will be referred to as the master schema, lies at the basis of the implementation. The final version of the master schema can be found in Appendix A.1. The core of this schema has been created with the requirements of valid XLink links in mind. Conforming to the XLink standard, all the required elements and attributes are stated in the schema. The attributes that enable the enriched linking features (e.g. user access right, layers, ...) have been added in the basic schema. As we have investigated in the research phase, the metadata required for and by selectors will be stored as additional attributes of the element that carries the `xlink:type="selector"` attribute. The complex type declaration for the selector element has been defined as an *abstract* element that contains several common attributes, which are used by every selector (regardless of the document type). Selector schemas specific for new or different document types must extend the abstract base class in order to be correctly used in linking applications that conform to the master schema. Links that do not conform to the master schema are not deemed valid and will be removed by our implementation. Listing 4.1 displays the specification of the abstract selector type. The first attributes defined, are the ones responsible for the enriched features. For instance, via the `accessibleTo` attribute, a selector can specify which individuals or groups are allowed to access it. After these attributes, the XLink related attributes are specified in compliance to the original XLink standard.

```
<xsd:complexType name="abstractSelector" >
xsd:attribute type="xsd:string" name="accessibleTo" use="optional"/>
xsd:attribute type="xsd:string" name="inaccessibleTo" use="optional"/>
xsd:attribute type="xsd:string" name="createdBy" use="optional"/>
xsd:attribute type="xsd:integer" name="layer" use="optional" />
xsd:attribute ref="xlink:type"  fixed="locator" />
xsd:attribute ref="xlink:href"  id="href" />
xsd:attribute ref="xlink:label" use="optional" />
xsd:attribute ref="xlink:titleattr" use="optional" />
xsd:attribute ref="xlink:role"  use="optional" />
</xsd:complexType>
```

Listing 4.1: Master schema snippet: abstract selector type

Two schema plugins of selectors of representative document types are presented and can be found in Appendix A.2. In the plugin for PDF document types, we see an extension of the selector base class (`abstractSelector`) with five additionally required attributes: the page index, X-position, Y-position, width and height.

4.2 RSL Implementation Structure

Before we jump into the implementation of the XLink related classes, we evaluate how the code supporting the RSL metamodel is structured in the core of the open cross-document link creating and visualisation browser. The implementation is completely done in Java and the Eclipse[1] SDK version 4.2.1 is used to run the application. The Resource Selector Link metamodel is the core of the link service and explains why most of the bundles are RSL related. Bundle `com.rsl.core` contains the heart of the RSL metamodel implementation. It includes the basic

[1]https://www.eclipse.org

classes for Entity, Resource, Selector, Link, Layer, User, Group, etc. Communication with the database is performed by the `com.rsl.databasemanager` bundles, and communication to the Web is done via sockets, REST and TCP classes located in the `com.rsl.communication.protocols` bundles. Overall the structure of the RSL implementation is straightforward, so we put focus on the relevant bundles: those in charge of supporting plugins.

In order to add a new document type in the RSL implementation, two types plugins must be introduced. One that is the data plugin and one that is the visual plugin. The data plugin must extend the default `Resource` and the `Selector` classes. This is where appropriate class members can be added (e.g. start- and end index for text files). The visual plugin must subclass the `DefaultPanel` and override the relevant default methods with its document type specific features.

The `com.rsl.userInterface` bundles contains the packages that are responsible for tracking plugins, visualising documents and communicating with the user interface. The two most relevant classes for our enhanced XLink implementation will be `MainGUI`, responsible for starting and initialising the link service, and `iServerInterfaceImp2`, responsible for implementing the RSL interface and the database operations.

4.3 XLink Implementation

In this section we discuss the implementation of the enhanced XLink standard in the link service. By now we have the master schema which specifies how the links should be defined, we have seen how the RSL metamodel has been implemented in the link service and we will add our enhanced XLink component to the system.

4.3.1 Class Generation via Java Architecture for XML Binding

The classes for our enhanced XLink solution have been generated with the JAXB schema compiler (xjc). This tool, included in the Eclipse SDK, is used to create a mapping between XML schemas and Java classes annotated by the Java Architecture for XML Binding (JAXB). The generated output of xjc is a collection of Java classes that all have JAXB annotations. These annotations allow an application to easily serialise object instances to XML documents and vice-versa. By using this technology, with the master schema, the implementation would be immediately ready to be integrated in the link service.

In order to be able to generate the classes via xjc, the original XLink schema definition[2] had to be stored and referenced locally from within the master schema. The original XLink schema definition, however, introduced a name clash error which prevented xjc from completing its execution. The title attribute also appears as an element in this schema, which is not allowed by the xjc compiler. This has been circumvented by renaming all the occurences of the name of the title attribute in the XLink schema definition and then generating the classes. To restore the code to the valid XLink norm, the edited names have been restored in the annotations for the XML metadata. The XML metadata annotations define how the elements in the XML document are (de)serialised as, so editing them from `titleattr` back to `title` would have successfully reverted the required change to generate the classes. The command used to generate these classes is shown in Listing 4.2. The first argument is the master schema, the second argument specifies the output directory and the last argument specifies the package name. A selection of generated classes can be found in Appendix A.12.

```
xjc "C:\Users\david\thesis\enriched_xlink\master_schema.xsd"
 -d "C:\Users\david\thesis\enriched_xlink\output"
 -p com.xlink.gen
```

Listing 4.2: Command used with the xjc tool to generate the classes

4.3.2 IServerInterfaceImp2

In the link service, the RSL interfaces and database interactions are combined and used in the `IServerInterfaceImp2` instance. This rather large class contains all the methods which we will be using for our XLink implementation integration. As we have mentioned before, the db4o database packages will be reused for the users' management. This means that the ID's of individuals and groups all remain the same. For our enhanced XLink solution, a custom version of this class will be implemented.

The main customisation will be the disabling of the communication between the RSL objects and the db4o database, since the links will be stored in our linkbase. Because of the automatic generation of the enhanced XLink classes, it does not take much code to (de)serialise the links in the linkbase and keep the instances in the working memory of the application. During the application life-cycle, the GUI will consult/alter the objects in memory and only write (serialise) these back into the linkbase upon closing the program. During the de-serialisation, a schema

[2]http://www.w3.org/1999/xlink.xsd

definition can be added to validate the XML links while they are being processed. By passing the links through the master schema, the link service can prevent badly defined links from entering the system.

Figure 4.1 shows, in a simplified way, how the relevant structure was in the original link service. We observe a model-view-controller pattern, where the `iServerInterfaceImp2` plays the role of the controller. In our custom version of this class, we introduce a linkbase which will be used to store the links. The structure of our implementation can be seen in Figure 4.2. The RSL components have not been completely removed, so the mapping between the models can be evaluated. The mapping between these two linking models can best be evaluated by having the link service read the links from a linkbase and converting them to RSL objects. These RSL objects can then be altered by the `iServerInterfaceImp2` and converted back to XLink links at the end of the link service life cycle.

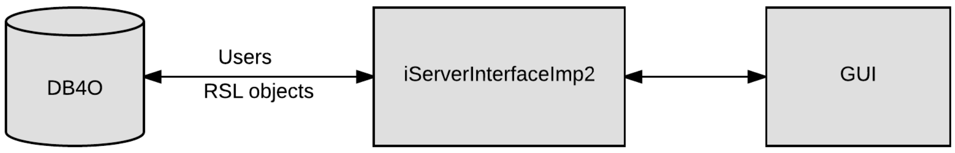


Figure 4.1: Structure of the original application

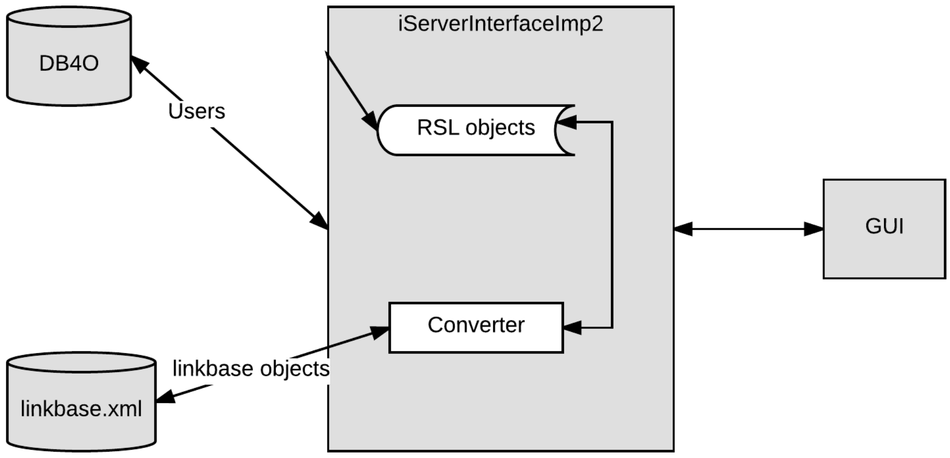


Figure 4.2: Structure of the enriched XLink implementation

4.3.3 Mapping Link Objects

The top level conversion between XLink objects and RSL objects is described in pseudo-code in Listing 4.3. By top level we mean the focus on resources, selectors and links. Lower level conversion is not that interesting, except for the different selectors, since a lot of properties are just literally copied over.

```
For all RSL link: create XLink link
    For all RSL source: add locator to XLink link
    For all RSL target: add locator to XLink link
        For all RSL source: create an arc from the locator of the source to the
    locator of the target
```

Listing 4.3: Pseudo code explaining how the conversion between RSL links and XLink links is performed

The assumption here is that each of the RSL link's sources will link to all the RSL link's targets. If the RSL source/target resource has a selector specifying a certain part of it, we must call upon the selector plugins to see if they know how to convert the selector data. Plugin converter classes that match the document type, know which attributes are associated with the document type, and which XLink selector subclass is needed to convert the selection.

4.3.4 Plugin Management

Plugins for the application must subclass `AbstractSelector`. These plugin classes could (and should) be generated from their schema definitions via xjc to be compliant to the other classes. The plugin schema definition should be comprised of just one complexType, with as name the MIME type of the type of document it wants to represent. One must bear in mind that forward slashes are not accepted and should be replaced by underscores. The content of the complexType should be an extension of the base, which is `abstractSelector`, and contain only the attributes that are required for the document type specific selector.

Once the object class has been generated, a plugin converter class also needs to be implemented. Note that this is only necessary for the proof of concept implementation where both RSL and XLink are used in the application to demonstrate the mapping functionality. The required plugin converter class should be a subclass of BaseConverter. This is an abstract class that provides two methods. One is for mapping an RSL selector to an XLink selector and one for the other way around. The `Converter` class, which implements the singleton pattern, contains a list of plugin converters and also offers two main public methods. The first takes an RSL selector object as input and will iterate the list of known converters, each one trying to convert the object. If a converter matches the MIME type of the RSL resource, it will create an XLink selector of this document type, which will be returned to the original caller. If the entire list is iterated and not a single converter matched the given MIME type, an exception will be thrown as the application will have no way of knowing which selector metadata needs to be extracted and information would get lost. The second method does the opposite: it accepts an AbstractSelector representing a selector object for XLink links and converts it via one of the known converters into an RSL selector. This process is shown in Figure 4.3 and the code can be found in Appendix A.7.

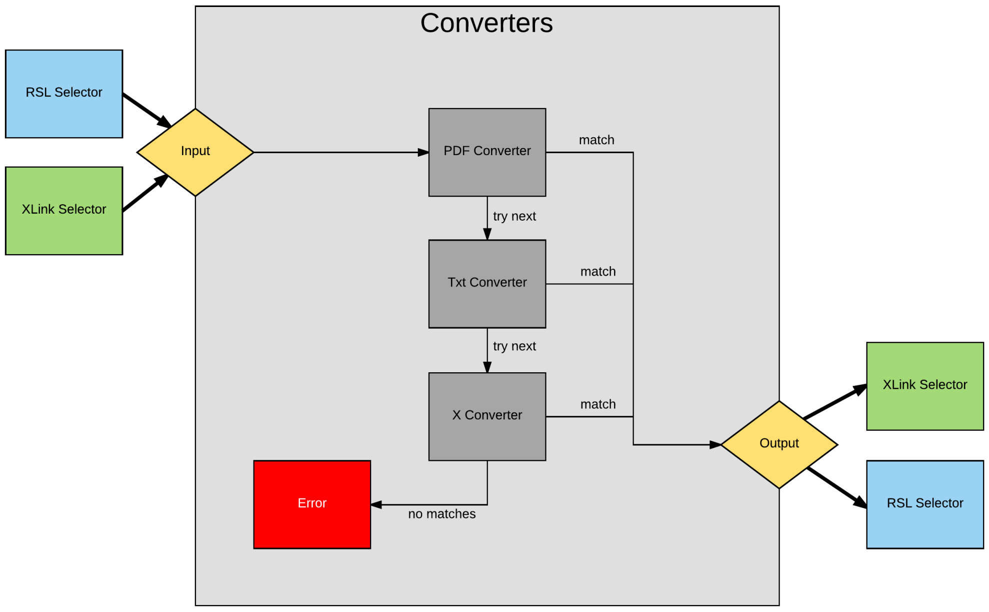


Figure 4.3: Converters data flow

4.4 Summary

In this chapter we have presented an XML schema definition that we named the master schema. It defines exactly how our enhanced XLink standard should be used (i.e. which elements and attributes are optional/required). The schema uses an abstract type for the selectors, so plugins can extend it and add the attributes required to make a selection of their document type. We have then used the master schema with the JAXB schema compiler to generate Java classes for our enhanced XLink solution. Since these generated classes are annotated with JAXB metadata, they can be easily serialised into linkbases. These classes have been introduced to the link service, where we have located the most controlling class and implemented an edited version. The edited version now uses linkbases to store the links when the application is shut down. During start-up, the link service will de-serialise our linkbase and convert the enhanced XLink links to RSL objects via various plugin converters. When the application is stopped, the RSL links in memory will be converted back and the XLink links are serialised to the linkbase, demonstrating the mapping between the two linking models.

5 Conclusions and Future Work

Cross-document linking on the Web is still not a common practice. The introduction of the XLink standard by the W3C was a big step in the right direction, but it still lacks some important features that can be found in other linking models. One of these unsupported features is cross-document linking in *all* document types, since XLink is limited to linking in XML-based documents only. Another one is the absence of user rights management (i.e. being able to claim ownership or to decide who can and cannot view your links). Factors like these have undoubtedly led to the emergence of new linking models, such as the Resource Selector Link (RSL) metamodel. The RSL metamodel does possess multiple sophisticated linking features, but it has the drawback of not being a standard. Standards on the Web are important guidelines for maintaining stability and compatibility of different technologies. Therefore, we investigated how the XLink standard can be enhanced with additional linking features and how it can be made to support linking in existing as well as emerging document formats.

In our research, we have explored several approaches to the presented problem. The first approach consisted of embedding the XLink links in the documents themselves. The advantage there was that the source of the link was the document itself. However, this did not allow the XLink standard to practise cross-document linking in non-XML-based document types, which is the first feature we would prefer to support. Since the main problem of XLink is that it can only address parts of XML documents, we reason that the unsupported target documents need to be made accessible. This resulted in a second approach, where we associate XML documents with each linked document. These *virtual documents*, as we call them, store information about their associated document such as the filename, document type, location and selectors. Since the virtual documents know the document type it is associated with, correct selector data can be stored in them. Via this method, the XLink standard can link to parts of every document type via linking to a selector in the virtual document. Other linking features, inspired by the RSL metamodel, could enrich the XLink standard by adding their attributes to the XLink elements. The XLink standard allows any elements and attributes to be added to link elements, as long as they do not clash with the attributes of the XLink standard. For example, to associate ownership of links, a `createdBy` attribute could be added to the link. While this approach looked quite promising, we predict that overhead will be caused by the virtual documents. By this overhead we refer to the reality that managing these virtual documents would not scale well to a larger environment. In order to move on from managing all these virtual documents, the selector data would have to be stored in the links as well. This leads us to the final solution: embedding all the data of these linking feature requests via attributes in the links.

We then deduced which attributes are required and which are optional for creating these enhanced XLink links. For example, attributes such as `createdBy` and `accessibleTo` enforce user rights and by also adding the document type specific selector attributes, cross-document linking remains possible for all the formats. This sounds good, but with so many new variables in play, the enhanced links need to be validated by means of XML schema definitions. We then created said ‘master’ schema to validate the enriched XLink links. By means of plugin schemas, the master schema definition is able to validate links with selector data of different document types. This solution has then been implemented in a link service application that initially used the RSL metamodel. The implementation has been done via serialisable XLink objects, which makes it an effortless task to read and write links from and to the link database (linkbase). To demonstrate the mapping between our enhanced XLink model and the RSL metamodel, we keep using RSL elements in the working memory of the application, but convert them to XLink links for storage in a linkbase. The conversion of XLink and RSL links can also be easily extended via converter plugins to support new document types. A limitation of this solution is that all links are read into the memory all at once when the application starts. For a proof of concept this suffices, but for a larger application this should be further optimised.

Potential future work for the presented solution can exist of extending the master schema with even more linking features. These features could be inspired from other linking documents or perhaps a survey could present us with desired features that do not exist yet. Furthermore, in the current master schema, selector plugins are directly injected via XSLT transformations. It would be better if the master schema imported just one schema that collected all the plugins. Because of the mapping demonstration, the RSL core is still lingering in the link service. For future work, one could potentially remove the RSL model completely so that the enhanced XLink standard is the only linking model in the application.

A
Appendix

Master Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <xsd:import namespace="http://www.w3.org/1999/xlink"
    schemaLocation="xlink.xsd" />

  <xsd:include schemaLocation="http://wilma.vub.ac.be/~dsverdlo/xlink/master_txt.
    xsd" />
  <xsd:include schemaLocation="http://wilma.vub.ac.be/~dsverdlo/xlink/master_pdf.
    xsd" />

  <xsd:element name="linkbase">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="link" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
      <xsd:sequence>

   <xsd:element name="resource" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
    <xsd:attribute type="xsd:string" name="accessibleTo" use="optional"/>
    <xsd:attribute type="xsd:string" name="inaccessibleTo" use="optional"/>
    <xsd:attribute type="xsd:string" name="createdBy" use="optional"/>
    <xsd:attribute ref="xlink:type"  fixed="resource" />
    <xsd:attribute ref="xlink:label" use="optional" />
    <xsd:attribute ref="xlink:titleattr" use="optional" />
    <xsd:attribute ref="xlink:role"  use="optional" />
              </xsd:complexType>
```

```
            </xsd:element>

            <xsd:element name="locator" minOccurs="0" maxOccurs="unbounded" type=
    "abstractSelector" />

            <xsd:element name="arc" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
    <xsd:attribute ref="xlink:type"  fixed="arc" />
    <xsd:attribute type="xsd:string" name="accessibleTo" use="optional"/>
    <xsd:attribute type="xsd:string" name="inaccessibleTo" use="optional"/>
    <xsd:attribute type="xsd:string" name="createdBy" use="optional"/>
    <xsd:attribute ref="xlink:arcrole" fixed="http://www.w3.org/1999/xlink/
    properties/linkbase" />
    <xsd:attribute ref="xlink:titleattr" use="optional" />
    <xsd:attribute ref="xlink:from"  use="optional" />
    <xsd:attribute ref="xlink:to"  use="optional" />
    <xsd:attribute ref="xlink:show"  use="optional" />
    <xsd:attribute ref="xlink:actuate" use="optional" />

              </xsd:complexType>
            </xsd:element>

   </xsd:sequence>

            <xsd:attribute ref="xlink:type" fixed="extended" />
   <xsd:attribute type="xsd:string" name="accessibleTo" use="optional"/>
   <xsd:attribute type="xsd:string" name="inaccessibleTo" use="optional"/>
   <xsd:attribute type="xsd:string" name="createdBy" use="optional"/>
            <xsd:attribute ref="xlink:role" use="optional" />
            <xsd:attribute ref="xlink:titleattr" use="optional" />
   <xsd:attribute ref="xlink:show"  use="optional" />
   <xsd:attribute ref="xlink:actuate" use="optional" />
   <xsd:attribute type="xsd:string" name="id"/>

          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="abstractSelector" >
 <xsd:attribute type="xsd:string" name="accessibleTo" use="optional"/>
 <xsd:attribute type="xsd:string" name="inaccessibleTo" use="optional"/>
 <xsd:attribute type="xsd:string" name="createdBy" use="optional"/>
 <xsd:attribute type="xsd:integer" name="layer" use="optional" />
 <xsd:attribute ref="xlink:type"  fixed="locator" />
 <xsd:attribute ref="xlink:href"  id="href" />
 <xsd:attribute ref="xlink:label" use="optional" />
 <xsd:attribute ref="xlink:titleattr" use="optional" />
 <xsd:attribute ref="xlink:role"  use="optional" />
  </xsd:complexType>

</xsd:schema>
```

Listing A.1: Master schema

Text Plugin for the Master Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:complexType name="text_plain">
  <xs:complexContent>
   <xs:extension base="abstractSelector">
     <xs:attribute name="selectFrom" type="xs:integer" />
     <xs:attribute name="selectTo" type="xs:integer" />

   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

</xs:schema>
```

Listing A.2: Schema plugin for text files

PDF Plugin for the Master Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="application_pdf">
 <xs:complexContent>
   <xs:extension base="abstractSelector">
    <xs:attribute name="selectX" type="xs:double" />
    <xs:attribute name="selectY" type="xs:double" />
    <xs:attribute name="selectW" type="xs:double" />
    <xs:attribute name="selectH" type="xs:double" />
    <xs:attribute name="selectPage" type="xs:integer" />
   </xs:extension>
 </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

Listing A.3: Schema plugin for PDF files

XSLT Adding Plugins to the Master Schema

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

<!-- This XSLT document allows the insertion of a new file format selector plugin
    .
 The include tag is to be inserted below the import of the XLink schema
    definition.
 David Sverdlov, 2015 -->
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Copy everything in the schema -->
  <xsl:template match="xsd:schema">
     <xsl:copy>
      <xsl:apply-templates />
     </xsl:copy>
   </xsl:template>

  <!-- When copying import, add the plugin below it -->
<xsl:template match="xsd:import">
  <xsl:copy-of select="." />
   ###LOCATION###
</xsl:template>

  <!-- Copy instructions (recursive) -->
  <xsl:template match="@*|node()">
     <xsl:copy>
       <xsl:apply-templates select="@*|node()" />
     </xsl:copy>
   </xsl:template>

</xsl:stylesheet>
```

Listing A.4: XSLT script to add a plugin to the master schema

XSLT Removing Plugins from the Master Schema

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


<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Copy everything in the schema -->
  <xsl:template match="xsd:schema">
     <xsl:copy>
      <xsl:apply-templates />
     </xsl:copy>
   </xsl:template>

  <!-- When copying include, add the plugin below it -->
<xsl:template match="xsd:include[@schemaLocation='###LOCATION###']">
```

```
</xsl:template>

  <!-- Copy instructions (recursive) -->
  <xsl:template match="@*|node()">
     <xsl:copy>
       <xsl:apply-templates select="@*|node()" />
     </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Listing A.5: XSLT script to remove a plugin from the master schema

Linkbase Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<linkbase xmlns:xlink="http://www.w3.org/1999/xlink">
    <link xlink:type="extended"
   createdBy="ahmed"
   id="wsbcmahyfiwqffjchlnw">
        <locator xsi:type="text_plain"
    selectFrom="296"
    selectTo="305"
    xlink:type="locator"
    xlink:href="C:\Users\david\LinkedDocuments\epub.txt"
    xlink:label="qqvepuychsecbcehxsin"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        <locator xsi:type="text_plain"
    selectFrom="312"
    selectTo="319"
    xlink:type="locator"
    xlink:href="C:\Users\david\LinkedDocuments\epub.txt"
    xlink:label="bycihsrufpvpwazbuhah"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        <arc xlink:type="arc"
    xlink:from="qqvepuychsecbcehxsin"
    xlink:to="bycihsrufpvpwazbuhah"/>
    </link>
</linkbase>
```

Listing A.6: Linkbase generated by the link service

Converter Classes Package

```
package com.xlink.gen2;

import java.util.ArrayList;

import com.rsl.core.Entity;
import com.rsl.core.Selector;
import com.xlink.gen.AbstractSelector;

public class Converter {
 private Converter _instance;
 private ArrayList<BaseConverter> _converters;

 public Converter() {
  _converters = new ArrayList<BaseConverter>();
 }

 public AbstractSelector convertToXLink(Entity rslEntity) throws Exception {
  for(BaseConverter conv: _converters) {
   AbstractSelector converted = conv.convertToXLink(rslEntity);
   if (converted != null) {
    return converted;
   }
  }
  throw new Exception("RSL selector type " + rslEntity.getClass() + " not
    convertable.");
 }

 public Entity convertToRSLLink(AbstractSelector xLinkLocator) throws Exception {
  for(BaseConverter conv: _converters) {
   Entity converted = conv.convertToRSL(xLinkLocator);
   if (converted != null) {
    return converted;
   }
  }
  throw new Exception("XLink selector type " + xLinkLocator.getClass() + " not
    convertable.");
 }

 public void add_converter(BaseConverter conv) {
  _converters.add(conv);
 }

 public Converter get_instance() {
  if (_instance == null) {
   _instance = new Converter();
  }
  return _instance;
 }


}
```

Listing A.7: Converter singleton

```
package com.xlink.gen2;

import com.rsl.core.Entity;
import com.rsl.core.Selector;
import com.xlink.gen.AbstractSelector;


public abstract class BaseConverter {

 public AbstractSelector convertToXLink(AbstractSelector xLinkSelector, Selector
    rslSelector) {
  System.out.println("BaseConverter RSL->XLink");
  return xLinkSelector;
 }

 public Entity convertToRSL(Selector rslSelector, AbstractSelector xLinkSelector)
     {
  rslSelector.setName(xLinkSelector.getLabel());
  return rslSelector;
 }

 public abstract AbstractSelector convertToXLink(Entity entity);
 public abstract Entity convertToRSL(AbstractSelector selector);
}
```

Listing A.8: BaseConverter abstract class

```
package com.xlink.gen2;

import java.awt.geom.Rectangle2D;
import java.math.BigInteger;

import org.rsl.pdf.data.*;

import com.rsl.core.Entity;
import com.xlink.gen.AbstractSelector;
import com.xlink.gen.ApplicationPdf;
import com.xlink.gen.TypeType;

public class PdfConverter extends BaseConverter {

 @Override
 public AbstractSelector convertToXLink(Entity entity) {
  try {
   System.out.println("PdfConverter RSL->XLink");
   // Cast to corresponding plugin
   PDFSelector rslSelector = (PDFSelector) entity;

   // Create object to return
   ApplicationPdf xLinkSelector = new ApplicationPdf();

   // Specific fields
   int pageIndex = rslSelector.getPageIndex();
   xLinkSelector.setSelectPage(BigInteger.valueOf(pageIndex));
   Rectangle2D rect = rslSelector.getRec();
   xLinkSelector.setSelectX(rect.getX());
   xLinkSelector.setSelectY(rect.getY());
   xLinkSelector.setSelectW(rect.getWidth());
   xLinkSelector.setSelectH(rect.getHeight());
   xLinkSelector.setType(TypeType.LOCATOR);
```

```
      PDFResource textResource = (PDFResource) rslSelector.getResource();
      xLinkSelector.setHref(textResource.getPath());

      // Let super have his way
      convertToXLink(xLinkSelector, rslSelector);

      return xLinkSelector;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }

  @Override
  public Entity convertToRSL(AbstractSelector xLinkSelector) {
    try {
      // Try to cast
      ApplicationPdf xLinkPdfSelector = (ApplicationPdf) xLinkSelector;

      // Create RSL entity
      Double x = xLinkPdfSelector.getSelectX();
      Double y = xLinkPdfSelector.getSelectY();
      Double width = xLinkPdfSelector.getSelectW();
      Double height = xLinkPdfSelector.getSelectH();
      BigInteger pageIndex = xLinkPdfSelector.getSelectPage();
      Rectangle2D rect = new Rectangle2D.Double(x,y,width,height);
      PDFSelector selector = new PDFSelector(rect, pageIndex.intValue());
      PDFResource resource = new PDFResource(xLinkPdfSelector.getHref());
      selector.setResource(resource);
      // let super have its way
      convertToRSL(selector, xLinkPdfSelector);

      return selector;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }


}
```

Listing A.9: Converter for the PDF document type

```
package com.xlink.gen2;

import java.math.BigInteger;

import org.rsl.text.data.TextResource;
import org.rsl.text.data.TextSelector;

import com.rsl.core.Entity;
import com.xlink.gen.AbstractSelector;
import com.xlink.gen.TextPlain;
import com.xlink.gen.TypeType;

public class TextConverter extends BaseConverter {

  @Override
  public AbstractSelector convertToXLink(Entity entity) {
    try {
      System.out.println("TextConverter RSL->XLink");
      // Cast to corresponding plugin
      TextSelector rslTextSelector = (TextSelector) entity;

      // Create object to return
      TextPlain xLinkSelector = new TextPlain();

      // Specific fields
      int from = rslTextSelector.getStart();
      int to = rslTextSelector.getEnd();
      xLinkSelector.setSelectFrom(BigInteger.valueOf(from));
      xLinkSelector.setSelectTo(BigInteger.valueOf(to));
      xLinkSelector.setType(TypeType.LOCATOR);
      TextResource textResource = (TextResource) rslTextSelector.getResource();
      xLinkSelector.setHref(textResource.getPath());

      // Let super have his way
      convertToXLink(xLinkSelector, rslTextSelector);

      return xLinkSelector;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }

  @Override
  public Entity convertToRSL(AbstractSelector xLinkSelector) {
    try {
      // Try to cast
      TextPlain xLinkTextSelector = (TextPlain) xLinkSelector;

      // Create RSL entity
      BigInteger from = xLinkTextSelector.getSelectFrom();
      BigInteger to = xLinkTextSelector.getSelectTo();
      TextSelector selector = new TextSelector(from.intValue(), to.intValue());
      TextResource resource = new TextResource(xLinkTextSelector.getHref());
      selector.setResource(resource);

      // Let super have its way
      convertToRSL(selector, xLinkTextSelector);

      return selector;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }


}
```

Listing A.10: Converter for the text document type

Custom IServerInterfaceImp2

```
package com.xlink.gen;

...

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

import com.rsl.core.*;
import com.rsl.databasemanager.*;
import com.rsl.databasemanager.DatabaseManager.DbSource;
import com.xlink.gen.*;
import com.xlink.gen.Linkbase.Link.Arc;
import com.xlink.gen2.Configs;
import com.xlink.gen2.Converter;
import com.xlink.gen2.PdfConverter;
import com.xlink.gen2.TextConverter;

import com.rsl.plugin.tracker.*;


/**
 * This class provides implementation of RSL interfaces + API for (CRUD)
    operations with Db4o
 *
 * Should be enhanced to be a singleton
 *
 * @author ahmed tayeh ahmed.tayeh@vub.ac.be
 *
 * Edited by David Sverdlov to support the enhanced XLink standard
 * !! Only included the changes in this version !!
 *
 */
public class IServerInterfaceImp2 implements RSInterface{



 private HashMap<String, Entity> map_entities = null;
 private HashMap<String, Resource> map_resources = null;
 private HashMap<String, Selector> map_selectors = null;
 private HashMap<String, Link> map_links = null;
 private HashMap<String, Layer> map_layers = null;

 private Converter converter = null;

 /**
  * Constructs a new IServerInterfaceImp instance.
  */
  public IServerInterfaceImp2(HashSet bundles){
   if(getDatabaseManager() == null){
    setDatabaseManager(DatabaseManager.getDatabaseManager(DbSource.Db4o, null,
    bundles));

    System.out.println("Nov 6 —— Initialising the DB");

    System.out.println("Apr 17 —— Implementing with XLink");

    map_entities = new HashMap<String, Entity>();
    map_resources = new HashMap<String, Resource>();
    map_selectors = new HashMap<String, Selector>();
    map_links = new HashMap<String, Link>();
    map_layers = new HashMap<String, Layer>();

    // Grab the config file
    try {
     // Deserialize linkbase
     JAXBContext context = JAXBContext.newInstance("com.xlink.gen2");
     Unmarshaller m2 = context.createUnmarshaller();
     byte[] encoded = Files.readAllBytes(Paths.get("c:\\configfile.xml"));
     Configs cfg = (Configs) m2.unmarshal(new StringReader(new String(encoded,
    StandardCharsets.UTF_8)));
     // load jar
     // add converters
     System.out.println("Static config file read!");
     converter = new Converter();
     converter.add_converter(new TextConverter());
     converter.add_converter(new PdfConverter());
    } catch (Exception e) { e.printStackTrace(); }


    // Grab the links, convert them and put RSL objects in memory
    ObjectFactory factory = new ObjectFactory();
    try {
     // Deserialize linkbase
     JAXBContext context = JAXBContext.newInstance("com.xlink.gen");
     Unmarshaller m2 = context.createUnmarshaller();
     byte[] encoded = Files.readAllBytes(Paths.get("c:\\mylinkbase.xml"));
     Linkbase base = (Linkbase) m2.unmarshal(new StringReader(new String(encoded,
     StandardCharsets.UTF_8)));

     // for each XLink link (l): create RSL link object
     for(Linkbase.Link l: base.getLink()) {
      System.out.println("Found a link in the linkbase...");
      Link rslLink = new Link();
      setLinkAssociations(rslLink);
      rslLink.setName(l.getId());
      if(l.getCreatedBy() != null) {
       Individual ind = getIndividual(l.getCreatedBy());
       if (ind != null) {
        rslLink.setCreator(ind);
       }
      }

      // add locators as resources
      for(AbstractSelector sel: l.getLocator()){
       // Also convert in converter

       Selector sourceSel = (Selector) converter.convertToRSLLink(sel);

       sourceSel.setSelectorAssociations((BidirectionalAssociation)db.read(
    BidirectionalAssociation.class, "type == \"" + "krefersTo"+"\"").getFirst(),
         (BidirectionalAssociation)db.read(BidirectionalAssociation.class, "type
     == \"" + "onLayer"+"\"").getFirst());
       System.out.println("Sel" + sourceSel);

       Resource r = map_resources.get(sel.getLabel());
       if (r == null) {
        if(sourceSel.getResource() == null) {
         r = new Resource();
        } else {
         r = sourceSel.getResource();
```

```
          }
          setResourceAssociations(r);
          r.setName(sel.getLabel());
          map_resources.put(sel.getLabel(), r);
        }
        r.setUri(sel.getHref());
        // create selector and add to resource
        System.out.println("Resource: " + r);
        sourceSel.setResource(r);
        r.addSelector(sourceSel);
        if (sel.getLayer() != null) {
          Layer sourceLayer = map_layers.get(sel.getLayer().toString());
          if(sourceLayer == null) {
            sourceLayer = new Layer();
            map_layers.put(sel.getLayer().toString(), sourceLayer);
          }
          sourceLayer.setPosition(sel.getLayer().intValue());
          sourceSel.setLayer(sourceLayer);
        }

        map_selectors.put(r.getName()+"s", sourceSel);

      } // end locator loop

      for(Arc arc: l.getArc()){
        // For each arc, add 'from' to sources and 'to' to targs
        String from = arc.getFrom();
        String to = arc.getTo();

        Resource source = map_resources.get(from);
        if (source == null) {
          source = new Resource();
          setResourceAssociations(source);
          source.setName(from);
          map_resources.put(from, source);
        }
        Resource target = map_resources.get(to);
        if (target == null) {
          target = new Resource();
          setResourceAssociations(target);
          target.setName(to);
          map_resources.put(to, target);
        }

        // Add sources and targets to rsl link
        rslLink.addSource(source);
        rslLink.addTarget(target);

        map_links.put(rslLink.getId(), rslLink);

      }
    }


  } catch(Exception e) { e.printStackTrace(); }
 }
} // IServerImp
```

```
//*******************************
   /// ///////////////////////// DONE WITH GUI
//*******************************



 private static IDatabaseConnector db = null;


 //Handle DataBase
 /**
     * Used by subclasses to set the iServer databaseManager.
     * @param db the DatabaseManager to be set.
     */
    protected synchronized void setDatabaseManager (DatabaseManager database){
     db = database;
         //db.resetDb();
     AbstractRslElement.setCtr(db.getInitId());
     initAssociations();
    } // setDatabaseManager


    /**
     * Checks whether the associations are in the database if not it creates the
    associations
     * with the specified cardinalities as mentioned in the model.
     */
    private void initAssociations(){...
    } // initAssociations


    /**
     * Resets the iServer database.
     */
 @Override
 public synchronized void resetDatabase() {
  // TODO Auto-generated method stub
     db.resetDb();
     AbstractRslElement.setCtr(0);
 }

      /**
     * Initialize and starts the auto-commit function of the iServer.
     * @param amount the specified amount of interactions between two commits.
     * @param timeMs the specified time in milliseconds between two commits.
     */
 @Override
 public synchronized void autoCommit(int amount, int timeMs) {
  // TODO Auto-generated method stub
    db.startAutoCommit(amount, timeMs);

 }

 public synchronized void closeDB(){
  System.out.println("Writing to linkbase");
  try {
   // Serialize linkbase (write memory to linkbase)
   JAXBContext context = JAXBContext.newInstance("com.xlink.gen");
   StringWriter writer = new StringWriter();
   Marshaller m = context.createMarshaller();
   m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE );
```

```
m.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");

// Build linkbase
Linkbase base = new Linkbase();

// for each link in memory... create XLink
for(Link rslLink: map_links.values()) {
 System.out.println("Found a link in memory.. writing to linkbase");
 Linkbase.Link xlink = new Linkbase.Link();
 xlink.setType(TypeType.EXTENDED);
 xlink.setId(rslLink.getName());
 if (rslLink.getCreator() != null) {
  Individual ind = rslLink.getCreator();
  if(ind != null) {
   xlink.setCreatedBy(ind.getLogin());
  }
 }


 // For each RSL source, add them as locators
 for(Entity resource_entity: rslLink.getSources()) {
  System.out.println("RSL link has source: " + resource_entity.toString());

  if ( resource_entity.getClass() == Resource.class) {
   Resource resource = (Resource) resource_entity;
   System.out.println("RSL link has REsource uri: " + resource.getUri());
   if(resource.getSelectors() == null){
    AbstractSelector source = new AbstractSelector();
    source.setHref(resource.getUri());
    source.setLabel(resource.getName());
    xlink.getLocator().add(source);
   } else {
    for(Selector resource_selector: resource.getSelectors()) {
     System.out.println("source (resource) has selector: " + resource_selector
 .toString());
     AbstractSelector source = converter.convertToXLink(resource_selector);
     source.setLabel(resource_selector.getResource().getName());
     xlink.getLocator().add(source);
    }
   }
  } else {
   AbstractSelector source = converter.convertToXLink(resource_entity);
   Selector resource_selector = (Selector) resource_entity;
   Resource resource_selector_resource = resource_selector.getResource();
   if (source.getHref() == null) {
    System.out.println("Nohref found");
    Resource r = resource_selector.getResource();
   }
   source.setLabel(resource_selector.getName());
   xlink.getLocator().add(source);
  }

 }

 // For each RSL target, create an arc from every source
 for(Entity resource_entity: rslLink.getTargets()) {
  System.out.println("RSL link has target: " + resource_entity.toString());

  if ( resource_entity.getClass() == Resource.class) {
   Resource resource = (Resource) resource_entity;
   if(resource.getSelectors() == null){
    AbstractSelector target = new AbstractSelector();
```

```
        target.setHref(resource.getUri());
        target.setLabel(resource.getName());
        xlink.getLocator().add(target);
      } else {
        for(Selector resource_selector: resource.getSelectors()) {
          AbstractSelector target = converter.convertToXLink(resource_selector);
          target.setLabel(resource_selector.getResource().getName());
          xlink.getLocator().add(target);
        }
      }
    } else {
      AbstractSelector target = converter.convertToXLink(resource_entity);
      target.setLabel(resource_entity.getName());
      xlink.getLocator().add(target);
    }

    for(Entity arc_source: rslLink.getSources()) {
      Arc arc = new Arc();
      arc.setType(TypeType.ARC);
      arc.setFrom(arc_source.getName());
      arc.setTo(resource_entity.getName());
      xlink.getArc().add(arc);
    }
   }

   // add link to base
   base.getLink().add(xlink);
  }

  m.marshal(base, writer);
  String buffer = writer.getBuffer().toString();
  System.out.println(buffer);
  PrintWriter out = new PrintWriter("c:\\mylinkbase.xml");
  out.print(buffer);
  out.close();
 } catch (Exception e) { e.printStackTrace(); }

 db.closeDb();
}

    /**
    * Calls the commit function of the current database.
    */


//Handle Resources


/**
  * Creates a new resource instance.
  * @param name the name of the resource instance.
  * @param creator the individual who creates the resource instance.
  * @return the new resource instance.
  * @throws CardinalityConstraintException
  */

@Override
public Resource createResource(String name, Resource resource, Individual
   creator) throws CardinalityConstraintException {
 // TODO Auto-generated method stub
 /* selector.setSelectorAssociations((BidirectionalAssociation)db.read(
    BidirectionalAssociation.class, "type == \"" + "refersTo"+"\"").getFirst(),
     (BidirectionalAssociation)db.read(BidirectionalAssociation.class, "type ==
    \"" + "onLayer"+"\"").getFirst());
 */
  setResourceAssociations(resource);
      resource.setName(name);
      resource.setLabel(name);
      resource.setCreator(creator);
  // db.create(resource);
      map_resources.put(name, resource);
  return resource;
}


/**
  * Returns the resource with the specified name.
  * @param name the name of the resource which has to be returned.
  * @return the resource with the specified name.
  */

@Override
public Resource getResource(String name) {
 // TODO Auto-generated method stub
 // return (Resource) db.read(Resource.class, "name == \""+name+"\"").getFirst()
    ;
 return map_resources.get(name);
}


/**
  * Returns the resource with the specified id.
  * @param id the id of the resource which has to be returned.
  * @return the resource with the specified id.
  */

@Override
public Resource getResource(long id) {
 // TODO Auto-generated method stub
 //return (Resource) db.read(Resource.class, "id == \""+id+"\"").getFirst();
 return map_resources.get(""+id);
}

/**
  * Updates the specified resource instance.
  * @param group the specified resource to be updated.
  */

@Override
public void updateResource(Resource resource) {
 // TODO Auto-generated method stub
 //db.update(resource);
 map_resources.put(resource.getName(), resource);

}


/**
  * Creates a new selector instance.
  * @param name the name of the selector instance.
  * @param layer the layer on which the selector is defined.
  * @param resource the resource to which the selector refers to.
```

```
   * @param creator the individual who creates the selector instance.
   * @return the new selector instance.
   * @throws CardinalityConstraintException
   */

@Override
public Selector createSelector(String name, Selector selector, Layer layer,
    Resource resource,
  Individual creator) throws CardinalityConstraintException {
 // TODO Auto-generated method stub

  setEntityAssociations(selector);
  selector.setSelectorAssociations((BidirectionalAssociation)db.read(
    BidirectionalAssociation.class, "type == \"" + "krefersTo"+"\"").getFirst(),
     (BidirectionalAssociation)db.read(BidirectionalAssociation.class, "type ==
    \"" + "onLayer"+"\"").getFirst());
  selector.setName(name);
  selector.setLayerAssociation(layer);
  selector.setResource(resource);
  selector.setCreator(creator);
  resource.addSelector(selector);

  map_selectors.put(name, selector);
  // db.create(selector);
  return selector;
}


/**
   * Returns the selector with the specified name.
   * @param name the name of the selector that has to be returned.
   * @return the selector with the specified name.
   */

@Override
public Selector getSelector(String name) {
 // TODO Auto-generated method stub
 // return (Selector)db.read(Selector.class, "name == \""+name+"\"").getFirst();
 return map_selectors.get(name);
}

//get TextSelector


/**
   * Returns the selector with the specified id.
   * @param id the id of the selector that has to be returned.
   * @return the selector with the specified id.
   */

@Override
public Selector getSelector(long id) {
 // TODO Auto-generated method stub
 //return (Selector)db.read(Selector.class, "id == \""+id+"\"").getFirst();
 return map_selectors.get(""+id);
}


 /**
  * Creates a new layer instance.
  * Note that the layer instance is set on active by default.
  * @param name the name of the layer instance.
```

```
   * @return the new layer instance.
   */

 @Override
 public Layer createLayer(String name) {
  // TODO Auto-generated method stub
  Layer layer = new Layer();
   layer.setName(name);
   layer.setActive(true);
   //db.create(layer); //  Will always make a new layer now
   map_layers.put(name, layer);

   return layer;
 }


 /**
   * Creates a new layer instance.
   * Note that the layer instance is set on active by default.
   * @param name the name of the layer instance.
   * @param position the position of the layer instance.
   * @return the new layer instance.
   */

 @Override
 public Layer createLayer(String name, int position) {
  // TODO Auto-generated method stub
   Layer layer = new Layer();
   layer.setName(name);
   layer.setPosition(position);
   layer.setActive(true);
   //db.create(layer);
   map_layers.put(name, layer);

   return layer;
 }

  /**
   * Returns the layer with the specified name.
   * @param name the name of the layer which has to be returned.
   * @return the layer with the specified name.
   */

 @Override
 public Layer getLayer(String name) {
  // TODO Auto-generated method stub
  return map_layers.get(name);
  //return (Layer) db.read(Layer.class, "name == \""+name+"\"").getFirst();
 }

 /**
   * Returns the layer with the specified id.
   * @param id the id of the layer which has to be returned.
   * @return the layer with the specified id.
   */

 @Override
 public Layer getLayer(long id) {
  // TODO Auto-generated method stub
   //return (Layer) db.read(Layer.class, "id == \""+id+"\"").getFirst();
  return map_layers.get(""+id);
 }
  /**
   * Creates a new link instance between a source and a target entity.
   * Note that this method call doesn't associate the link/target to a certain
     context.
   * @param name the name of the link instance.
   * @param source the source entity of the link instance.
   * @param target the target entity of the link instance.
   * @param creator the individual who creates the link instance.
   * @return the new link instance.
   * @throws CardinalityConstraintException
   */

 /*
 public Link createLink(String name, Entity source, Entity target,
   Individual creator) throws CardinalityConstraintException {
  // TODO Auto-generated method stub
  Link link = new Link();
   setLinkAssociations(link);
   link.setName(name);
   link.setCreator(creator);
   link.addSource(source);
   link.addTarget(target);
   db.create(link);

   return link;
 }
 */


 @Override
 public Link createLink(String name, HashSet<Entity> sources,
   HashSet<Entity> targets, Individual creator)
     throws CardinalityConstraintException{
  Link link = new Link();
  setLinkAssociations(link);
  link.setName(name);
  link.setCreator(creator);

  for(Entity e: sources){
   link.addSource(e);
  }
  for(Entity e: targets){
   link.addTarget(e);
  }
  //db.create(link);
  //db.commit();

  map_links.put(name, link);
  return link;

 }


 /**
   * Returns the link with the specified name.
   * @param name the name of the link which has to be returned.
   * @return the link with the specified name.
   */

 @Override
```

```
public Link getLink(String name) {
 // TODO Auto-generated method stub
 //return (Link) db.read(Link.class, "name == \""+name+"\"").getFirst();
 return map_links.get(name);
}
```

Listing A.11: Custom IServerInterfaceImp2 class, stripped to our changes

Generated Classes for the Enriched XLink Standard

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB)
     Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source
     schema.
// Generated on: 2016.04.22 at 04:54:44 PM CEST
//


package com.xlink.gen;

import java.math.BigInteger;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;


/**
 * <p>Java class for abstractSelector complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
     within this class.
 *
 * <pre>
 * &lt;complexType name="abstractSelector">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="accessibleTo" type="{http://www.w3.org/2001/
     XMLSchema}string" />
 *       &lt;attribute name="inaccessibleTo" type="{http://www.w3.org/2001/
     XMLSchema}string" />
 *       &lt;attribute name="createdBy" type="{http://www.w3.org/2001/XMLSchema}
     string" />
 *       &lt;attribute name="layer" type="{http://www.w3.org/2001/XMLSchema}
     integer" />
 *       &lt;attribute ref="{http://www.w3.org/1999/xlink}type fixed="locator""/>
 *       &lt;attribute ref="{http://www.w3.org/1999/xlink}href"/>
 *       &lt;attribute ref="{http://www.w3.org/1999/xlink}label"/>
 *       &lt;attribute ref="{http://www.w3.org/1999/xlink}titleattr"/>
 *       &lt;attribute ref="{http://www.w3.org/1999/xlink}role"/>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "abstractSelector", namespace = "")
@XmlSeeAlso({
    ApplicationPdf.class,
    TextPlain.class
})
public class AbstractSelector {
```

```
@XmlAttribute(name = "accessibleTo")
protected String accessibleTo;
@XmlAttribute(name = "inaccessibleTo")
protected String inaccessibleTo;
@XmlAttribute(name = "createdBy")
protected String createdBy;
@XmlAttribute(name = "layer")
protected BigInteger layer;
@XmlAttribute(name = "type", namespace = "http://www.w3.org/1999/xlink")
protected TypeType type;
@XmlAttribute(name = "href", namespace = "http://www.w3.org/1999/xlink")
protected String href;
@XmlAttribute(name = "label", namespace = "http://www.w3.org/1999/xlink")
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
protected String label;
@XmlAttribute(name = "titleattr", namespace = "http://www.w3.org/1999/xlink")
protected String titleattr;
@XmlAttribute(name = "role", namespace = "http://www.w3.org/1999/xlink")
protected String role;

/**
 * Gets the value of the accessibleTo property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getAccessibleTo() {
    return accessibleTo;
}

/**
 * Sets the value of the accessibleTo property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setAccessibleTo(String value) {
    this.accessibleTo = value;
}

/**
 * Gets the value of the inaccessibleTo property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getInaccessibleTo() {
    return inaccessibleTo;
}

/**
 * Sets the value of the inaccessibleTo property.
 *
 * @param value
 *     allowed object is
```

```
     *     {@link String }
     *
     */
    public void setInaccessibleTo(String value) {
        this.inaccessibleTo = value;
    }

    /**
     * Gets the value of the createdBy property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getCreatedBy() {
        return createdBy;
    }

    /**
     * Sets the value of the createdBy property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setCreatedBy(String value) {
        this.createdBy = value;
    }

    /**
     * Gets the value of the layer property.
     *
     * @return
     *     possible object is
     *     {@link BigInteger }
     *
     */
    public BigInteger getLayer() {
        return layer;
    }

    /**
     * Sets the value of the layer property.
     *
     * @param value
     *     allowed object is
     *     {@link BigInteger }
     *
     */
    public void setLayer(BigInteger value) {
        this.layer = value;
    }

    /**
     * Gets the value of the type property.
     *
     * @return
     *     possible object is
     *     {@link TypeType }
     *
     */
    public TypeType getType() {
        if (type == null) {
            return TypeType.LOCATOR;
        } else {
            return type;
        }
    }

    /**
     * Sets the value of the type property.
     *
     * @param value
     *     allowed object is
     *     {@link TypeType }
     *
     */
    public void setType(TypeType value) {
        this.type = value;
    }

    /**
     * Gets the value of the href property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getHref() {
        return href;
    }

    /**
     * Sets the value of the href property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setHref(String value) {
        this.href = value;
    }

    /**
     * Gets the value of the label property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getLabel() {
        return label;
    }

    /**
     * Sets the value of the label property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setLabel(String value) {
        this.label = value;
    }

    /**
     * Gets the value of the titleattr property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getTitleattr() {
        return titleattr;
    }

    /**
     * Sets the value of the titleattr property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setTitleattr(String value) {
        this.titleattr = value;
    }

    /**
     * Gets the value of the role property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getRole() {
        return role;
    }

    /**
     * Sets the value of the role property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setRole(String value) {
        this.role = value;
    }

}
```

Listing A.12: Generated AbstractSelector class

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB)
     Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source
     schema.
// Generated on: 2016.04.22 at 04:54:44 PM CEST
//


package com.xlink.gen;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;


/**
 * <p>Java class for locatorType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
     within this class.
 *
 * <pre>
 * &lt;complexType name="locatorType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;group ref="{http://www.w3.org/1999/xlink}locatorModel"/>
 *       &lt;attGroup ref="{http://www.w3.org/1999/xlink}locatorAttrs"/>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "locatorType", propOrder = {
    "title"
})
public class LocatorType {

    protected List<TitleEltType> title;
    @XmlAttribute(name = "type", namespace = "http://www.w3.org/1999/xlink",
    required = true)
    protected TypeType type;
    @XmlAttribute(name = "href", namespace = "http://www.w3.org/1999/xlink",
    required = true)
    protected String href;
    @XmlAttribute(name = "role", namespace = "http://www.w3.org/1999/xlink")
    protected String role;
    @XmlAttribute(name = "titleattr", namespace = "http://www.w3.org/1999/xlink")
    protected String titleattr;
    @XmlAttribute(name = "label", namespace = "http://www.w3.org/1999/xlink")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    protected String label;
    /**
     * Gets the value of the title property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the title property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getTitle().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link TitleEltType }
     *
     *
     */
    public List<TitleEltType> getTitle() {
        if (title == null) {
            title = new ArrayList<TitleEltType>();
        }
        return this.title;
    }

    /**
     * Gets the value of the type property.
     *
     * @return
     *     possible object is
     *     {@link TypeType }
     *
     */
    public TypeType getType() {
        if (type == null) {
            return TypeType.LOCATOR;
        } else {
            return type;
        }
    }

    /**
     * Sets the value of the type property.
     *
     * @param value
     *     allowed object is
     *     {@link TypeType }
     *
     */
    public void setType(TypeType value) {
        this.type = value;
    }

    /**
     * Gets the value of the href property.
     *
     * @return
     *         possible object is
     *         {@link String }
     *
     */
    public String getHref() {
        return href;
    }

    /**
     * Sets the value of the href property.
     *
     * @param value
     *         allowed object is
     *         {@link String }
     *
     */
    public void setHref(String value) {
        this.href = value;
    }

    /**
     * Gets the value of the role property.
     *
     * @return
     *         possible object is
     *         {@link String }
     *
     */
    public String getRole() {
        return role;
    }

    /**
     * Sets the value of the role property.
     *
     * @param value
     *         allowed object is
     *         {@link String }
     *
     */
    public void setRole(String value) {
        this.role = value;
    }

    /**
     * Gets the value of the titleattr property.
     *
     * @return
     *         possible object is
     *         {@link String }
     *
     */
    public String getTitleattr() {
        return titleattr;
    }

    /**
     * Sets the value of the titleattr property.
     *
     * @param value
     *         allowed object is
     *         {@link String }
     *
     */
    public void setTitleattr(String value) {
        this.titleattr = value;
    }

    /**
     *
     *        label is not required, but locators have no particular
     *        XLink function if they are not labeled.
     *
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getLabel() {
        return label;
    }

    /**
     * Sets the value of the label property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setLabel(String value) {
        this.label = value;
    }

}
```

Listing A.13: Generated LocatorType class

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB)
     Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source
     schema.
// Generated on: 2016.04.22 at 04:54:44 PM CEST
//


package com.xlink.gen;

import java.math.BigInteger;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for application_pdf complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
     within this class.
 *
 * <pre>
 * &lt;complexType name="application_pdf">
 *   &lt;complexContent>
 *     &lt;extension base="{}abstractSelector">
 *       &lt;attribute name="selectX" type="{http://www.w3.org/2001/XMLSchema}
    double" />
 *       &lt;attribute name="selectY" type="{http://www.w3.org/2001/XMLSchema}
    double" />
 *       &lt;attribute name="selectW" type="{http://www.w3.org/2001/XMLSchema}
    double" />
 *       &lt;attribute name="selectH" type="{http://www.w3.org/2001/XMLSchema}
    double" />
 *       &lt;attribute name="selectPage" type="{http://www.w3.org/2001/XMLSchema}
    integer" />
 *     &lt;/extension>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "application_pdf", namespace = "")
public class ApplicationPdf
    extends AbstractSelector
{

    @XmlAttribute(name = "selectX")
    protected Double selectX;
    @XmlAttribute(name = "selectY")
    protected Double selectY;
    @XmlAttribute(name = "selectW")
    protected Double selectW;
    @XmlAttribute(name = "selectH")
    protected Double selectH;
    @XmlAttribute(name = "selectPage")
    protected BigInteger selectPage;

    /**
     * Gets the value of the selectX property.
     *
     * @return
     *     possible object is
     *     {@link Double }
     *
     */
    public Double getSelectX() {
        return selectX;
    }

    /**
     * Sets the value of the selectX property.
     *
     * @param value
     *     allowed object is
     *     {@link Double }
     *
     */
    public void setSelectX(Double value) {
        this.selectX = value;
    }
    /**
     * Gets the value of the selectY property.
     *
     * @return
     *     possible object is
     *     {@link Double }
     *
     */
    public Double getSelectY() {
        return selectY;
    }

    /**
     * Sets the value of the selectY property.
     *
     * @param value
     *     allowed object is
     *     {@link Double }
     *
     */
    public void setSelectY(Double value) {
        this.selectY = value;
    }

    /**
     * Gets the value of the selectW property.
     *
     * @return
     *     possible object is
     *     {@link Double }
     *
     */
    public Double getSelectW() {
        return selectW;
    }

    /**
     * Sets the value of the selectW property.
     *
     * @param value
     *     allowed object is
     *     {@link Double }
     *
     */
    public void setSelectW(Double value) {
        this.selectW = value;
    }

    /**
     * Gets the value of the selectH property.
     *
     * @return
     *     possible object is
     *     {@link Double }
     *
     */
    public Double getSelectH() {
        return selectH;
    }

    /**
```

```
     * Sets the value of the selectH property.
     *
     * @param value
     *     allowed object is
     *     {@link Double }
     *
     */
    public void setSelectH(Double value) {
        this.selectH = value;
    }

    /**
     * Gets the value of the selectPage property.
     *
     * @return
     *     possible object is
     *     {@link BigInteger }
     *
     */
    public BigInteger getSelectPage() {
        return selectPage;
    }

    /**
     * Sets the value of the selectPage property.
     *
     * @param value
     *     allowed object is
     *     {@link BigInteger }
     *
     */
    public void setSelectPage(BigInteger value) {
        this.selectPage = value;
    }

}
```

Listing A.14: Generated PDF plugin class

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB)
     Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source
     schema.
// Generated on: 2016.04.22 at 04:54:44 PM CEST
//


package com.xlink.gen;

import java.math.BigInteger;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for text_plain complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
     within this class.
 *
 * <pre>
 * &lt;complexType name="text_plain">
 *   &lt;complexContent>
 *     &lt;extension base="{}abstractSelector">
 *       &lt;attribute name="selectFrom" type="{http://www.w3.org/2001/XMLSchema}
     integer" />
 *       &lt;attribute name="selectTo" type="{http://www.w3.org/2001/XMLSchema}
     integer" />
 *     &lt;/extension>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "text_plain", namespace = "")
public class TextPlain
    extends AbstractSelector
{

    @XmlAttribute(name = "selectFrom")
    protected BigInteger selectFrom;
    @XmlAttribute(name = "selectTo")
    protected BigInteger selectTo;

    /**
     * Gets the value of the selectFrom property.
     *
     * @return
     *     possible object is
     *     {@link BigInteger }
     *
     */
    public BigInteger getSelectFrom() {
        return selectFrom;
    }

    /**
     * Sets the value of the selectFrom property.
     *
     * @param value
     *     allowed object is
     *     {@link BigInteger }
     *
     */
    public void setSelectFrom(BigInteger value) {
        this.selectFrom = value;
    }

    /**
     * Gets the value of the selectTo property.
     *
     * @return
     *     possible object is
     *     {@link BigInteger }
     *
     */
    public BigInteger getSelectTo() {
        return selectTo;
    }

    /**
     * Sets the value of the selectTo property.
     *
     * @param value
     *     allowed object is
     *     {@link BigInteger }
     *
     */
    public void setSelectTo(BigInteger value) {
        this.selectTo = value;
    }

}
```

Listing A.15: Generated text plugin class

Bibliography

[1] K. M. Anderson, R. N. Taylor, and E. J. Whitehead Jr. Chimera: Hypermedia for Heterogeneous Software Development Environments. *ACM Transactions on Information Systems*, 18(3):211–245, 2000.

[2] T. Berners-Lee, M. Fischetti, and M. L. Foreword By-Dertouzos. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. HarperInformation, 2000.

[3] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML Query Language, 2002.

[4] V. Bush. As We May Think. *Atlantic Monthly*, 176(1):101–108, 1945.

[5] P. Ciancarini, F. Folli, D. Rossi, and F. Vitali. XLinkProxy: External Linkbases with XLink. In *Proceedings of DocEng 2002, ACM Symposium on Document Engineering*, McLean, USA, November 2002.

[6] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. [online: https://www.w3.org/TR/xpath], November 1999.

[7] S. DeRose, E. Maler, and R. Daniel Jr. XML Pointer Language (XPointer) Version 1.0. [online: https://www.w3.org/TR/WD-xptr], January 2001.

[8] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. [online: https://www.w3.org/TR/2000/PR-xlink-20001220], June 2001.

[9] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. [online: https://www.w3.org/TR/2000/PR-xlink-20001220], June 2001.

[10] S. DeRose, E. Maler, D. Orchard, and N. Walsh. XML Linking Language (XLink) Version 1.1. Technical report, W3C, May 2010. http://www.w3.org/TR/xlink11/.

[11] R. Hall, K. Pauls, S. McCulloch, and D. Savage. *OSGi in Action: Creating Modular Applications in Java*. Manning Publications, 2011.

[12] A. D. Iorio, G. Montemari, and F. Vitali. Beyond Proxies: XLink Support in the Browser. In *Proceedings of ITA 2005, International Conference on Internet Technologies and Applications*, Wrexham, UK, September 2005.

[13] M. Kay et al. XSL Transformations (XSLT) Version 2.0. *W3C Recommendation*, 23:52–71, 2007.

[14] O. Kiselyov. SXML Specification. *SIGPLAN Not.*, 37(6):52–58, June 2002.

[15] K. Lisovsky and D. Lizorkin. XSLT and XLink and their Implementation with Functional Techniques. *Russian Digital Libraries Journal*, 6(5), 2003.

[16] D. Lizorkin and K. Y. Lisovsky. Implementation of the XML Linking Language XLink by Functional Methods. *Programming and Computer Software*, 31(1):34–46, 2005.

[17] D. Martin and H. Ashman. Goate: An Infrastructure for New Web Linking. In *Proceedings of the International Workshop on Open Hypermedia Systems at Hypertext 2002 Conference*, Maryland, USA, June 2002.

[18] D. Martin and H. Ashman. Goate: XLink and Beyond. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia*, HYPERTEXT 2002, pages 142–143, New York, USA, 2002. ACM.

[19] J. McAffer, P. VanderLei, and S. Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley Professional, 2010.

[20] T. H. Nelson. Complex Information Processing: a File Structure for the Complex, the Changing and the Indeterminate. In *Proceedings of ACM 1965, 20th ACM National Conference*, Cleveland, USA, August 1965.

[21] T. H. Nelson. *Literary Machines*. Mindful Press, 1982.

[22] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelsteiin. xlinkit: A Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.

[23] T. O'reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications & strategies*, (1):17, 2007.

[24] A. Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proceedings of Hypertext 1989, 2nd ACM Conference on Hypertext and Hypermedia*, Pittsburgh, USA, November 1989.

[25] B. K. Reid. *Scribe: A Document Specification Language and Its Compiler*. Phd thesis, Carnegie-Mellon University Computer Science Department, Pittsburgh, USA, 1980.

[26] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. Books on Demand GmbH, 2008.

[27] B. Signer and M. C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In *Proceedings of ER 2007, 26th International Conference on Conceptual Modelling*, Auckland, New Zealand, November 2007.

[28] A. A.O. Tayeh and B. Signer. Open Cross-Document Linking and Browsing based on A Visual Plug-in Architecture. In *Proceedings of WISE 2014, 15th Web Information System Engineering Conference*, Thessaloniki, Greece, October 2014.

[29] A. A.O. Tayeh and B. Signer. A Dynamically Extensible Open Cross-Document Link Service. In *Proceedings of WISE 2015, 16th Web Information System Engineering Conference*, Miami, USA, November 2015.

[30] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*, volume 1. O'Reilly Media, Inc., 1999.

[31] R. Weir. OpenDocument Format: The Standard for Office Documents. *IEEE Internet Computing*, 13(2):83–87, 2009.

[32] N. Yankelovich, B. J. Haan, N. K. Meyrowitz, and S. M. Drucker. Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, 21(1):81–83, 1988.