

Dissertation

**A Dynamically Extensible Cross-Document
Link Service**

A dissertation submitted to the
Vrije Universiteit Brussel

for the degree of
Doctor of Philosophy in Sciences

presented by

Ahmed A.O. Tayeh

21st October 2016

Dedication

To

My lovely parents

Haneen, my precious wife

Mariam, my lovely daughter

My lovely sisters & brothers

Abstract

Digital documents often do not exist in isolation but are implicitly or explicitly linked to parts of other documents. The hyperlink concept, which was instrumental in the success of the World Wide Web, is considered the basis for creating and managing relations between documents. Using hyperlinks, most recent digital document formats as well as existing link systems enable users to associate information within as well as across different documents. However, due to a lack of empirical studies that investigate the end-user needs and behaviour when associating information within and across documents, the development of most existing document linking approaches is not driven by end-user requirements. Furthermore, existing document linking solutions often show some shortcomings in terms of the offered link granularity and cannot easily be extended to support new document formats. Most existing document formats only support hyperlinks to web resources but do not provide support for linking to parts of arbitrary third-party documents. In addition, the majority of current link systems enable the linking to a predefined set of document formats but it is not evident how the architecture of these systems can be extended to support new document formats.

In this dissertation we address the lack of user-driven and extensible cross-document linking solutions. Our approach consists of two major efforts including a user study and an architecture in combination with an extensible link service prototype. The user study relies on a multi-case design approach consisting of an online survey and interviews with participants of the online survey in order to investigate users' current behaviour in associating information as well as their appreciation and criticism of existing solutions. The insights from our user study enabled us to formulate a number of design implications for a dynamically extensible cross-document linking solution.

The presented cross-document link service, which is based on the RSL hypermedia metamodel, meets end-user requirements and enables the linking of arbitrary documents at different levels of granularity. In our dynamically extensible cross-document link service, emerging document formats are supported via new data and visual plug-ins for the presented link browser or by integrating third-party document viewers via gateways. Our cross-document link service supports the dynamic extensibility and configuration of supported document formats without the need to redeploy the core link service. The presented link service currently supports the linking of six different document formats, with three of them being integrated with their third-party document viewers.

The extensibility of the presented link service has been verified in two different evaluations. An end-user study has further been conducted in order to evaluate the usability of the proposed cross-document link service. We feel confident that the presented concepts for a dynamically extensible cross-document link service improve the maintainability of documents in so-called cross-media information spaces and enable the future-proof linking across different document formats.

Samenvatting

Digitale documenten staan vaak niet op zichzelf maar worden vaak impliciet of expliciet gelinkt aan delen van andere documenten. Het concept van de hyperlink, een doorslaggevende factor in het succes van het World Wide Web, wordt beschouwd als de basis voor het creëren en beheren van relaties tussen documenten. Door gebruik te maken van hyperlinks laten recente digitale bestandsformaten alsook bestaande link systemen gebruikers toe om informatie te associeren, zowel binnenin als tussen verschillende documenten. Door een gebrek aan empirisch onderzoek dat de noden en het gedrag van de eindgebruikers bestudeert bij het associeren van informatie is het echter zo dat bestaande technieken voor het linken van documenten niet voortgekomen zijn uit de noden en eisen van de eindgebruiker. Daarbovenop schieten bestaande link oplossingen vaak te kort in termen van de aangeboden link granulariteit en ze kunnen ook niet uitgebreid worden om nieuwe bestandsformaten te ondersteunen. De meeste bestandsformaten ondersteunen enkel links naar bronnen op het web en niet naar stukken uit arbitraire documenten door derden. Daarbovenop ondersteunen de meerderheid van de bestaande link systemen enkel links voor een voorgedefinieerde groep van bestandsformaten en het is niet evident om deze systemen uit te breiden om nieuwe bestandsformaten te ondersteunen.

In dit proefschrift adresseren we het gebrek aan gebruikersgestuurde en uitbreidbare cross-document link oplossingen. Onze aanpak is tweeledig en bestaat uit een gebruikersstudie en architectuur in combinatie met een prototype van een uitbreidbare link service. De gebruikersstudie is gebaseerd op een multi-case design dat bestaat uit een online vragenlijst samen met interviews met participanten van de vragenlijst, om zo het gedrag van gebruikers na te gaan bij het associeren van informatie. Zo werden ook sterke punten en tekortkomingen van bestaande systemen verkregen. De inzichten verkregen via deze gebruikersstudie lieten

ons toe om ontwerp-implicaties te formuleren die geschikt zijn voor een dynamisch uitbreidbare cross-document link oplossing.

De voorgestelde cross-document link service, gebaseerd op het RSL hypermedia metamodel, voldoet aan de noden van de eindgebruiker en laat het linken van arbitraire documenten op verschillende niveaus van granulariteit toe. In ons dynamisch uitbreidbare cross-document link service kan ondersteuning voor opkomende documentformaten toegevoegd worden door middel van nieuwe data en visualisatie plug-ins voor de link browser, of door externe document viewers te integreren via gateways. Onze cross-document link service ondersteunt de dynamische uitbreidbaarheid en configuratie van de ondersteunde documentformaten zonder dat de kern van de link service opnieuw opgestart moet worden. De gepresenteerde link service ondersteunt momenteel het linken van zes verschillende documentformaten waarvan er drie gintegreerd werden met hun respectievelijke externe document viewers.

De uitbreidbaarheid van de voorgestelde link service werd gevalueerd in twee verschillende evaluaties. Daarbovenop werd een studie voor eindgebruikers uitgevoerd om de

bruikbaarheid van de voorgestelde cross-media link service te evalueren. We hebben er vertrouwen in dat de gepresenteerde concepten voor een dynamisch uitbreidbare cross-document link service de beheerbaarheid van documenten in zogenaamde cross-media information spaces verbeteren, alsook het linken van verschillende bestandsformaten op een toekomstbestendige manier kan ondersteunen.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my parents Abed Elrahman Tayeh and Amna Tayeh for their motivation and aid throughout my life and study. Without their excellent support, I would have not had the courage and motivation to finish my PhD. In the same vein, I would like to thank all of my family members for their invaluable support, including my wife Haneen Tayeh, brothers, sisters, father in law Ibrahim Elhabil and my mother in law Inshirah Elhabil.

With regards to this PhD, my greatest gratitude goes to Prof. Dr. Beat Signer, co-director of the WISE Laboratory of the Vrije Universiteit Brussel. His logical way of thinking as well as his wide knowledge have been of great value for me. Prof. Dr. Signer was always available for advice, guidance, encouragement and fruitful stimulating discussions. I really appreciate his understanding, encouragement and personal guidance through the four years of my PhD journey. Furthermore, I am very grateful for his thorough and careful review of this thesis. I hope we can continue to cooperate together in the near future.

Next, I would like to thank the members of my PhD jury, Prof. Dr. Olga De Troyer, Prof. Dr. Bernard Manderick, Prof. Dr. Wouter Verbeke, Prof. Dr. Bruno Dumas and Prof. Dr. Angelo Di Iorio, for providing me with good pointers to further improve the quality of my text.

I would like to extend my gratitude towards most members of the WISE group research, for all the productive discussions as well as enjoyable chats. Special thanks goes to Reinout Roels, Dr. Joachim Vlieghe and Dieter Van Thienen for the proof-reading of some parts of my thesis.

I am also deeply indebted to all of my friends who supported me during this PhD and through the difficult times I went through in Belgium. Thanks to Mohammed Zaid, Zohair Qashlan, Omar El Mansi,

Rezeq Zomlot, Asharf Hamamreh, Mustapha El Baba, Dr. Ahmed Ewais, Dr. Abdalghani Mushataha, Dr. Wael Al Sarraj, Mohammed Ali, Mohammed El Najjar, Ashraf Dabour, Mohammed El Khaldi, Dr. Mahmoud El Najjar and Luis Alberto Guillen. Special thanks goes to Mohammed Saed, Ezziddeen Tayeh and Nidal Tayeh for taking care of me after some surgeries, the productive discussions regarding this thesis as well as for the proof-reading of parts of this thesis.

Finally, I would like to thank all the people I forgot to mention here who contributed to this PhD in some way or another.

*I do thank you all, **Ahmed***

Table of Contents

1	Introduction	1
1.1	Context	1
1.2	Problem Statement	3
1.3	Research Objectives	6
1.4	Research Questions	7
1.5	Research Approach and Methodology	7
1.6	Contributions	10
1.7	Thesis Structure	12
2	Background	15
2.1	Basic Terminology	15
2.2	A Short History of Hypermedia	17
2.3	Evolution of Document Linking	19
2.4	Link Models	22
2.4.1	The XML Linking Language (XLink)	22
2.4.2	The Resource Selector Link Metamodel	24
2.5	Linking Features in Document Formats	27
2.6	Hypermedia and Annotation Systems	31
2.6.1	Open Hypermedia Systems	32

2.6.2	Annotation Tools	35
2.6.3	Extensible Architecture for Annotation and Link Services	37
2.7	Discussion	39
2.8	Towards an Ideal Cross-Document Link Service	40
2.8.1	Requirements for an Ideal Link Service	40
2.8.2	A Comparison between Existing Annotation and Linking Systems	43
2.9	Summary	45
3	User Behaviour in Associating Information	47
3.1	Context	47
3.2	Literature Review	48
3.3	Methodology	49
3.3.1	Data Collection	50
3.3.1.1	Online Survey	50
3.3.1.2	Interviews	52
3.3.2	Population	53
3.3.3	Data Analysis	53
3.4	Results	53
3.4.1	Associating Information in Physical Documents .	56
3.4.2	Associating Information in Digital Documents . .	59
3.4.2.1	Single Digital Document	59
3.4.2.2	Different Documents of the Same Docu- ment Type	61
3.4.2.3	Different Documents of Different Types	63
3.4.3	Associating Information Across Physical and Di- gital Documents	65

3.4.4	A Deeper Look into the Information Association Mechanisms	66
3.4.4.1	Characteristics of the Associations	66
3.4.4.2	User Satisfaction with Used Association Mechanisms	69
3.4.4.3	The Need for a Linking Tool	71
3.4.4.4	User Work Practices	72
3.5	Discussion and Design Implications	73
3.6	Threats to Validity	76
3.7	Summary	77
4	A Dynamically Extensible Cross-Document Link Service	79
4.1	Proposed Solution	80
4.1.1	Link Browser	82
4.1.2	Link Browser Extensibility	85
4.1.3	Link Model	85
4.1.4	Integration of Third-Party Document Viewers	88
4.1.4.1	Communication Between the Link Service and Add-ins	91
4.1.5	Plug-in Metadata	93
4.1.6	Users of the Link Service	93
4.1.6.1	End Users	93
4.1.6.2	Third-Party Developers	95
4.2	Architecture Overview	95
4.3	Communication Between Link Service Components	97
4.4	Dynamic Link Service Extensibility	100
4.4.1	Metadata and Online Repository	102
4.4.2	Plug-in Tracking	104

4.5	Integration of Document Formats	105
4.5.1	Data Plug-ins	105
4.5.2	Visual Plug-ins	106
4.5.3	Requirements for Integrating a Document Format	110
4.5.4	Supported Document Formats	111
4.6	Integration of Third-Party Document Viewers	113
4.6.1	Third-Party Document Viewer Add-ins	113
4.6.2	Gateway Plug-ins	115
4.6.3	Communication Channels	118
4.6.3.1	Messages	119
4.6.4	Requirements for Integrating a Document Viewer	120
4.6.5	Supported Document Viewers	121
4.6.5.1	Google Chrome Add-in	121
4.6.5.2	HTML Document Format Plug-ins	122
4.7	Discussion	122
4.8	Summary	125
5	Evaluation	127
5.1	Supported Document Formats and Viewers	127
5.1.1	PDF Document Format	128
5.1.2	XML Document Format	128
5.1.3	Images	129
5.1.4	Microsoft Word	130
5.1.4.1	JavaScript-based Add-in	130
5.1.4.2	C# Add-in	132
5.1.4.3	Word Document Format Plug-ins	133
5.1.5	Microsoft PowerPoint	133

5.1.5.1	Microsoft PowerPoint Add-in	133
5.1.5.2	PowerPoint Document Format Plug-ins	133
5.1.6	Google Chrome	133
5.1.6.1	Google Chrome Add-in	134
5.1.6.2	YouTube Video Plug-ins	135
5.1.7	Discussion	135
5.2	Integration of Existing Document Formats and Viewers	136
5.2.1	Methodology	136
5.2.2	Results	137
5.2.2.1	Third-Party Document Viewers	137
5.2.2.2	Document Formats	139
5.2.3	Discussion	140
5.3	End-User Evaluation	141
5.3.1	Goal	141
5.3.2	Methodology	141
5.3.3	Population	142
5.3.4	Setup	142
5.3.5	Results	143
5.3.6	Discussion	146
5.4	Summary	147
6	Use Case	149
6.1	Document Retrieval Systems	150
6.2	Motivation	151
6.3	Enhanced Document Retrieval and Discovery	152
6.3.1	Use of a Document's Content and Metadata	153
6.3.2	The Use of Explicit Hyperlinks	154

6.3.3	Support for Multiple Visualisations	156
6.3.4	System Architecture	158
6.3.5	Implementation	160
6.4	User Evaluation	161
6.5	Summary	164
7	Conclusions and Future Work	165
7.1	Summary	165
7.1.1	User Study	166
7.1.2	A Dynamically Extensible Cross-Document Link Service	166
7.2	Discussion	168
7.2.1	Contributions	170
7.2.2	Limitations	172
7.3	Future Work	173
	Appendices	175
A	Survey on Cross-Document Associations	177
B	Abstract DefaultDocument Class for Visual Plug-ins	193
C	Gateway Interface	199
D	Link Service Evaluation Questionnaire	203

1

Introduction

1.1 Context

Documents do not exist in isolation but are often related to other documents. These inter-document relationships can either be defined via explicit references and footnotes or be established implicitly based on the similarity of content in different documents. Associative hyperlinks are considered the basis for creating and managing relations and associations between documents as well as other information objects [79]. They were originally introduced in Vannevar Bush's visionary paper about the Memex [26]. Rather than classifying documents in hierarchical structures as realised in the Dewey Decimal Classification (DDC), Bush proposed to mimic the working of the human brain by supporting associative hyperlinks or so-called trails between documents. The associative hyperlinks proposed by Bush were seminal for succeeding digital hypermedia models and systems such as the oN-Line System (NLS) [51]. The concept of hyperlinks was further instrumental in the success of the World Wide Web by enabling the referencing, annotation and augmentation of content.

Most recent digital document formats support simple forms of linking, enabling users to associate information across different documents. For instance, the well-known Hypertext Markup Language (HTML) supports simple linking features that allow users to define unidirectional

hyperlinks that can be traversed only from their source to their target. Like HTML, the Portable Document Format (PDF) [3], the Office Open XML (OOXML) standard [50] or EPUB [41] support simple unidirectional hyperlinks to define associations between documents [134]. The simple linking features offered by most document formats only allow users with write permissions to the source document (e.g. the owner of a document) to create new hyperlinks. For example, hyperlinks within a web page represented by an HTML document always have to be authored by its developer. Thereby, users without write permissions who would like to associate information across documents are not able to create hyperlinks based on the simple document linking features. The unidirectional hyperlinks offered by most document formats further imply that a linked document (target) as well as its reader are not aware of hyperlinks pointing to it from other source documents.

Many solutions including open hypermedia systems, annotation systems or the XLink standard [46] have been proposed in order to address the shortcomings of the previously mentioned simple hyperlinks. Some of these solutions have been developed to open web documents for third-party annotations and associations to external resources as part of the Web 2.0 movement where users have become producers as well as consumers of information. Open hypermedia systems (also known as link services) separate a document's content from its hyperlinks; something that is not supported by most document formats. In order to do so, link services manage hyperlinks separately from the linked documents in centralised databases or so-called linkbases. Thereby, users without write permissions are able to create new hyperlinks in any of the document formats supported by these systems. Some open hypermedia systems such as Webvise [66] and Arakne [21] have enabled third-party users to augment web pages with new hyperlinks.

The advent of the Extensible Markup Language (XML) [60] and its linking model (XLink) has been a major step towards enhancing linking on the Web. XML and XLink can be used to create hyperlinks that go beyond the simple unidirectional hyperlinks. Similar to open hypermedia systems, XML in combination with XLink can be used to separate a document's content from its hyperlinks, allowing users without write permissions to create hyperlinks between different XML documents. Besides the simple unidirectional hyperlinks, XLink also supports so-called *extended hyperlinks* which enable the definition of bi- and multidirectional hyperlinks.

Many annotation systems allow users without write permissions to annotate and attach external resources to the referenced documents. An annotation can be considered as a hyperlink [8, 126] since it establishes a relationship between parts of a document and external resources (e.g. comments or notes). Nowadays, most document viewers (e.g. Adobe Acrobat Reader¹) are shipped with built-in annotation features that allow users to annotate the supported document formats. More recently, various digital library management systems (DLMSs) [53] have incorporated interactive annotation features that facilitate discussion among users and open the documents in digital libraries for linking to external resources. While in most DLMSs the annotation features are offered by built-in components, the Flexible Annotation Service Tool (FAST) [6] has been developed as a stand-alone annotation tool in order to offer its services to multiple DLMSs.

1.2 Problem Statement

There is no doubt that specific document formats and hypermedia solutions facilitate the creation of associations between “some” documents. Nevertheless, we identify three main shortcomings with respect to the offered linking features. First, while many document formats offer the possibility to link to entire third-party documents, it is normally not possible to address parts of documents. Second, document formats as well as existing hypermedia solutions support the linking to a specific (mostly predefined) set of document formats and are not extensible to support the linking to additional existing as well as emerging document formats. Third, the development of most existing linking solutions has been solely motivated by limitations of their predecessors or new features to be supported rather than being based on a clear understanding of the needs and requirements of end users. In the following, we further elaborate on each of these three shortcomings.

Addressing parts of documents: Some document formats have taken into account that relationships between documents might exist at any level of granularity—at the document, paragraph or even word level—by enabling the linking to snippets of information in other documents of the same document format. For instance, the HTML document format allows the creation of hyperlinks to snippets of information in

¹<https://get.adobe.com/reader>

other HTML documents. XML and XLink can also be used to create hyperlinks that address parts of other XML documents. Nevertheless, *most document formats do not offer the possibility to link to snippets of information in documents of another document format*. For example, while a developer can create hyperlinks in an HTML document targeting entire third-party documents which are supported by the growing Web (e.g. a PDF or Word document), they cannot address parts of these documents. In a PDF document we can also create hyperlinks to entire Word documents via a Uniform Resource Identifier (URI) but we cannot link to specific parts within a Word document. Please bear in mind that in the rest of this thesis we will use the term *cross-document linking* for hyperlinks between snippets of information (at any level of granularity) in different documents (regardless their formats).

Extensibility of linking solutions: Over the last decades, we have witnessed a dramatic increase in the number of digital document formats that we use as part of our daily activities. This implies that information is scattered over these documents and therefore we should be able to create associations between pieces of information in the different document formats. *This asks for an extensible linking solutions enabling the linking to existing as well as emerging document formats*. Nowadays, we see this extensibility feature only realised for hyperlink resolution in web browsers. When a web page links to an entire third-party document (e.g. PDF or Word), the web browser calls a specific plug-in to visualise the document based on the media type² (formerly MIME type) of the hyperlink's target. The extensibility of a linking solution should not be limited to resolving hyperlinks but also allow users to create hyperlinks between snippets of information in different document formats. Moreover, it should be possible to seamlessly integrate any document format in the linking solution.

The linking features of various document formats often imply that hyperlinks can only be defined to web resources and are not extensible to support other existing or emerging document formats. This means that the possibilities for creating explicit associations between desktop documents via hyperlinks are rather limited. One would expect that the emergence of cloud computing [13] would bridge the gap between desktop and web documents, especially since an increasing number of document formats (e.g. Word or PowerPoint document formats) are edited and stored in the cloud. However, as previously mentioned, the HTML linking

²<https://www.w3.org/TR/CSS21/media.html>

features are not sufficient to address parts of documents supported by the growing Web when linking to them.

Link services were mainly developed to enhance the management of hyperlinks. Most existing link services have two main shortcomings. First of all, they support the linking across a predefined set of document formats that have to be visualised and authored within the link service itself. In other words, users have to leave their preferred third-party document viewers (e.g. Microsoft Word or Adobe Acrobat Reader) and their supported document formats (e.g. Word or PDF) in order to profit from the features offered by a link service. Second, it is not evident how the architectures of existing link services can be extended in order to support other existing and emerging document formats [135, 134]. Usually, in order to extend a link service to support a new document format, its link model and the user interface have to be extended [125, 126]. The link model has to be extended in order to be able to address parts of documents in a new document format. On the other hand, the user interface has to be extended in order to support the visualisation of the new document format. Unfortunately, even though there are some link services that have extensible link models, to the best of our knowledge none of them offers the extensibility on the visualisation (user interface) layer [125, 126, 135, 134]. This has the drawback that these link services have to be reimplemented or redeployed whenever a new document format has to be supported.

Most web and digital library annotation systems as well as the built-in annotation features of document viewers have one major limitation. They only support simple annotation features (e.g. notes or comments) and it is not evident how their architectures can be extended in order to support the linking across different document formats. It is worth mentioning that some of the web annotation tools are based on the powerful XLink standard but do not exploit its strengths for linking between XML documents.

User-driven linking solutions: The understanding of end-user behaviour in associating and linking documents as well as the end-user requirements for a document linking solution leads to the development of successful and usable linking solutions. In fact, there have been different studies in a number of domains that revealed some interesting general findings regarding the user behaviour in associating information. For example, some studies have shown that users rely on digital or physical folders when organising and associating *entire* documents [81, 18, 90].

Other studies revealed that the cross-document referencing task is carried out while reading and writing [2, 109, 110]. Nevertheless, we could not find a single study mainly investigating user behaviour in linking and associating information within and across documents. This lack of study results might be a main reason why the development of most existing linking solutions is not driven by end-user requirements.

1.3 Research Objectives

This dissertation consists of two main research parts. In the first part of our research we investigate the current user behaviour in associating information within and across documents. The result of this research provides some insights about end-user requirements for a cross-document linking solution.

The second essential part of the research focuses on investigating the possibilities to support cross-document linking features in existing as well as emerging document formats. We are interested in realising an extensible cross-document link service that helps users in associating information within and across documents. This means that we are not considering the development of a new document standard to overcome the limitations of existing document formats. Our research is situated in the domain of Hypermedia and Software Engineering. We come up with a cross-document link service that supports the integration of existing as well as emerging document formats. Our link service makes use of an existing powerful link metamodel. Therefore, it is worth to note that most of Our cross-document link service takes into account that users rely on proprietary third-party document viewers to author and visualise specific document formats (e.g. Microsoft Word or Adobe Acrobat Reader). Therefore, our link service is not only extensible to support new document formats, but it also addresses the challenge of seamlessly integrating third-party document viewers. Furthermore, our link service is *dynamically extensible*. We can outline various reasons why a link service should be dynamically extensible. First of all, it is not feasible to extend or redeploy an existing link service every time a new document format has to be supported. Imagine that each time we navigate to a new document type on the Web, we would have to install a new version of the web browser. This would definitely be a big burden for any user. Further, each user might only make use of a small subset from the mul-

titude of existing document formats. Rather than having a monolithic link service that supports all document formats, users should be able to dynamically extend the link service in order to support their preferred document formats. Finally, offering cross-document linking features to proprietary document viewers should not ask for changes to the core of these systems since this might not be accepted by their creators.

In the proposed link service we also integrate different document formats as well as third-party document viewers. There is no doubt that the integration of different document formats in the link service serves as an assessment of the link service's extensibility. Nevertheless, we also conduct a technical evaluation for the link service's extensibility. Last but not least, we evaluate the usability of the proposed link service in an end-user study. It is worth mentioning that our end-user evaluation does not address the potential cognitive overhead of creating and navigating the hyperlinks using our link service.

1.4 Research Questions

This dissertation provides answers to the following research questions and the related sub-questions that have been formulated based on the previously mentioned problem statement:

1. **RQ1: What is the user behaviour in associating information within and across documents?**
2. **RQ2: How to support cross-document linking functionality in different document formats?**
 - (a) RQ2.1: What are the design and end-user requirements for a dynamically extensible cross-document link service?
 - (b) RQ2.2: How to support extensibility in a cross-document link service—in the link model and on the visualisation layer—in order to support existing and emerging document formats?

1.5 Research Approach and Methodology

We have adopted the Design Science Research Methodology (DSRM) defined by Peffers et al. [113] to tackle the formulated research ques-

tions. DSRM provides a process model for carrying and evaluating design science research in information systems. The DSRM process model includes six steps: 1) problem identification and motivation, 2) definition of the objectives for a solution, 3) design and development of the solution, 4) demonstration, 5) evaluation and 6) communication. In the rest of this section, we explain how these steps have been realised in our research.

1. **Problem identification and motivation:** This step has been covered in Section 1.2. It has been realised by acquiring knowledge of the state of art in document linking approaches and the importance of finding solutions for the problem statement discussed in Section 1.2.
2. **Definition of the objectives for a solution:** The objectives of the solution have been defined based on two empirical literature studies. Some objectives of the solution have been defined after the first empirical literature study which investigated the supported linking features in different document formats, existing link standards, annotation tools and link services. The objective of the first study was to provide answers for some *knowledge questions* [142] in order to highlight the shortcomings of existing linking solutions. These knowledge questions include but are not limited to i) *what are the supported linking features in some popular document formats*, ii) *what are the solutions proposed to enhance the linking in document formats* and iii) *what are the architectures used by the different linking solutions and what are their strengths and limitations*. The answers provided by the knowledge questions are presented in Chapter 2. Other objectives of the solution have been defined after realising that there is no single study investigating the current user behaviour in associating information within and across documents of different formats. Therefore, we have carried out a second empirical literature study of some previous research studies that revealed general findings regarding the associating of information as presented later in Section 3.2. This second literature study has helped us in carrying out an exploratory study that is mainly investigating the user behaviour in associating information within and across documents and is introduced in Chapter 3.

The answers from both empirical studies were used to outline the objectives for realising the solution proposed in this dissertation. The objectives of the solution were used to rationalise the signific-

ance of the problem specification described in Section 1.2 as well as the suitability of the research questions listed in Section 1.4.

3. **Design and development of the solution** We have applied a mixed methods approach in order to explore the current user behaviour in associating information across documents. The mixed methods approach involved an online survey as well as interviews. Our study has revealed a number of interesting results based on which we derived some design implications and requirements for a dynamically extensible cross-document link service. The exploratory study, its results as well as the design implications are presented in Chapter 3.

Our proposed solution is a dynamically extensible cross-document link service which fulfils the second part of the objectives of a solution defined in the second research step in Section 1.3. There were multiple research steps involved. First, we outlined essential requirements for an ideal cross-document link service that is extensible and flexible to support existing as well as emerging document formats. This step provided answers for a part of the knowledge question RQ2.1. Second, we provided answers to RQ2.2 by designing an extensible architecture for a cross-document link service. This step indeed has given answers to multiple knowledge as well as design questions [142]. Knowledge questions contained in this step include but are not limited to i) *what are the extensibility approaches and mechanisms that can be used in a cross-document linking solution* and ii) *how can existing third-party document viewers be integrated in a cross-document linking solution. What is an architecture for an extensible dynamic cross-document link service* was the main design question contained of this step.

4. **Demonstration** In this step, we have demonstrated the usefulness of the proposed cross-document link service by integrating multiple document formats as well as external third-party document viewers with the link service. Our link service currently supports the linking of XML, plain text, HTML, PDF, Word and PowerPoint document formats. The HTML, Word and PowerPoint documents are supported by integrating their own third-party document viewers (i.e. Google Chrome, Microsoft Word and Microsoft PowerPoint). Users are able to create bi- and multidirectional hyperlinks between the supported document formats. Moreover, we

have demonstrated the usefulness of the proposed solution by developing an enhanced desktop environment which is presented in Chapter 6 and exploits the hyperlinks defined via the link service in combination with some data mining algorithms to enhance the search and retrieval of desktop documents.

5. **Evaluation** The presented link service has been evaluated in three different evaluations which are going to be presented in Chapter 5. Two evaluations were conducted to validate the extensibility of the

link service whereas the third evaluation investigated the usability of the link service in an end-user study.

6. **Communication** Some of the findings of this dissertation have been disseminated at the International Conference on Web Information Systems Engineering (WISE) where two full papers describing the link service have been published:

- (a) Tayeh, A.A.O. and Signer, B.: “A Dynamically Extensible Open Cross-Document Link Service”, Proceedings of WISE 2015, 16th International Conference on Web Information System Engineering, Miami, USA, November, 2015
- (b) Tayeh, A.A.O. and Signer, B.: “Open Cross-Document Linking and Browsing based on a Visual Plug-in Architecture”, Proceedings of WISE 2014, 15th International Conference on Web Information System Engineering, Thessaloniki, Greece, October, 2014

1.6 Contributions

In the following, we would like to highlight the main contributions of this dissertation.

1. **A user study investigating the user behaviour in associating information within and across documents**

To the best of our knowledge our study is the first study that investigates the user behaviour in associating information within and across different document formats. Our user study investigates

the behaviour of 238 knowledge workers in associating information. The contributions of the user study can be summarised as follows:

- (a) The user study revealed association mechanisms adopted by participants when associating information in digital as well as physical documents. It further identified some characteristics of different types of associations identified by users as a result of the different association mechanisms.
- (b) The user study revealed some general findings about common work practises and the appreciation or criticism of users with regard to each association mechanism.
- (c) Based on the results of the user study we formulated a number of design implications for an effective information and cross-document linking solution.

2. Requirements for an ideal link service

Based on a critical review and analysis of existing linking solutions as well as our expertise, we managed to derive a number of fundamental requirements for an extensible cross-document link service. These requirements have been published in [135].

3. A dynamically extensible cross-document link service

To the best of our knowledge, our link service that has been published and presented at the WISE conferences [134, 135] is the first linking solution that is dynamically extensible on the link model as well as the user interface level. In contrast to most existing linking solutions, our link service allows users to create advanced hyperlinks (i.e. bi- and multidirectional hyperlinks) between documents. Furthermore, existing third-party document viewers can seamlessly be integrated with our link service. Thereby, in contrast to most existing link services, users willing to benefit from our link service should not have to abandon their preferred third-party document viewers. They can create hyperlinks between documents that are visualised in the link service as well as documents shown in their personal third-party document viewers. Besides a working link service prototype we made a number of contributions:

- (a) We presented an architecture for a dynamically extensible cross-document link service.

- (b) Based on a visual plug-in mechanism, our link service’s user interface (link browser) is extensible to support existing as well as emerging document formats.
- (c) We proposed a mechanism for integrating existing as well as emerging third-party document viewers with our link service. In contrast to some existing linking solutions, our proposed mechanism does not require changes to the core of third-party document viewers and further allows any “extensible” third-party document viewer (e.g. Google Chrome) to be seamlessly integrated with our link service.
- (d) We integrated a number of document formats (e.g. plain text, XML, Word and PDF) as well as third-party document viewers (e.g. Microsoft Word, Microsoft PowerPoint and Google Chrome) with our link service. We further illustrated the usefulness of our link service by presenting a framework that exploits the link service’s hyperlinks in a desktop document retrieval tasks.

1.7 Thesis Structure

The remainder of this dissertation is structured as follows:

Chapter 2. Background. In this chapter we discuss the necessary background and the state of art in document linking approaches. We discuss the evolution of document linking approaches as well as the linking support in existing document formats and link models. After discussing existing link services and highlighting their shortcomings, we outline six fundamental requirements for an ideal cross-document link service.

Chapter 3. User Behaviour in Associating Information. In this chapter we present our user study that investigates end-user behaviour in associating information within and across document formats. After presenting the results of the user study and critically discussing its results, we propose a number of design implications for the development of future cross-document linking solutions.

Chapter 4. A Dynamically Extensible Cross-Document Link Service. In this chapter we present our link service that overcomes the shortcomings of existing linking solutions and allows end users to create advanced hyperlinks across documents of different formats. We discuss

the proposed link service after presenting its proposed architecture and components.

Chapter 5. Evaluation. In this chapter we present the three different evaluations of the link service including two technical evaluations as well as an end-user study. In the technical evaluations we critically evaluate and discuss the extensibility of our link service. In the end-user study we evaluate and discuss end-users' satisfaction with the presented link service.

Chapter 6. Use Case. In this chapter we illustrate the usefulness of our link service by presenting a framework for enhancing desktop document discovery and retrieval that exploits the link service's data. We further present an end-user study of the presented framework.

Chapter 7. Conclusions and Future Work. After critically discussing our user study and the proposed solution for cross-document linking, in this chapter we propose future research directions and provide some concluding remarks.

2

Background

In this chapter, we discuss the necessary background and the state of art in document linking approaches. We start by introducing the basic terminology that will be used in this thesis. We briefly outline the history of hypermedia/hypertext systems and models before discussing the evolution of document linking approaches. After presenting existing link models, we provide a review of the hyperlink support in existing document formats. We then discuss existing document linking systems and highlight their limitations regarding the cross-document linking features. We conclude this chapter by outlining six important requirements for an ideal cross-document link service and provide a comparison between existing linking systems in the light of these six requirements.

2.1 Basic Terminology

Before we start presenting related work, we introduce some basic hypermedia terminology that is used in this thesis. The concept of a hyperlink is heavily used in this thesis. A hyperlink simply defines an association between two or more information entities (e.g. images, documents or multimedia content). Nowadays, the most frequently used hyperlinks are so-called *unidirectional hyperlinks* (e.g. web hyperlinks). A unidirectional hyperlink implies that the hyperlink L is directed from a source

entity **S** to a target entity **T** as shown in Figure 2.1. This means that the hyperlink can be traversed only from the source to the target. The target entity further is not aware of the explicit relationship defined within the source entity.

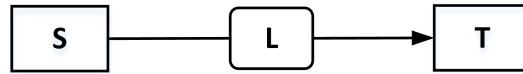


Figure 2.1: Unidirectional link **L** directed from a source entity **S** to a target entity **T**

A unidirectional HTML hyperlink is illustrated in Listing 2.1. In this example, the hyperlink **L** is defined using the `<a>` tag. The hyperlink source **S** is specified as the word **VUB**. A user should click on the **VUB** word in order to navigate to the hyperlink target **T**. The hyperlink target is the VUB website whose address is given by the `href` attribute. In contrast to unidirectional hyperlinks, a *bidirectional hyperlink* implies that a hyperlink can be traced from both endpoints as shown in Figure 2.2. Even though the hyperlink can be traced from both endpoints in a bidirectional hyperlink, it is still very common to name one endpoint as a source and the other one as a target.

```

1 <!DOCTYPE html>
2 <html>
3   <body>
4     <a href="http://www.vub.ac.be/">VUB</a>
5     ...
6   </body>
7 </html>
  
```

Listing 2.1: Unidirectional HTML hyperlink

A *multidirectional hyperlink* defines an association between one or multiple source entities and one or multiple target entities. As illustrated in Figure 2.3, a multidirectional hyperlink can either be a multi-target (Figure 2.3a), a multi-source (Figure 2.3b) or a multi-source multi-target hyperlink (Figure 2.3c).

A *link model* defines a conceptual model with abstractions and concepts to link different entities in a hypermedia application or in specific document formats. The link features offered by different link models range from simple unidirectional hyperlinks to advanced multidirectional hyperlinks. Some link models [124, 46] are general enough to be used in many hypermedia applications, while other models [75, 22] provide

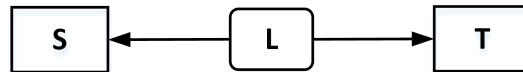


Figure 2.2: Bidirectional hyperlink which can be traced from both end-points

application-specific features such as the adaptation of the linked entities. It is worth mentioning that many hypermedia applications [112, 68] are not based on a documented and explicit link model, but are rather using some hard-coded linking features.

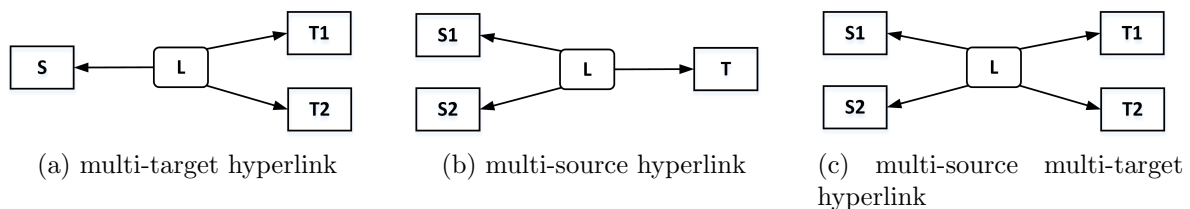


Figure 2.3: A multidirectional hyperlink connects one or multiple source entities with one or multiple target entities

The last two main concepts are *embedded* and *external hyperlinks*. An embedded hyperlink implies that the hyperlink is forming a part of a document. For example, an HTML hyperlink has to be authored by the document owner and is represented via an `<a>` tag that is located in the HTML document itself (see Listing 2.1). In most cases, a document format which supports embedded hyperlink features implies that only the owner of a document can add new hyperlinks to the document. In contrast, an external hyperlink does not form a part of the document's content but is rather stored externally from the linked document. The idea of external hyperlinks has been envisioned by Vannevar Bush in 1945 [26] and has been used in open hypermedia systems such as Sun's link service [68] and Microcosm [73]. A hypermedia application supporting external hyperlinks may allow any user to define new hyperlink for its supported document formats.

2.2 A Short History of Hypermedia

The Memex [26] envisioned by Vannevar Bush is accredited as being the origin of hypertext systems. According to Bush's vision, users should use

the Memex system in order to introduce new trails (associative hyperlinks) between snippets of information in different documents stored in a persistent storage of microfilms that was used at that time. Every associative hyperlink should be identified via a unique identification code and then stored externally from the linked documents. Thereby, users are able to later retrieve any associative hyperlink along with the linked documents by entering the associative hyperlink's code. Inspired by Bush's associative hyperlinks and trails, Ted Nelson coined the term *hypertext* as *non-sequential writing* [105], where at any time, pointers can be introduced in documents which direct the reader to a different section, paragraph or another (part of a) document. The term hypermedia was also coined by Nelson [104] and implies that hyperlinks are defined across different types of media such as images, movies, text or audio.

Over the last decades various hypermedia models for different domains have been proposed based on the hyperlink concept. Xanadu [105], a well-known research project, has been proposed by Nelson. The Xanadu project and its Xanalogical model introduced various new ideas to the hypertext community, including advanced hypertext concepts such as bidirectional hyperlinks and transclusion which enables the inclusion of content by referencing it. The Dexter hypertext reference model [71] is another well-known model for hypertext systems. It defines a conceptual model for abstractions and concepts found in many hypermedia systems in order to promote the interoperability and interchange between the different hypermedia systems. The Amsterdam Hypermedia Model (AHM) [75] enriched the Dexter reference model with the notions of time and context and thereby provides abstractions for linking dynamic multimedia information. Furthermore, the Fundamental Open Hypertext Model (FOHM) [100] was built in order to support the interoperability between hypermedia systems by allowing the exchange of metadata and information about hyperlinks. Moreover, the AHAM reference model [22] is a Dexter-based reference model that provides abstractions for content and hyperlink structure adaptations in adaptive hypermedia systems. As previously mentioned, XLink is a powerful link model proposed to enhance the linking in XML documents. XLink offers advanced linking functionality such as bi- and multidirectional external hyperlinks. Last but not least, the Resource Selector Link (RSL) metamodel [124] is a general and flexible link metamodel that has been used in various hypermedia systems over the last decade. It is worth mentioning that we have used the RSL metamodel as a link model for our cross-document link

service and therefore a brief explanation of RSL and XLink is presented in Section 2.4. A motivation for the use of the RSL metamodel and a comparison of existing link models is provided later in Section 4.5.1.

Besides the hypermedia models, various hypermedia applications have been developed for different domains. The Hypertext Editing System (HES) [30], FRESS [47] and NLS [51] were early pioneering hypertext research prototypes. Note that the HES system pioneered the concept of unidirectional hyperlinks. As mentioned earlier, many open hypermedia systems such as Intermedia [68] and Sun's link service [112] have been developed in order to enhance the management of hyperlinks. The Web is the most successful hypermedia system that enables the referencing, annotation and augmentation of content. Note that the Web and its Semantic technologies such as RDF [43], RDF Schema and OWL [10] promoted the *linked data* concept [76], where Web content and Web hyperlinks are semantically enriched in order to enable the processing of data by computers.

2.3 Evolution of Document Linking

The fact that documents do not exist in isolation [26] has always been the motivation for many researchers in different domains (e.g. hypermedia, document engineering and digital libraries) to develop models and systems for enhancing the linking across documents. The Memex system, mentioned in Section 2.2 and depicted in Figure 2.4 has been envisioned by Bush in 1945 as a result of his observation of the existence of relations between documents. The Memex which looks like a desktop has two displays for visualising two adjacent pages. The pages are stored on microfilms and are accessed by the Memex in order to project them on the two displays. A user is able to create trails between the projected pages. Even though the Memex is based on relatively naive design concepts, it introduced a number of interesting ideas for linking different documents. Inspired by Bush's Memex, many early hypermedia systems such as HES and Xanadu have made significant contributions to concepts for linking across documents. HES was developed by Andries van Dam and Ted Nelson at Brown University. Even though the main focus of HES was on text editing, formatting and printing, it pioneered many other hypermedia systems including the Web.



Figure 2.4: The visionay Memex system, based on [26].

Starting from 1960 and for almost twenty years [12], documents were very simple due to their simple document production systems. In other words, they were totally different from today's rich and sophisticated document formats. Documents were essentially composed of textual content using simple editing systems and formatted using simple low-level formatters [12]. Examples of such simple editing and formatting systems are TROFF [82] and RUNOFF [119]. Due to the simplicity of document structures at that time, linking features were not supported in logical document structures but were rather offered at the application level (e.g. in HES).

The introduction of Scribe [115], the first markup language [12], was a revolutionary step towards a new generation of document formats (i.e. markup languages). With the markup language approach, a document is no longer dependant on simple formatters anymore but rather depends on its supported markup commands for describing relations between content. Furthermore, in general a markup language produces portable non-proprietary documents where documents can be edited and visualised using any document viewer that understands its specification. An HTML document, for instance, can be visualised using arbitrary web browsers such as Google Chrome or Firefox.

Scribe and other early markup languages such as the Generalised Markup Language (GML) [58] do not support the linking to other documents or external resources. These document formats only enable so-called *cross-references* within a document itself such as references to figures, tables or headings. The introduction of the Standard Generalised Markup Language (SGML) [59] was a major step towards supporting linking in different document formats. A number of concepts have been introduced in the SGML metalanguage that facilitate the creation of embedded hyperlinks addressing parts of the same document or other documents. SGML hyperlinks can be created using complex structures and attributes describing the relationships between document nodes and other documents. A hyperlink in SGML is represented as a separate element which can potentially be enriched with additional metadata. Inspired by SGML, HTML also supports embedded hyperlinks. However, in contrast to SGML hyperlinks, HTML hyperlinks are easy to create as illustrated in Listing 2.1. Most recent markup document formats such as DocBook [138] have been inspired by the HTML link features and offer simple embedded and unidirectional hyperlinks to other document formats.

In contrast to markup languages, many recent document formats such as PDF or Word do not discriminate between a document's content and its formatting. These document formats adopted the *What You See Is What You Get* (WYSIWYG) approach for document production. A WYSIWYG document is modelled as a sequence of paragraphs [57] and is represented as monolithic blocks of linear content [122] without any semantic interpretation [17]. In other words, as criticised by various authors [103, 122, 134] WYSIWYG document formats stuck to a conservative representation of information and have degraded the computer to a paper simulator. Thereby, the WYSIWYG document formats do unfortunately not contribute to the advancement of document linking, but rather emulate the simple linking features of early markup languages.

The simple forms of unidirectional and embedded linking that are supported by most document formats yielded to many research and commercial systems that tried to offer more advanced linking features for different document formats. A number of link models such as XLink have been proposed in order to enrich some document formats with more linking features. Furthermore, many open hypermedia systems as well as annotation systems have been developed to play a major role in opening some document formats for third-party associations and annotations.

In the following sections, we elaborate on the linking features supported in most popular document formats, some important link models as well as different systems that contributed to enhancing the linking of documents.

2.4 Link Models

2.4.1 The XML Linking Language (XLink)

The Xlink model has been introduced in 2001 [46] in order to enable the linking across XML documents. It also overcomes the limitations of embedded unidirectional web hyperlinks. XLink offers the possibility to create hyperlinks that go beyond the simple embedded unidirectional hyperlinks. As mentioned before, XLink supports external hyperlinks that can be either simple unidirectional or extended hyperlinks. An extended hyperlink defines associations and paths between a collection of resources, where each path associates two resources. A resource may be linked to any subset of resources forming part of the collection. This mechanism allows for bi- and multidirectional hyperlinks. Listing 2.2 shows an example of an extended multidirectional hyperlink that is defined between three different resources (three different Google service web pages). Four different associations (arcs) have been defined between the three Google service web pages.

Based on XPointer expressions [45], XLink can be used to create hyperlinks that address parts of an XML structure. XPointer is mainly based on XPath [38] that can be used to navigate through the XML document tree. Moreover, with XPointer expressions it is further possible to realise generic hyperlinks. A generic hyperlink associates (parts of) documents based on string pattern matching [78]. For example, using XLink and XPointer a generic hyperlink can be defined from any occurrence of a specific term in an XML document to another term in another XML document. Even though the use of XPointer and XLink seems advantageous, XLink shows one main shortcoming. It is limited to tree-like document models since XPointer is only applicable to tree document models. In fact, according to the XPointer specification [45], XLink hyperlinks with their XPointer expressions are limited to a subset of XML-based documents.

```

1 <!-- List of Google Services, google.xml document -->
2 <googleServices>
3   <services xlink:type= "extended"
4     xmlns:xlink= "http://www.w3.org/1999/xlink">
5     <owner>Google</owner>
6 <!-- locator elements -->
7   <service xlink:type="locator" xlink:label= "s1"
8     xlink:href="http://www.google.com"
9     <title>Google Search Engine </title>
10  </service>
11  <service xlink:type="locator" xlink:label= "s2"
12    xlink:href="http://translate.google.com"
13    <title>Google Translate </title>
14  </service>
15  <service xlink:type="locator" xlink:label= "s3"
16    xlink:href="scholar.google.com"
17    <title>Google Scholar </title>
18  </service>
19 <!-- arcs -->
20  <next xlink:type="arc" xlink:from="s1" xlink:to="s2" />
21  <next xlink:type="arc" xlink:from="s2" xlink:to="s3" />
22  <previous xlink:type="arc" xlink:from="s2" xlink:to="s1" />
23  <previous xlink:type="arc" xlink:from="s3" xlink:to="s2" />
24 </services>
25 </googleServices>

```

Listing 2.2: An extended XLink hyperlink

There exist various implementations of the XLink standard. The Amaya¹ web browser and Mozilla Firefox implement the simple hyperlinks of the XLink standard. Amaya opens web pages for third-party annotations that are internally represented using XLink and RDF. The Goate [94] system implements XLink and represents an attempt to enhance the simple embedded link model for the Web. Goate is based on an HTTP proxy architecture to augment HTML documents with features of the XLink model such as bi- and multidirectional hyperlinks. Bidirectional hyperlinks have been realised by two unidirectional hyperlinks pointing to each other, while a multidirectional hyperlink has been realised by a collection of unidirectional hyperlinks with the same source. Goate offers these advanced hyperlink features by inserting the hyperlinks and destinations into an HTML document via a web proxy. The XLinkProxy [37] system and its descendant XLinkZilla [78] represent another approach that is similar to Goate and can be used for augmenting HTML documents with XLink features through a web proxy. Xspect [36] is another tool similar to XLinkProxy and Goate that fa-

¹<https://www.w3.org/Amaya>

cilitates the browsing and visualisation of a linkbase's hyperlinks on the Web. Webvise [66] is another system aimed to enrich web documents with third-party associations such as advanced multidirectional hyperlinks and annotations. It was built on top of an early Devise Hypermedia (DHM) framework [67, 64, 65] open hypermedia system which implements the Dexter Hypertext Reference Model. The multidirectional hyperlinks of Webvise are similar to the extended XLink hyperlinks and therefore some researchers considered Webvise as an implementation of XLink [78]. In the same way as Goate, XLinkZilla or XLinkProxy, DHM uses a web proxy architecture to augment web pages with multidirectional hyperlinks. Last but not least, X2X² represents a Java-based tool that implements XLink and facilitates the creation, management and storage of hyperlinks that are inserted into documents on the fly.

2.4.2 The Resource Selector Link Metamodel

The Resource Selector Link (RSL) metamodel [124] whose core that is depicted in Figure 2.5 has been proposed to be general and flexible enough in order to be used for evolving hypermedia systems. RSL is based on the concept of linking arbitrary entities, whereas an entity is either a resource, a selector or a link. A *resource* is an abstract concept that represents a media type such as a video, a text, an audio file or a complete document. The RSL resource must be extended in order to represent the concrete media types that exist in a given hypermedia application. The *selector* is an abstract concept to address parts of a resource. The RSL selector must be extended in order to represent the concrete fragment identifier of a specific media type. For example, a text selector for a text resource can be defined by its start and end indices whereas a video selector might be defined as a timespan. The **RefersTo** association between the resource and selector concepts with the corresponding cardinality constraints reflects the fact that a selector always has to be associated with exactly one resource, while many selectors can be defined and attached to a resource. Finally, the *link* concept defines a one-to-one, one-to-many, many-to-one or many-to-many association between any entities. The associations **HasSource** and **HasTarget** enforce that each link must have at least one source and at least one target. By using such a constraint, a hypermedia system can to some extent overcome the problem of broken or dangling hyperlinks. A hypermedia system that is based

²<http://xml.coverpages.org/x2xAnn19991202.html>

on the RSL metamodel might, for example, not allow the deletion of a document that is still a target of a hyperlink.

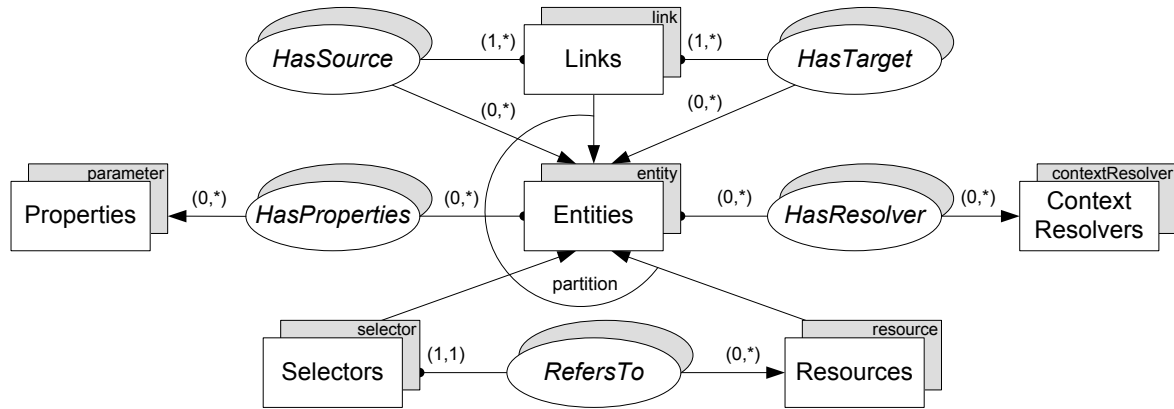


Figure 2.5: Core RSL components, based on [124].

The RSL core also provides two interesting concepts that play an important role in customising the behaviour of entities. *Context resolvers* are attached to any entity and can be used to define conditions that have to be met in order to access the entity. One can for example use this valuable feature in a one-to-many hyperlink. Context resolvers can then be attached to each hyperlink target in order to specify the context in which a specific hyperlink target is visible. This means that we can adapt the resolution of a hyperlink to a specific context. A *property* is a key and value tuple that can be defined in order to customise an entity's behaviour for a specific system. Note that each entity can be associated with multiple properties.

RSL further supports user rights management where access rights can be defined at the entity level. Figure 2.6 highlights the user rights management offered by the RSL metamodel. RSL distinguishes between two kinds of users; individuals and groups. A group of users contains individuals or other groups of users. Access permissions for each entity are defined by using the associations **AccessibleTo** and **InaccessibleTo**. Furthermore, the user rights management component of RSL takes into account that each user might probably have different preferences, for example for the visualisation of different entities. Hence, each user might be associated with a number of preferences.

Last but not least, RSL takes the initiative by supporting the resolution of so-called *overlapping hyperlinks* at the metamodel level. An overlapping hyperlink occurs when parts of selectors of a resource are

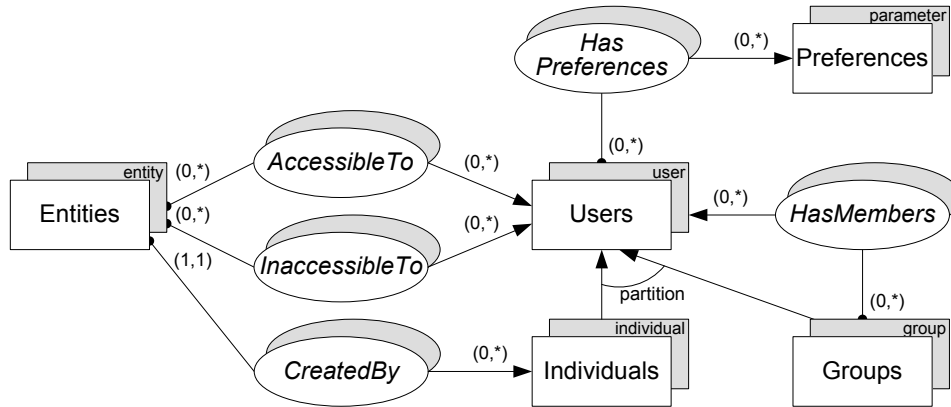


Figure 2.6: RSL user management component, based on [124].

overlapping. Usually, an overlapping hyperlink causes a *hyperlink resolution* problem since it is not clear which anchor/selector should be activated. XLink supports the creation of overlapping hyperlinks, but to the best of our knowledge, RSL is the only link model that provides a mechanism for the resolution of overlapping hyperlinks. Figure 2.7 illustrates the RSL layer component which takes responsibility for managing and resolving overlapping hyperlinks. Two main concepts have been introduced in this component that deal with the problem. The **Layers** collection maintains the different overlapping selectors of a resource by defining different layers overlapping selector can be associated with. Hence, every overlapping selector has to be on a different layer. Each resource can have multiple ordered layers. The **Active Layers** sub-collection of the **Layers** collection defines the currently active layers. Layers can be activated or deactivated depending on the selectors we want to access. It is out of the scope of this thesis to provide a full description of the RSL metamodel but all the details about the RSL hypermedia metamodel can be found in [124].

Over the last decade, the RSL metamodel has proven its flexibility, generality and extensibility and served as a basis for the implementation of many hypermedia solutions. RSL served as a basis for the implementation of the iServer cross-media platform [123, 124]. iServer has then been used in a variety of projects for physical-digital information integration and in particular for the implementation of the iPaper interactive paper framework [107, 120]. Based on the iPaper framework, bi- and multidirectional hyperlinks could be realised between printed papers and web pages. Furthermore, iServer has been used to build a semantic file

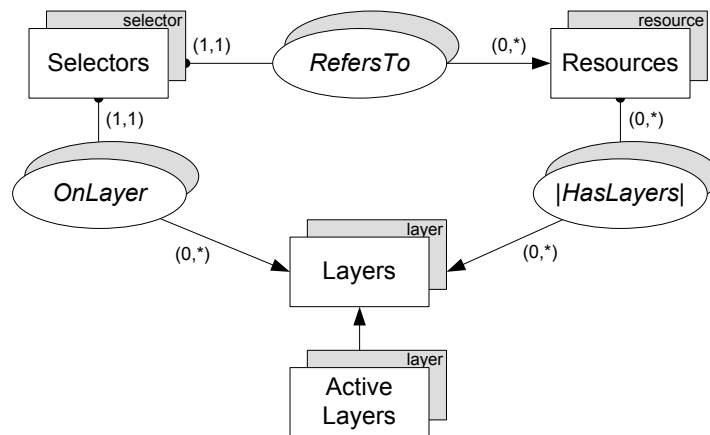


Figure 2.7: RSL layer component, based on [124].

system to overcome the classical hierarchical file management [29]. Last but not least, RSL has been used as a basis for a metamodel for fluid cross-media document formats [133].

2.5 Linking Features in Document Formats

In this section, we present a review of the hyperlink support in some existing document formats. As shown in Table 2.1, we have selected the most prominent and widely-used document formats. As mentioned in Section 2.3, early document formats such as GML, tnt [56] and Scribe did not support the linking to resources outside of a document but only supported intra-document references. Nevertheless, it is worth mentioning that Scribe and GML support the division of a document into smaller sub-documents. This feature can be seen as a limited form of outgoing hyperlinks to external documents where the main document contains commands to embed the smaller sub-documents when needed.

Unidirectional and embedded hyperlinks were supported in SGML, a descendant of the GML document format. SGML was developed to be a markup metalanguage that can be used to define the specification of other markup languages. The SGML metalanguage was used to create HTML which is used to create structured web documents. HTML hyperlinks are technically limited by the possibilities of the SGML metalanguage. HTML hyperlinks form a basic building block of the document itself. It offers simple typed and embedded unidirectional hyperlinks

Format	Hyperlink Type	Supported Target Resources
SGML	embedded unidirectional	web resources, entire third-party documents
HTML	embedded unidirectional	web resources, entire third-party documents
RTF	embedded unidirectional	web resources, entire third-party documents
L^AT_EX	embedded unidirectional	web resources, entire third-party documents
PDF	embedded unidirectional	web resources, entire third-party documents, parts of PDF documents
XML	external uni-, bi- and multi- directional	web resources, entire third-party documents, parts of XML-based documents
XHTML	embedded unidirectional	web resources, entire third-party documents
DocBook	embedded unidirectional	web resources, entire third-party documents, parts of other DocBook documents
OOXML	embedded unidirectional	web resources, entire third-party documents, parts of other OOXML documents
OpenDocument	embedded unidirectional	web resources, entire third-party documents
EPUB	embedded unidirectional	web resources, entire third-party documents
IBA	embedded unidirectional	web resources, entire third-party documents

Table 2.1: Supported link models in existing document formats

which are defined via the `link` and `a` tags. In its latest HTML5 revision, there is the possibility to enrich hyperlinks with extra metadata via HTML microdata [127] or RDFa [129]. With HTML hyperlinks, we can address any web resource and therefore link to any entire third-party document such as PDF or OOXML documents. When an HTML hyperlink has an entire third-party document such as a PowerPoint, PDF or Word document as a target, the web browser invokes a specific plug-in to visualise the document based on the third-party document's media type (e.g. `application/pdf` or `application/msword`). Even though HTML hyperlinks have been criticised by many researchers [106, 84, 122] for being embedded and unidirectional, there is no doubt that they were instrumental in the success of the most dominant and successful hyper-media system, the World Wide Web.

The Extensible Markup Language (XML) [60] is a simplified variant of the SGML metalanguage for the definition and creation of special-purpose markup languages. The XML metalanguage is the most widely known and used metalanguage. Various document formats have been developed using XML such as OOXML and XHTML [114]. The XML metalanguage does not offer a direct mechanism for creating hyperlinks. XLink and XPointer expressions [45] can be used to link parts of XML-based documents with each other. As mentioned in Section 2.4.1, XLink and XPointer expressions only support the linking to XML documents.

The XHTML markup language is an XML-based reformulation of HTML. The only difference between HTML and XHTML is that XHTML has to be well-formed in order to be parsed using the standard XML parsers. The fact that all features of HTML 4.01 were mapped to XHTML implies that there are no differences in the linking features offered by both document formats. Therefore, also XHTML only supports embedded unidirectional hyperlinks to web resources or entire third-party documents.

Besides the support for linking to web documents, many desktop document viewers such as Microsoft Word offer some linking features in their proprietary document formats (e.g. Rich Text Format (RTF) [132]). Moreover, most publishing standards such as L^AT_EX[86], PDF, EPUB and DocBook [138] provide different linking features. Most of these document formats support simple forms of linking to entire third-party documents and web resources.

RTF is a proprietary document format developed by Microsoft Corporation. It is used for presentation and cross-platform interchange of text documents. The support for hyperlinks has been introduced in early revisions of RTF. Users are able to create bookmarks within the document itself and link to these bookmarks from other places in the document. Moreover, embedded and unidirectional hyperlinks to web resources and entire external documents can be defined in a document.

L^AT_EX, which has been used to write this dissertation, is a markup language that is suitable for producing scientific documents, letters and presentations. The original L^AT_EX document format specification has no support for hyperlinks. However, over the years L^AT_EX has been extended via various packages. One of these packages, called **hyperref**, allows the creation of embedded and unidirectional hyperlinks within L^AT_EX documents and supports hyperlinks to web resources as well as entire external third-party documents. In contrast to L^AT_EX, the PDF specification sup-

ports four different kinds of unidirectional and embedded hyperlinks. PDF hyperlinks either reference intra-document anchors via the *GoTo* actions, anchors in external PDF documents via the *GoToR* actions, anchors in an embedded PDF document via the *GoToE* actions or any web resource via the *URI* action.

The DocBook document format is a semantic markup language for writing structural documents such as technical and scientific documents. The latest revision of DocBook is an XML application but it neglects all the rich features of the XLink and XPointer standards. DocBook offers different possibilities to enrich documents with hyperlinks. An **anchor** tag can be used to create bookmarks in the document itself which will serve as targets for intra-document references. The **XLink:href** tag is used to create a hyperlink to an already defined anchor, web resource or external third-party document. The **XRef** tag is used for establishing more advanced linking features which leads to content reuse of the referenced part. Finally, the **olink** tag can be used for establishing hyperlinks across documents, where Internationalized Resource Identifier (IRI)-based [49] linking is inappropriate. Thereby, the olink can be used for establishing application-specific links in order to for example retrieve information from a database.

EPUB stands for the Electronic PUblication standard. EPUB is a recent open eBook document format developed in order to overcome many shortcomings that existed in its predecessor, the Open eBook Publication Structure (OEBPS) [1], and to profit from new web standards and technologies. The EPUB standard is nowadays one of the most acceptable and supported document formats in eBook readers such as the Sony eBook reader³ and the Eee reader⁴. Due to the fact that EPUB is based on XHTML, EPUB hyperlinks are embedded and unidirectional. They enable the addressing of any web resource and entire external third-party documents. Nevertheless, it is worth mentioning that EPUB offers a mechanism called the EPUB Canonical Fragment Identifier (epubcfi) [128] which allows the referencing to any position in an EPUB document, in a way similar to the XPointer.

The iBook Author document format (IBA) [98] is another recent proprietary document format developed by Apple Inc. for its devices and operating systems. The document can be authored using a tool with the

³<http://www.sony.co.uk/electronics/reader/t>

⁴https://www.asus.com/Eee-Family/Eee_Reader_DR900

same name, iBook Author⁵. IBA is mainly based on the EPUB standard with more customised widgets that offer more interactive features. The IBA format does not go beyond what is supported in the EPUB standard, and hence, it supports embedded and unidirectional hyperlinks to address web resources and entire external third-party documents.

Last but not least, the Office Open XML (OOXML) [50] and the Open Document Format for Office Applications (OpenDocument or ODF) [140] are recent document formats that resulted from applying XML technologies to open proprietary document formats (e.g. Microsoft Office documents). Unfortunately, only the simple linking features of the XML link model have been adopted in these document formats. OOXML is used for presenting word, spreadsheets and presentation documents produced by Microsoft Office applications. The primary goal of the OOXML standard is to facilitate the interoperability between multiple office applications on multiple platforms. As mentioned, OOXML neglects the advanced linking features that are already supported by XLink and only adopted the simple embedded unidirectional hyperlinks that were supported in office documents. OOXML hyperlinks are able to address web resources, anchors in the same document as well as anchors in other OOXML documents. Nevertheless, it worth mentioning that the latest OOXML standard does not specify how hyperlinks to parts of other OOXML documents should be realised. Like the OOXML format, the OpenDocument standard is capable of presenting word processing, spreadsheets and presentation documents. It also facilitates the extensibility and the interoperability between the different office applications on multiple platforms. OpenDocument hyperlinks are embedded and unidirectional, so they are able to address web resources and entire external third-party documents.

2.6 Hypermedia and Annotation Systems

As mentioned earlier, hypermedia and annotation systems are considered the main trend for enhancing the linking of documents. Hypermedia systems were developed to facilitate the linking across different media types (e.g. text, images or videos) and/or different document formats. The different hypermedia systems can be classified into two main categories. The first class of systems contains closed proprietary hypermedia

⁵<http://www.apple.com/ibooks-author>

systems. Closed hypermedia systems offer linking features to some media types or proprietary document formats with the major limitation that hyperlinks are embedded inside the linked objects. This limitation prevents the management and manipulation of hyperlink metadata. An example of such hypermedia systems is Guide [23]. The second class of hypermedia systems are open hypermedia systems or link services. Open hypermedia systems overcome the shortcomings of the closed hypermedia systems by enabling the management and manipulation of hyperlink metadata externally to the linked objects. Some open hypermedia systems offer linking features not only to their own proprietary document formats but also to other document formats such as HTML. Examples of these systems include Sun's Link Service [112] and Microcosm [73].

Annotations are in fact either metadata or content that is attached to a document or parts of a document [9, 8]. In both cases, an annotation can be seen as a hyperlink since it establishes an association between a resource (a document or parts of it) with an external resource in the form of notes, comments or formal metadata (RDF) [8, 126, 125]. Over the last decades, a variety of annotation tools have been developed for different domains and different purposes. For example, various systems have been developed to enhance collaborations and discussions on the Web [70, 69, 24, 25]. Furthermore, a number of annotation systems have been realised for enhancing the collaboration and discussion among scientists in different digital libraries [4, 5, 7, 6, 42, 55, 52]. In the following two subsections, we discuss linking features as well as limitations of existing link services and annotation tools. In Section 2.6.3, we present an idea for an extensible annotation and link service proposed by Signer and Norrie [125, 126] which inspired the design of our extensible link service architecture. After a critical discussion of the presented systems, in Section 2.8 we outline essential requirements for an ideal extensible link service as well as a comparison of existing annotation and link systems in the light of these requirements.

2.6.1 Open Hypermedia Systems

One of the most well-known open hypermedia systems is Intermedia [68] which was developed at Brown University's Institute for Research in Information and Scholarship (IRIS). The Intermedia link service is a multi-user hypermedia system for Unix environments which demonstrates various hypermedia features. Intermedia has an explicit layered architecture

that clearly separates between data, logic and presentation layers. In the data layer, a DBMS model is used to store the hyperlink metadata externally from the linked documents. The presentation layer enables users to author documents as well as their hyperlinks. Intermedia's presentation layer supports the authoring and visualisation of five different document formats. Users are able to create bidirectional hyperlinks across the supported document formats. While Intermedia supports the linking across five different document formats, it also shows a number of shortcomings. Even though Intermedia is based on a layered architecture, it is not evident how Intermedia can be extended to support additional document formats. Moreover, Intermedia was intended to be used as a complete authoring tool and not purely as a link service. This implies that any document format that wants to profit from Intermedia's linking features has to be visualised and authored within an Intermedia viewer.

The integration of third-party document viewers was one of the main goals when developing Sun's link service [112] and Microcosm [73]. Sun's link service is a pure link service providing users the ability to create bidirectional hyperlinks between different document formats. The major contribution of Sun's link service is its ability to integrate external third-party document viewers by providing a protocol to communicate with them. Thereby, users are able to create hyperlinks between different documents that are visualised with their own third-party document viewers. However, Sun's link service has a major shortcoming. The third-party document viewers' integration protocol comes in the form of a program library that has to be included in any external application in order to communicate with the link service. This means that third-party document viewers have to be rewritten in order to benefit from the features offered by Sun's link service.

The Microcosm link service supports the external linking to Microsoft applications such as Microsoft Word. Furthermore, Microcosm offers some other features including the dynamic linking of documents as well as generic hyperlinks. With Microcosm's generic hyperlinks, a user can for example create a hyperlink from a snippet of information in a document to other documents containing a specific word or set of words. Even though Microcosm supports the linking across multiple external applications, it is not evident how Microcosm can be extended to support emerging document formats and their own third-party document viewers.

With the wide acceptance of the Web as a public hypermedia system, the open hypermedia community tried to enhance the Web's simple linking mechanism that prevents the manipulation as well as the management of web hyperlinks. The embedded HTML hyperlinks prevented end users from enriching web documents with associations to external resources such as annotations and other documents. Thus, many open hypermedia systems such as Hyper-G [95] and Webvise [66] have been developed in order to augment web pages with new associations and annotations. Moreover, some existing open hypermedia systems such as Microcosm and Chimera [11] have been extended for the Web. Furthermore, as discussed earlier some XLink-based systems such as Goate [94], XLinkZilla [78] and Xspect [36] have enriched HTML documents with more advanced hyperlinks.

Chimera [11] is an open hypermedia system that was mainly developed to offer linking features for heterogeneous software development environments. It offers advanced linking features such as multidirectional hyperlinks for heterogeneous software objects in a distributed and multi-user context. Nevertheless, we present this link service in this context for two main reasons. First of all, Chimera has an interesting architecture which offers the linking features to different software development environments by treating each of them as a client for the link service's server. Second, as mentioned earlier, Chimera has been used to enrich the Web with external hyperlinks by considering the Web as a client of the Chimera server. In Chimera, augmented web pages with annotations and hyperlinks are either rendered in an applet or in an ordinary web browser. The same shortcomings mentioned for the aforementioned Sun's link service and Microcosm systems are also valid for Chimera.

Hyper-G and its successor, HyperWave [96], are other open hypermedia systems for web augmentation [21]. Both systems have adopted a different mechanism than Chimera in order to augment web documents. First, they do not directly augment HTML documents but they use the special Hypertext Format (HTF) document format. Users are able to interact and augment HTF documents by using a proprietary Hyper-G or HyperWave browser. Later, users can access and visualise the augmented documents by using the proprietary or an ordinary web browser. In order to visualise the documents in an ordinary web browser, a special gateway is used to translate the HTF documents into HTML documents. Unfortunately, Hyper-G and HyperWave are targeting HTML documents and

it is not evident how they might be used to enrich non-HTML document formats with external hyperlinks.

The Distributed Link Service [31] is another well-known system for web augmentation. DLS is based on Microcosm's link service which can seamlessly be integrated in the user's web browser. By using DLS, users are able to directly interact with the HTML documents visualised in the web browser. Users are further not only able to create and navigate hyperlinks but also customise the visualisation of the augmented hyperlinks in order to distinguish them from a document's original hyperlinks. As in Microcosm, users are able to create generic hyperlinks based on a specific word or a pattern.

2.6.2 Annotation Tools

Since the introduction of the World Wide Web, researchers have been trying to enrich its simple embedded unidirectional linking model with more advanced features. As mentioned earlier, some Web augmentation systems have been proposed as well as the XLink and XPointer standards. Furthermore, many research and commercial annotation tools have been developed to open the Web for third-party annotations and associations to external web documents. It is worth mentioning that *most of these annotation tools did not offer an explicit feature to create hyperlinks between parts of web documents*. Nevertheless, their offered annotation features are very beneficial for web users and their collaboration. WebAnn [24] and CoNote [44] were early Web annotation tools that were mainly used for collaborative annotations and discussions on the Web. CoNote has been developed to facilitate the communication among students and their teachers using shared annotations and discussions. Using CoNote, teachers and students were able to have discussions within lecture documents. Annotations and discussions in CoNote were supported at document or section level. WebAnn has gone beyond CoNote by supporting fine-grained annotations of web pages, enabling annotations of a section, paragraph or even a word in a web page. It is worth mentioning that WeAann has been embedded in Microsoft Internet Explorer.

With the Web 2.0 movement, where users have become producers as well as consumers of information, the increased collaboration among web users motivated the World Wide Web Consortium (W3C)⁶ to create

⁶<https://www.w3.org>

the Annotea standard [83]. Annotea is the most well-known RDF-based standard to enhance collaboration on the Web via shared web annotations and bookmarks. These annotations can be notes, explanations or comments that are externally attached to a web page and are augmented with RDF metadata. Annotea uses XPointer expressions to address specific parts of a web page. A number of annotation tools have been implemented based on the Annotea standard, including the W3C's Amaya web browser or the Annozilla⁷ extension for Firefox. It is worth mentioning that various other business annotation tools have been developed including Diigo⁸ and Annotary⁹ which offer more or less the same features offered by Annotea-based tools.

MADCOW [20] is another annotation tool that enables the opening of web documents to third-party annotations and associations. MADCOW uses a client-server architecture where the client has been realised as a plug-in for Microsoft Internet Explorer. MADCOW overcomes Annotea-based tools by offering the possibility to annotate richer media types such images and videos. Moreover, it is worth mentioning that MADCOW has been used to offer annotation features to documents in a digital library [19]. The same criticism mentioned for Annotea-based tools is also valid for MADCOW since it only adopted the simple annotation concepts (e.g. notes or comments).

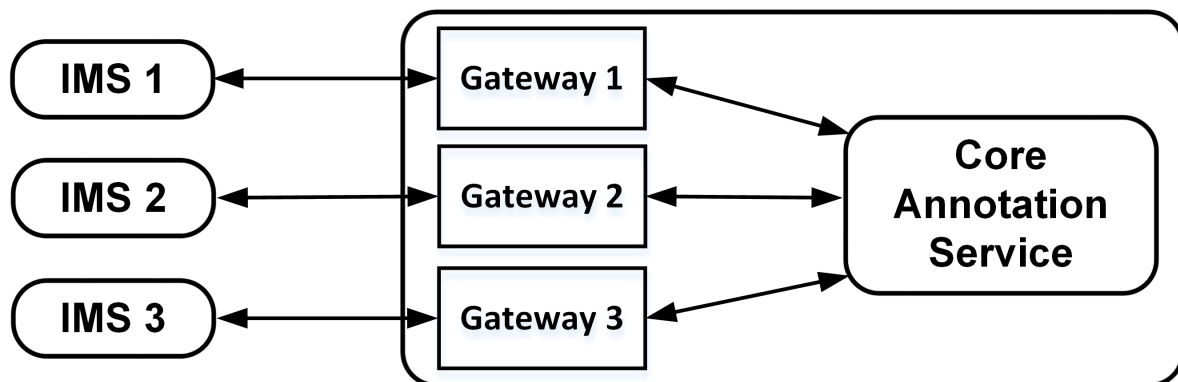


Figure 2.8: General architecture of the Flexible Annotation Service Tool (FAST), based on [6].

Whereas many built-in annotation components have been integrated in specific digital libraries in order to promote collaboration and discus-

⁷<http://annozilla.mozdev.org>

⁸<https://www.diigo.com>

⁹<https://annotary.com>

sions among researchers, the Flexible Annotation Service Tool (FAST) [6] has been implemented to be a stand-alone annotation service in order to offer its features and services to multiple digital libraries. Like Annotea-based tools or MADCOW, FAST offers simple annotation features without cross-document linking. Nevertheless, FAST has a very promising extensible architecture that enables the integration of external digital libraries that are managed using different information management systems (IMSs) as illustrated in Figure 2.8. The architecture has two main components; the core annotation service and a number of interfaces (gateways). Each gateway is connected to a different digital library (i.e. IMS) and ensures that the digital library gets access to the core annotation service offered by FAST. Thereby, in order to integrate a new digital library in the system, a new gateway has to be developed.

2.6.3 Extensible Architecture for Annotation and Link Services

The fact that most existing annotation and link services are not flexible and extensible to support the linking and annotation to existing and emerging media types was a main motivation for the proposal of the general architecture for open cross-media annotation and link services [125, 126] shown in Figure 2.9. Similar to the FAST architecture, its basic idea is the separation of linking and annotation functionality from the annotated media. Hence, the architecture makes a clear separation between the core link model and the user interface. An annotation and link service that is based on the proposed architecture knows how to deal with the core link model and is extensible to support other media types. The proposed link model architecture is based on the RSL metamodel presented in Section 2.4.2 which is flexible and extensible to support various media types. For any new media type to be supported, a “*data plug-in*” extending the RSL metamodel has to be provided. The data plug-in for a specific media type must provide the definition of its logical structure by extending the RSL **Resource** and must further define how to create selectors within its structure by extending the RSL **Selector**.

To the best of our knowledge, the cross-media annotation and link architecture by Signer and Norrie was the first approach to introduce the concept of *extensibility on the visualisation layer* of a link service. As mentioned before, when a link or annotation service is extended to support a new media type, also the user interface has to be extended to

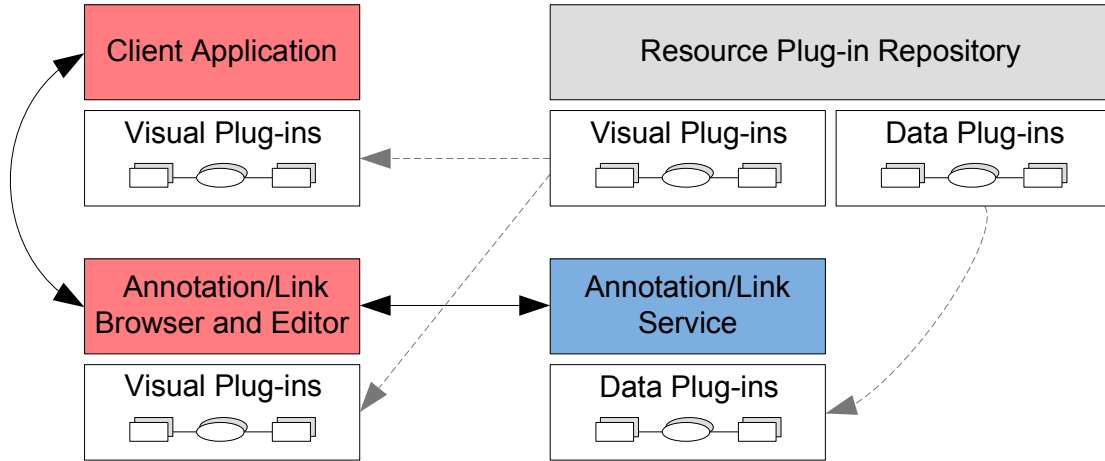


Figure 2.9: General architecture for open cross-media annotation and link services, based on [125, 126].

support the visualisation of the new media type. The extensibility on the visualisation layer proposes to use *visual plug-ins* in order to avoid a re-implementation and deployment of a link service’s entire user interface. A visual plug-in for a specific media type has two main purposes. First it should render a specific resource of the media type as well as its defined selectors (anchors). Second, it should provide functionality to create and delete resources as well as selectors.

The proposed cross-media annotation and link architecture further takes into account that end users use different third-party applications for rendering different media types. Thereby, the proposed architecture supports the linking to media types visualised via their third-party applications. In order to integrate a third-party (client) application in a link service, a visual plug-in extending its user interface as well as a data plug-in extending the RSL metamodel must be provided. The visual plug-in of a client application should communicate with the link service about hyperlinks and selectors to be created and highlighted. The visual plug-in must further provide visual handles to enable users to create, edit and delete selectors. Last but not least, an end user should be able to dynamically install the different plug-ins (i.e. data and visual plug-ins) in order to extend the link service and support a new media type to be visualised in the link service user interface or a corresponding third-party application.

2.7 Discussion

Most existing document formats have a simple embedded unidirectional link model. The majority of them pay attention to hyperlinks to web resources, neglecting the variety of other document formats. With the exception of the XML linking language, it seems that more recent document formats mimic the simple embedded hyperlink features supported by early document formats (e.g. SGML and HTML). Even though the advent of XML and XLink was a major step towards advanced linking on the Web, most recent document formats that are based on the XML standard (e.g. OpenDocument and OOXML) neglected the rich linking features offered by the XLink standard. All these reasons together limit the possibilities for cross-document linking where hyperlinks should be created between snippets of information in different document formats.

Existing annotation tools as well as link services do not solve the problem of cross-document linking, despite the fact that they were mainly developed to allow users to associate and link documents to external resources. Unfortunately, the development of most existing link services and tools has been motivated by limitations of their predecessors or new features to be supported rather than a clear understanding of the needs and requirement of end users. For example, the development of Sun's link service and Microcosm was motivated by the lack of a link service that supports the integration of third-party document viewers. The development of web augmentation open hypermedia systems (e.g. Hyper-G, DLS and Chimera) as well as XLink-based implementations (e.g. XLinkZilla and Goate) was motivated by the lack of advanced external hyperlinks on the Web.

The fact that most link services neglect the multitude of existing third-party document viewers forces users who are willing to benefit from a link service to abandon their preferred third-party document viewers. A link service should not only support the linking to its supported document formats but it should also provide a mechanism that allows the linking to documents visualised in their third-party document viewers. However, this mechanism should be different from Sun's link service where third-party document viewers have to be rewritten in order to benefit from the link service's features.

The support of a fixed set of document formats in a link service further does not give users the necessary flexibility in linking their preferred document formats. Nowadays, users deal with a variety of document

formats (e.g. HTML, PDF, PowerPoint, L^AT_EX and EPUB) and they will probably deal with emerging document formats that will support more features than existing document formats. Therefore, a link service must not only support the linking to existing document formats but also to emerging new document formats. This asks for a link service that is not only extensible on the link model level but also on the visual (user interface) level. In the next section, we outline six important requirements that should be fulfilled by an ideal link service in order to be flexible and extensible enough to support existing as well as emerging document formats and document viewers.

2.8 Towards an Ideal Cross-Document Link Service

2.8.1 Requirements for an Ideal Link Service

By carefully analysing the shortcomings of existing annotation and link services, we derived a number of fundamental requirements for an ideal extensible cross-document link service. We strongly believe that these requirements are the most essential requirements for integrating existing and emerging document formats as well as third-party document viewers with an extensible link service. This implies that it is still possible to have other requirements for a cross-document link service which are relevant for other features (e.g. collaboration or scalability) that are out of the scope of this thesis. In the following, we discuss each of the requirements in detail.

R1: Flexible and Extensible Link Service Architecture Most annotation and link services are not based on explicit link models. They often contain a mixture of conceptual and technical hard-coded link models. Other tools which are based on link standards are limited by the particular shortcomings of some of these standards such as the ones of the XLink standard discussed earlier. The fact that document formats have different document models means that anchors and selectors can be defined and addressed in different ways. For example, an anchor in a tree document model can be defined via an XPointer-like expression while an anchor in a text-only linear document model might be defined by its start and end indices. This challenge asks for a flexible and extensible link model. Thereby, flexibility means that the model should deal with

the different document formats supported by the link service whereas extensibility deals with emerging document formats to be supported at a later stage. Moreover, we strongly believe that the user interface of a link service must be extensible too as proposed by Signer and Norrie [126]. Most existing annotation and link services have to be redeployed every time a new media type has to be supported. Ideally, the user interface of a link service with an extensible link model will not have to be redeployed in order to support a new document format and the manipulation of the corresponding link anchors.

R2: Support for Multiple Document Formats This requirement can be divided into two sub-requirements: *a link service should be able to support existing document formats* and *should be able to deal with emerging document formats*. A link service should not be limited to a specific document format or a predefined set of document formats. An extensible link service should offer a simple mechanism to allow third-party developers or users to integrate additional document formats. The redeployment of the entire link service should not be an option since it requires the intervention of the link service deployer. The link service should benefit from the new dynamic extensibility techniques applied in applications and software development tools such as the Eclipse IDE¹⁰, L^AT_EX editors (e.g. TEXnicCenter¹¹) or the Google Chrome web browser.

R3: Easy Integration of Third-Party Document Viewers Supporting third-party document viewers should be taken into account in any successful link service. Otherwise, the link service must provide the authoring/editing of third-party documents and be appealing enough to convince users to no longer use their preferred third-party document viewers. This is not practical as users might want to continue using the document viewers they are familiar with. Nowadays, most proprietary third-party document viewers come along with their own Software Development Kit (SDK) or Application Programming Interface (API) (e.g. Foxit Reader Plug-in SDK¹², Microsoft's Office Developer Tools¹³ or the Acrobat Reader plug-in architecture SDK¹⁴) in order to be extended on demand to support some extra functionality. An extensible link service should benefit from a third-party document viewer's extensibility rather than forcing third-party document viewer vendors to rewrite their

¹⁰<https://eclipse.org>

¹¹<http://www.texniccenter.org>

¹²<http://www.foxitsoftware.com>

¹³<http://msdn.microsoft.com/en-us/library/jj620922.aspx>

¹⁴<http://www.adobe.com/devnet/acrobat.html>

applications. Plug-ins or add-ins can be implemented for these third-party document viewers in order to provide visual handles for creating and editing anchors in the supported document format. Furthermore, the link service should provide mechanisms to communicate with third-party plug-ins about newly created anchors as well as hyperlinks to be followed or highlighted.

R4: Flexible Communication Channels The support for third-party document viewers asks for communication across different protocols. The APIs and SDKs of some third-party document viewers limit their plug-ins or add-ins to a specific set of communication protocols. For example, Google’s Chrome Extension API¹⁵ and extensions for other web browsers only support WebSocket communication with third-party document viewers whereas TCP sockets are the default communication protocol for third-party desktop document viewers. An extensible link service should support the multitude of existing communication protocols since otherwise some third-party document viewers might not be integrated with the link service.

R5: Customisable Link Service The previous requirements imply that the different link service components are extensible via plug-ins extending the link model, visual plug-ins or third-party document viewers plug-ins. It is not practical to push users to install all plug-ins at once given that they might not use most of the supported document formats or third-party document viewers. Therefore, end users should be able to customise their link services by installing only the document formats that are really needed. Customisability of the link service means that the link service is extensible on demand. The L^AT_EX environment which has been used to write this thesis is a good example for on-demand extensibility via various packages to support extra functionality. The Eclipse IDE¹⁶ and Google’s Chrome web browser are two other examples, where different plug-ins are installed when needed. Extensibility on demand not only saves disk space but also increases the performance and memory efficiency of a tool. In order to successfully support on-demand customisation, the availability of the plug-ins should be ensured via a central online plug-in repository.

R6: Plug-in Versioning Different document format specifications are often updated to support new features. Moreover, third-party document viewers are normally updated with new features to either support

¹⁵<https://developer.chrome.com/extensions>

¹⁶<http://www.eclipse.org>

the new document format specification or to support new features in the application itself. Therefore, new versions of (some) plug-ins for a given document format are expected to be published. The link service should deal with this evolution by providing a mechanism for updating existing document format plug-ins.

2.8.2 A Comparison between Existing Annotation and Linking Systems

In Table 2.2 we present a comparison of a number of existing link services and annotation tools presented earlier in the light of the six requirements for an ideal cross-document link service. Each requirement is mapped to one dimension (column) in the comparison table, except for the first and second requirements. The first requirement is mapped to the two ‘extensible link model’ and ‘extensible user interface’ dimensions. The second requirement is mapped to the two ‘cross-document linking’ and ‘emerging documents’ dimensions. The former evaluates whether a system supports cross-document linking between multiple existing document formats, while the latter evaluates whether a system is extensible and might support emerging document formats. In the table, we use the ✓ symbol to illustrate that a feature is supported whereas the (✓) symbol means that there is only limited support for a given feature or the feature is supported with some major drawbacks.

As shown in the table, existing linking systems are far away from the ideal cross-document link service. None of the existing systems support the seamless integration of existing third-party document viewers or flexible communication channels. Furthermore, none of them supports the dynamic extensibility or the configuration of supported document formats or third-party document viewers.

Intermedia did not take into account any document viewer. Thereby, hyperlinks cannot be established from Intermedia documents to parts of other external third-party documents. Even though Intermedia is based on a layered architecture, it is not evident how the architecture can be extended to support more document formats on the level of the link model or on the user interface level. Similar to Intermedia, Sun’s link service has a monolithic user interface which requires a redeployment whenever a new document format has to be supported. As already discussed, the

Link Service	Extensible Link Model	Extensible User Interface	Cross-Document Linking	Emerging Documents	External Applications	Flexible Channels	Customisable	Plug-in Versioning
Intermedia	✗	✗	✓	✗	✗	✗	✗	✗
Sun's Link Service	(✓)	✗	✓	(✓)	(✓)	✗	✗	✗
Microcosm	✗	✗	✓	✗	(✓)	✗	✗	✗
Chimera	(✓)	✗	(✓)	(✓)	(✓)	✗	✗	✗
Hyper-G	✗	✗	(✓)	✗	✗	✗	✗	✗
DLS	✗	✗	(✓)	✗	✗	✗	✗	✗
Annotea-based Solutions	✗	✗	✗	✗	✗	✗	✗	✗
MADCOW	(✓)	✗	✗	✗	✗	✗	✗	✗
FAST	✗	✗	✗	✗	(✓)	✗	✗	✗
CoNote	✗	✗	✗	✗	✗	✗	✗	✗
WebAnn	✗	✗	✗	✗	✗	✗	✗	✗
Goate	✗	✗	(✓)	✗	✗	✗	✗	✗
XLinkProxy	✗	✗	(✓)	✗	✗	✗	✗	✗
Xspect	✗	✗	(✓)	✗	✗	✗	✗	✗
iServer	✓	✗	✓	(✓)	(✓)	✗	✗	✗

Table 2.2: Comparison of existing link services and annotation tools

mechanism for integrating third-party document viewers in Sun's link service asks for changes to the core of third-party document viewers. In contrast to Intermedia, Microcosm has a monolithic architecture which requires a redeployment whenever a new document format has to be integrated with the link service.

Chimera, Hyper-G and DLS have only augmented web pages with advanced hyperlinks. In contrast to Hyper-G and DLS that have been developed as monolithic components, third-party document viewers can benefit from Chimera's features since it treats external applications as clients for its server. We therefore believe that Chimera's link model is extensible to support other document formats besides HTML but not without redeployment.

As mentioned earlier, annotation tools including Annotea-based solutions (e.g. Amaya web browser and Annozilla extension for Firefox), WebAnn, CoNote, MADCOW and FAST adopt the simple annotation concepts (e.g. notes or comments) and do not support the creation of hyperlinks between existing content. We believe that even if the linking of existing content is supported and the extensibility is addressed,

these solutions are limited to the features offered by XLink. It is worth mentioning that it seems that MADCOW's underlying annotation/link model is extensible via redeployment since it offers the possibility to attach additional types of external resources (e.g. images and videos) to the annotated content. Furthermore, the integration of new digital libraries with the FAST system can be considered as a feature for integrating third-party document viewers.

The main goal of XLink-based systems such as Goate, XLinkProxy and Xspect was to enrich web documents with the advanced linking features of XLink. Therefore, these systems do not offer linking features to other existing document formats. Furthermore, it is not evident how the architecture of these systems can be extended to support other exiting document formats or third-party document viewers.

Last but not least, iServer which is based on the RSL metamodel offered external bi- and multidirectional hyperlinks to various multimedia types such as videos, web pages and printed papers. The flexibility offered by its link model allows the integration of other multimedia types and document formats with a redeployment and configuration of the system. However, iServer has a monolithic user interface which implies that it needs a redeployment and configuration whenever a new media type has to be supported. Even though iServer supports the linking between printed papers and web pages that are visualised via Firefox web browser, it does not offer an extensible mechanism for integrating other third-party document viewers.

2.9 Summary

In this chapter we presented the background and state of art of document linking approaches. We started by providing a brief history of hypermedia systems and the evolution of document linking approaches that have been started with the visionary idea of the Memex in 1945. We have then discussed XLink and RSL as well as their existing implementations. We have further discussed the linking features supported by existing document formats and highlighted some of their limitations. We then presented existing link services and annotation tools and discussed their architectures as well as their mechanisms offered to support cross-document linking in existing document formats. After critically discussing the existing approaches for cross-document linking, we have

defined six fundamental requirements that should be taken into account by any ideal cross-document link service in order to integrate existing and emerging document formats as well as third-party document viewers. In the light of these six requirements, we finally compared existing link services and annotation tools.

3

User Behaviour in Associating Information

In this chapter we explore the user behaviour in associating information across different physical and digital documents and address the current lack of empirical studies in this domain. The study provides insights into the strategies and systems that users apply to associate information across documents. In addition, it reveals the limitations of current practices and allows us to formulate suggestions to improve information association across documents. The findings also enable us to identify a set of design implications for the development of future cross-document linking solutions.

3.1 Context

As explained in the previous chapter, the development of most existing linking systems has been solely motivated by limitations of their predecessors or new features to be supported rather than a clear understanding of the needs and requirements of end users. We believe that the development of an extensible and successful cross-document linking solution requires insights into user expectations regarding cross-document link-

ing solutions and their functionality. These insights can be gained from studying users' current information associating behaviour. We will investigate information association mechanisms adopted by users in order to associate information within and across documents as well as their appreciation and criticism of existing mechanisms. While this thesis mainly focuses on the support of cross-document associations between digital documents, the study presented in this chapter also considers user behaviour in associating information in the physical space and across physical and digital documents. This enables us to compare the user behaviour in relation to both digital and physical media and to gain a deeper understanding of the best practices from the physical world. As such, the exploration provides insights into how to best support information linking and integration across both digital and physical media. These insights are also valuable for the general research on Cross-Media Information Spaces and Architectures (CISA) in the WISE research lab¹ which investigates conceptual models and architectures for integrating information across media spaces.

3.2 Literature Review

To the best of our knowledge, there are no existing studies that are mainly aiming to understand knowledge workers' behaviour in associating information within and across digital as well as physical documents. Nevertheless, there have been many different studies in different domains that revealed some general interesting findings regarding this matter. In the digital world, some of the studies have stated that system folders which are used to organise documents are a mean to associate "entire" documents. Besides folders, previous studies revealed that users use annotations in order to create associations between parts of documents. In the physical world, the filing and piling organisational strategies [90] are also used by users to organise relevant documents which can be considered as a kind of associating information.

Previous studies in Human Computer Interaction (HCI) that aimed to understand and analyse a user's reading and writing activities have stated that users perform a *cross documents referencing* task in order to integrate and associate information from one or multiple documents. According to Adler et al. [2], a cross-document referencing task forms

¹<http://wise.vub.ac.be/content/cross-media-information-spaces-and-architectures>

about 26% of the whole reading and writing activity tasks. The work done by O'Hara et al. [109, 110] revealed that readers of digital or physical documents tend to use different kinds of annotations (e.g. marginal notes or line markers) in order to make references between documents. In her work "*Toward an Ecology of Hypertext Annotation*" [93], Marshall stated that some user annotations in physical books are an emulation of some *hypertext patterns* such as creating and referring to anchors within the books themselves. Readers, for example, create highlighted or line-marked anchors in a printed book in order to refer to them from another part of the same book. Readers also tend to use annotations to explicitly make references to entire chapters or sections in the same book. Readers also mimic the creation of hypertext anchors by grouping some book content with an annotation or a highlight in order to refer to it from another part with the same book.

The literature within the Personal Information Management domain is mainly focusing on how people organise and retrieve information artefacts. Despite the fact that Whittaker et al. [141] have noted the lack of empirical research in PIM, there are some interesting findings that are relevant to the context of our research. Jones et al. [81] stated that folders are used to summarise, organise and *associate information* that is relevant for a specific user task (e.g. a planned project or some course material). Another study by Boardmann et al. [18] revealed that some users use a consistent folder naming convention to relate resources to each other. According to the same study, users create so-called *overlapping folders*, folders with the same name, using different tools (e.g. Outlook and file system) in order to organise resources that are related to the same production activity. Users might, for example, create a folder named *marriage* in an email client to store all emails concerning their marriage plan and at the same time create another system folder with the same name to store other related documents (e.g. wedding photos).

3.3 Methodology

In order to obtain robust as well as compelling evidence, we have chosen a qualitative approach using a multi-case design to explore different users' cross-document information association behaviour [144, 77]. The study relies on a mixed methods approach consisting of an online survey combined with interviews with participants of the online survey. The parti-

cipants in our research were informed that the collected data would be used for scientific research as well as scientific publications. Furthermore, we ensured the participants that their data would be treated confidentially and would be fully anonymised if used in any future publications.

3.3.1 Data Collection

3.3.1.1 Online Survey

For the purpose of this study we designed an online questionnaire which is included in Appendix A. The questionnaire focuses on investigating whether users associate information in both physical and digital media, which mechanisms are used by users to associate information, why they create these associations and how end users appreciate or criticise the mechanisms they currently use to define these associations. The online survey allowed us collect data from a much larger number of participants than it would have been feasible through alternative methods such as observations and think aloud experiments [144]. Furthermore, an experimental setup would cause a conflict with the purpose of this study which is to identify if, why and how people are creating associations between documents in everyday settings. In a survey, users should be able to freely report about their previous activities in associating information.

In order to identify reasons or barriers for associating information, we first needed to find out if users had ever “felt the need to associate information” in a particular manner (further referred to as a scenario). A negative reply to this question suggests that it is rather unlikely that they have engaged in this type of association activity, whereas a positive reply indicates that users are likely to have engaged in associating information or experienced an inability to do so. In the case of a negative reply, the survey immediately moved on to the next scenario. In case of a positive answer, participants were asked to provide more information about their behaviour in associating information or the difficulties and barriers that prevented them from creating these associations.

The survey contained both open-ended questions and quantitative questions using a 5-point Likert scale (e.g. questions enquiring the frequency of associating information). The open-ended questions enable participants to freely describe their previous activities in associating information within and across documents. It is worth mentioning that some of the survey questions investigate the information association mechan-

isms that have been already revealed by previous research, including the use of digital folders and annotations. The survey was conducted using the LimeSurvey² online tool and contained four groups of questions:

1. Questions related to a participant's demographic information such as gender, country of residence, education and age;
2. Questions concerning a participant's self-reported behaviour in associating information in the physical space. The questions differentiate between two main scenarios
 - Associating information within a **single physical** document (**SP**) (e.g. between two different sections or chapters in a printed book)
 - Associating information across **multiple physical** documents (**MP**). An annotation in a printed document that declares the existence of a relationship to (parts of) another printed document is an example for this scenario;
3. Questions related to a participant's self-reported behaviour in associating information within and across digital documents. The questions differentiate between three main scenarios:
 - Associating information in a **single digital** document (**SD**);
 - Associating information across **multiple digital** documents of the **same** document type (**MDS**) (e.g. between two PDF documents);
 - Associating information across **multiple digital** documents of **different** document types (**MDD**) (e.g. between a Word document and a PDF document).
4. Questions regarding a participant's self-reported behaviour in associating information across **digital** and **physical** documents (**DP**).

Please note that the scenario names and their abbreviations are extensively used in the rest of this chapter. Some general examples of associating information in all the different scenarios are illustrated in Figure 3.1.

²<https://www.limesurvey.org>

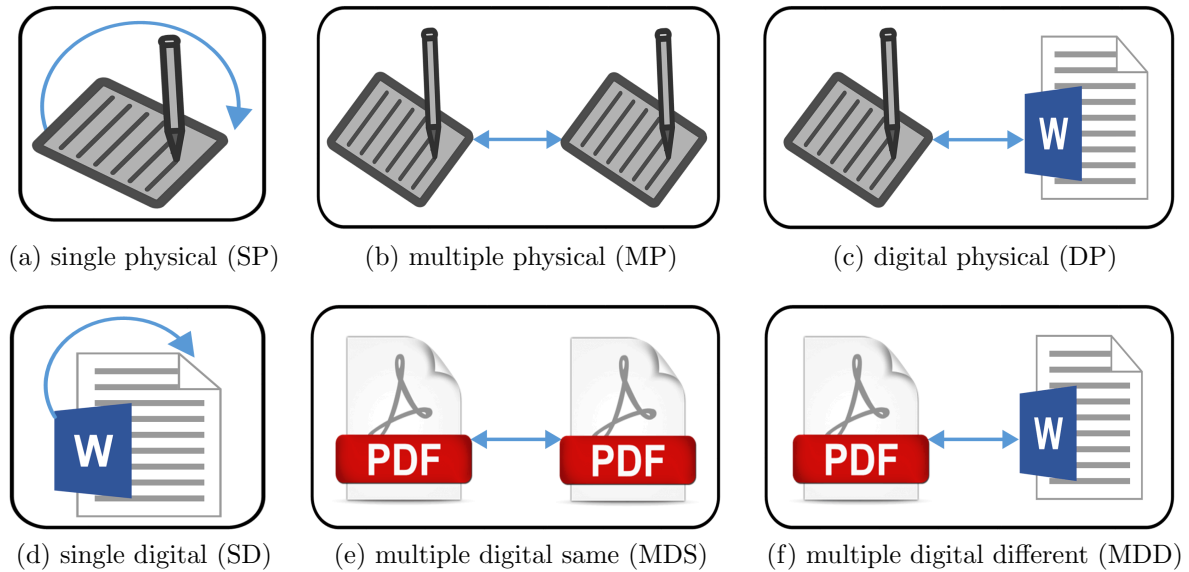


Figure 3.1: General examples of associating information in the different scenarios

At the end of the survey, the participants were invited to upload screenshots or images of associations they have created in the past. In total, we have received 15 images. Finally, the participants were also invited to provide their email address if they were willing to extend their participation in the form of an interview. As an incentive, participants who provided their email address could win a 30 EUR Amazon voucher. The email addresses were kept confidential and were only used to make arrangements for setting up the interviews.

3.3.1.2 Interviews

In addition to the online survey, we interviewed some of the survey participants in order to further investigate their answers to gain detailed insights about their information association behaviour. We used unstructured open questions to obtain descriptions of interesting mechanisms for associating information or clarifications of vague answers given during the survey. The interviews were performed either face-to-face or via Skype and lasted 15 minutes on average. Permission for recordings and their further use for research and publication purposes was obtained prior to the interviews. In addition to the recordings, the researchers also collected notes during the interviews. The interviews and notes were transcribed in a Word document to facilitate the data analysis.

3.3.2 Population

The link to our online survey has been internationally distributed to researchers via email through various mailing lists. Given the focus of our study, we choose to recruit participants from a population of researchers as they represent a group of knowledge workers who frequently use documents and can be expected to engage in document linking. It is worth mentioning that other user groups such as secretaries could also be considered as knowledge workers since they frequently use documents. However, we believe that researchers are more engaged in associating information especially when reading and writing scientific articles. In total, 238 people completed the survey. Our sample includes Master's students ($n=23$), PhD students ($n=169$) and researchers holding a PhD degree ($n=46$). The 238 participants consisted of 82 female and 156 male participants, with ages ranging from 21 to 60 years. While 97 participants have provided us their email addresses, we have only chosen 12 participants for the interviews.

3.3.3 Data Analysis

The collected quantitative data was analysed using descriptive statistics, while the qualitative data was analysed using an informal coding. During the qualitative analysis, the written comments of participants, the notes of the interviewer as well as the received images and screenshots were all taken into account. First, the qualitative data was carefully checked in search of common association mechanisms used by the participants. Based on this assessment, a list of association mechanisms was compiled for every scenario presented to the participants (i.e. SP, MP, SD, MDS, MDD, DP). Second, we identified the characteristics and limitations of each mechanism and calculated how many participants claimed to have used it.

3.4 Results

Our study shows that *most knowledge workers are either occasionally or frequently associating information across documents* as indicated in Figure 3.2. Users are associating information during both reading and writing activities which confirms the findings presented in [2, 109, 81]. As

shown in Table 3.2, users use different association mechanisms to associate information in the different scenarios. Please note that in Table 3.2, association mechanisms that have been already identified in previous research are shown in normal font, whereas the mechanisms we discovered are emphasised in italics.

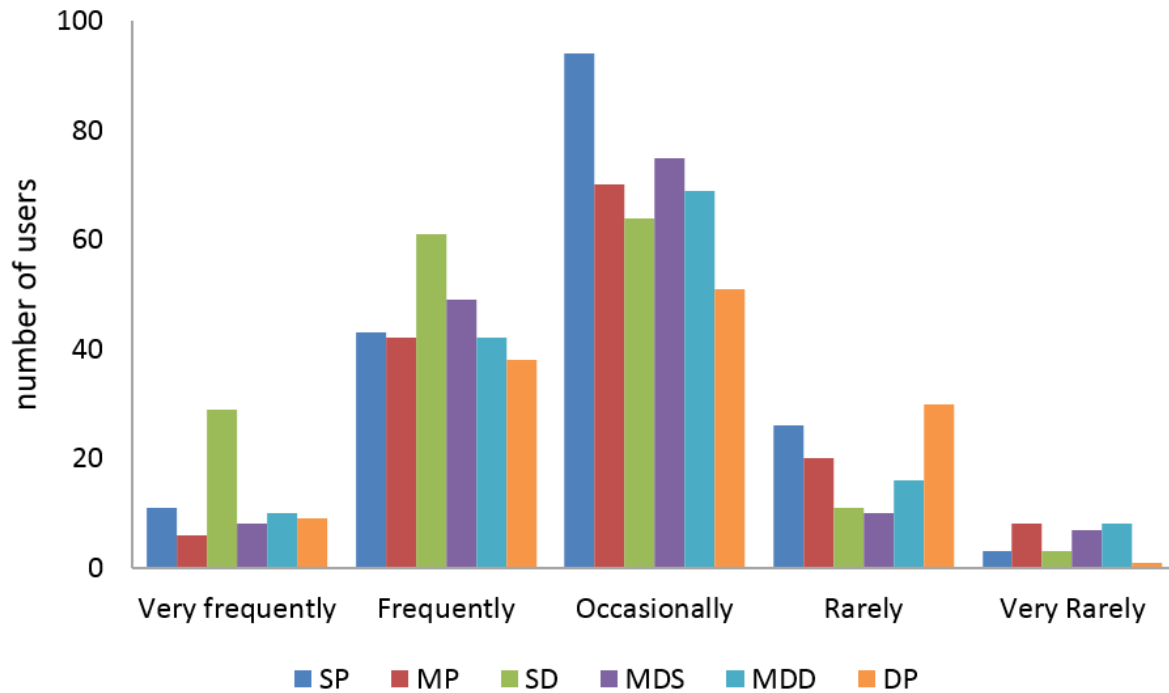


Figure 3.2: Frequency of associating information in the different scenarios

We begin by presenting the relative size of the subsample of participants who associate information in a particular scenario (see Table 3.1). Similar to the structure of the questionnaire, the findings presented in this section are divided into distinctive groups based on the different information association scenarios. We continue by providing a more detailed description of the different association mechanisms as well as a statistical description of how often each association mechanism has been adopted by users as shown in Table 3.2. Please note that the percentages presented in Table 3.2 are relative to the number of participants who associate information in the corresponding scenario (see Table 3.1). For instance, out of the 74.3% who associate information in scenario SP, 11.8% use line and arrow drawings in order to associate information. We close this section with a detailed explanation about the characteristics of different types of associations identified by users as a result of the different association mechanisms. In addition, we also address common

Scenario	Size of the subsample relative to the total number of participants
SP	74.3%
MP	61.3%
SD	70.5%
MDS	62.6%
MDD	60.9%
DP	54.2%

Table 3.1: Relative size of the subsample of participants who associate information in a particular scenario

work practices and the appreciation or criticism of users with regard to each mechanism. This detailed information is used in Section 3.5 to formulate a number of design implications for a suitable information and cross-document linking solution.

Mechanism		SP	MP	SD	MDS	MDD	DP
Annotations & highlights	B1	24.2%	28.7%	35.8%	30.2%	28.9%	21.7%
	B2	30.5%	29.4%	27.6%	20.1%	19.3%	24.8%
	U1	31%	22.6%	35%	45.6%	48.2%	25.5%
	U2	31%	42.4%	34.3%	33.5%	34.4%	35.6%
<i>Line and arrow drawings</i>		11.8%	—	—	—	—	—
<i>Common symbols</i>		2.2%	—	—	—	—	—
<i>Separate documents</i>		2.2%	7.5%	3.3%	12.7%	12.4%	4.6%
Physical folders		—	3.4%	—	—	—	—
Post-it notes		1.1%	1.3%	—	—	—	0.7%
<i>Physical counterparts</i>		—	—	2.8%	3.3%	3.4%	3.8%
<i>Copy & paste</i>		—	—	2.2%	1.3%	1.3%	—
Digital folders		—	—	—	63%	66%	—
<i>External applications</i>		—	—	—	3.3%	2.7%	—
<i>Writing physical parts into a digital document</i>		—	—	—	—	—	2.3%
<i>Scanning physical documents</i>		—	—	—	—	—	3.1%

Table 3.2: Association mechanisms used in the different information association scenarios

3.4.1 Associating Information in Physical Documents

As shown in Table 3.1, many participants (74.3%) indicated that they associate information in a single physical document and among them 11 participants (6.2%) reported doing this very frequently (Figure 3.2). In the case of MP, 61.3% of the participants indicated that they have faced situations where they had to associate information across two or more physical documents, with only 6 participants (4.1%) of them doing this very frequently. This difference in terms of frequency can be caused by various factors. As discussed later, some users apply simple and trivial mechanisms to associate information in SP that cannot be applied in MP (e.g. drawing a simple line between the associated parts). Furthermore, sometimes the closeness of the associated parts in SP (e.g. information in the same page) helps users to easily associate information in this scenario.

Our investigation revealed that participants are applying different mechanisms to create associations between pieces of information in both physical scenarios. Most of the participants use annotations (e.g. comments, arrows or highlights) in order to explicitly associate information. This seems to confirm the findings presented in [110, 93, 109]. The use of annotations as well as highlights enables participants to *establish associations at any level of granularity* since participants are able to highlight and annotate any part of a document. Annotations and highlights also yields to different types of *bidirectional* and *unidirectional* associations.

Our study revealed two main types of bidirectional associations resulting from using annotations and highlights as illustrated in Figure 3.3. For the first type (B1), the bidirectionality is established via highlighting the different parts as well as writing annotations that include references to each other next to all these parts. For the second type (B2), participants only write annotations next to each part without highlighting. In a bidirectional association, participants write references to all the linked (associated) information parts to help them later in information refinding tasks. The first type of bidirectional association (B1) mimics the creation of hypertext anchors since participants explicitly highlight the exact text anchors to be linked with each other. This finding confirms Marshall's findings [93] that participants create highlighted or line-marked anchors in a printed book in order to refer to them from another part in the same document.

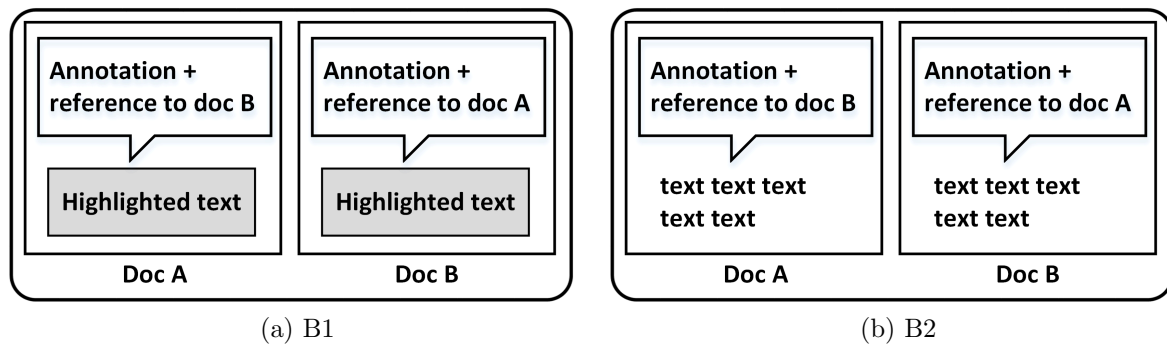


Figure 3.3: Two main types of bidirectional associations using annotations and highlights

In the single document scenario (SP), bidirectional associations of type B1 are used by 24.2% of the participants, whereas they are used by 28.7% of the participants in the MP scenario. The second type of bidirectional associations (B2) is used by 30.5% of the participants in the single document scenario and by 29.4% of the participants in the different documents scenario. We also discovered other interesting bidirectional association patterns. One important pattern emerges in a single document scenario where the parts are close to each other (e.g. on the same page). In these instances, 11.8% of the participants indicated that they would normally draw arrows or lines between the associated parts. Moreover, four participants (2.2%) indicated that they use common symbols or numbers in order to match the different pieces in the single document scenario. The use of line drawings as well as common symbols as association mechanisms illustrates the flexibility in creating associations in the single physical document scenario.

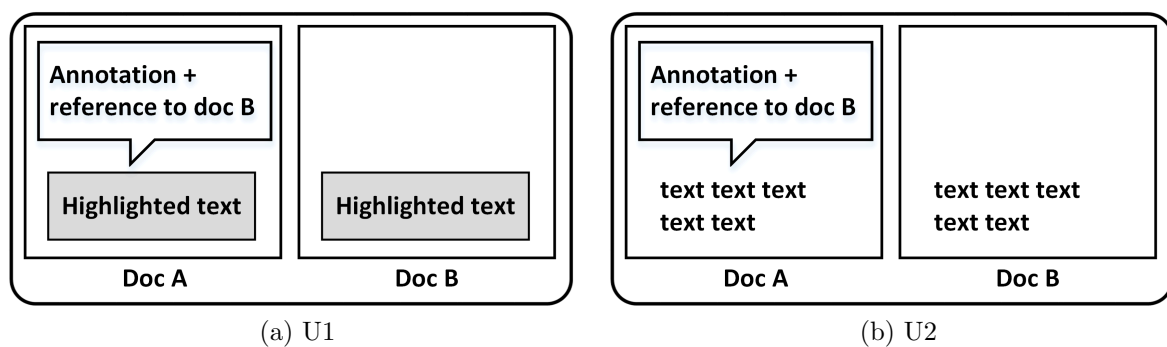


Figure 3.4: Two main types of unidirectional associations using annotations and highlights

We also discovered that many participants are creating *unidirectional associations* between the intended pieces of information. Two main types of unidirectional associations resulting from using annotations and highlights were discovered. As illustrated in Figure 3.4, in the first type (U1), the unidirectionality is established by highlighting the different parts and by writing annotations next to only one of the parts. In the second type (U2), participants do not highlight any part but rather write annotations next to one of the parts. Unidirectional associations of type U1 are used by 31% of the participants in scenario SP and by 22.6% of the participants in scenario MP. The U2 associations are used by 31% of the participants in the scenario SP and by 42.4% of the participants in scenario MP. The collected data from the survey and the interviews shows that participants prefer to have bidirectional associations due to the advantages in re-finding the related (linked) parts.

Another interesting finding is that numerous participants tend to use a *separate document* in order to explicitly create associations, especially in scenario MP. In the single physical document scenario (SP), 2.2% of the participants indicated that they use a separate document (digital or physical) in order to note, summarise or copy and paste the different parts. One participant wrote: *“I use an external document to link the parts by citing the paragraph numbers”*. On the other hand, in scenario MP, 7.5% of the participants are using a separate document to explicitly indicate the association. *“I start a new document and write something about these two different documents”*, one participant wrote. Another participant answered: *“I make a list of things to remember and link them together on a new blank document”*. Another participant who uses a separate digital document mentioned: *“I use my computer to link the specific parts of different physical documents by citing the paragraph numbers”*.

Other association mechanisms applied by participants include the use of physical folders, ring binders as well as paper stacks in order to associate information across multiple physical documents (MP), mentioned by 3.4% of the participants. These mechanisms result in the filing and piling organisational strategies that are well known in the PIM literature. It is out of the scope of this study to discuss these strategies but more information can be found in [90]. Other participants have indicated the use of post-it notes in order to associate information in both, scenario SP and MP. Last but not least, one participant (0.5%) in SP and two participants (1.3%) in MP said that they try to memorise the associ-

ations. One of them wrote: *“I just try to remember, and most likely I forget about the existence of the association”*.

3.4.2 Associating Information in Digital Documents

In general, most participants are applying numerous mechanisms to associate information in the different digital scenarios. Out of the participants who associate information, 7.7% in scenario SD, 3.3% in scenario MDS and 3.4% in scenario MDD have explicitly indicated their inability to create associations between the intended parts. In general, *the lack of a suitable linking tool* as well as *the effort in creating associations* between the intended parts are the main reasons that prevented various participants from creating associations. *“I do not have a convenient mean to do so”*, one participant commented. Other participants wrote: *“Too time intensive to make visual connections on most document types”*, *“I do not know how”* and *“In fact all annotation tools available in my software are not proper enough to establish the association”*.

Associations across different digital documents are created at *different levels of granularity*. As illustrated in Figure 3.5, they are created to link different parts of the documents (c), parts of a document with the entire other documents (b) or entire documents (a). In the context of scenario MDS, 75.1% of the participants who associate information associate information due to the relationship between the different parts of the documents (c). In the same scenario, 57% of the participants associate information based on the relationship between parts of a document with entire other documents (b), whereas 48.3% of the participants create associations as relationships between entire documents (a). In the context of scenario MDD, 46.2% of the participants who associate information associate information due to the relationship between entire documents (a). In the same scenario, 80% of the participants are motivated by the relationship that exists between parts of different documents (c), whereas 61.3% of the participants are motivated by the relationship that exists between parts of a document and entire other documents (b).

3.4.2.1 Single Digital Document

In scenario SD, 70.5% of the participants indicated that they have faced situations where they associated information within a single document. While 82.3% of those participants who associate information in a single

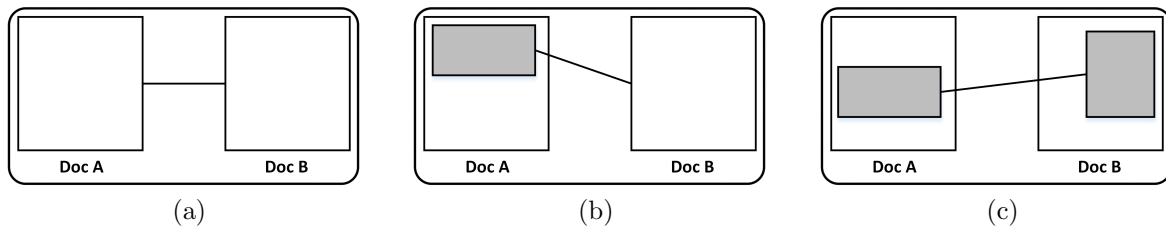


Figure 3.5: Associations between digital documents are created to link entire documents (a), parts of a document with the entire other documents (b) or different parts of the documents (c)

document use annotations as well as highlights to explicitly create the associations, another 10% of the participants have adopted various other interesting mechanisms to create the associations between the different parts. As mentioned before, the remaining 7.7% of the participants are not able to create the intended associations.

The annotation features provided by the different document viewers and applications (e.g. Adobe Acrobat Reader or Foxit Reader annotation features) have enabled the majority of the participants to create the associations using annotations and highlights not only in scenario SD but also in the other digital scenarios MDS and MDD. This is an interesting finding since it contradicts to some extent some findings of a previous study carried out by O'Hara et al. [109] which compared participants' reading activities of physical and digital documents. In their study published in early 1997 and conducted using Microsoft Word 6.0 for reading digital documents, the majority of participants did not prefer to annotate digital documents. According to O'Hara et al., the difficulty of annotating documents, the inflexibility of interaction techniques via mouse and keyboard and the changes to the original document (e.g. text underlining) were the main reasons that prevented participants from annotating digital documents. The contradiction between our finding and the previous study finding might be explained in different ways. In the study of O'Hara et. al., participants were obliged to use a specific "old" document viewer for reading documents. It is possible that some of the participants were not familiar with the annotation features offered by Microsoft Word 6.0. Furthermore, probably some participants who did not annotate documents in Microsoft Word 6.0 were annotating documents in other document viewers (e.g. Adobe Acrobat Reader). Last but not least, there is no doubt that some annotations of Microsoft Word (e.g. text underlining) will change the original document content. Never-

theless, nowadays most document viewers (e.g. Adobe Acrobat Reader) treat annotations as a separate layer on top of the original document content. Users are able to hide, display and print documents with or without their annotations.

By using the annotations and highlights mechanism to create associations, participants tend to create bidirectional and unidirectional associations between the different parts. Out of the 82.3% of the participants who use annotations and highlights, 35.8% create bidirectional associations by highlighting the different parts and writing annotations next to each of the parts (B1). 27.6% establish bidirectional associations of type B2. Unidirectional associations of type U2 are established by 34.3%, whereas U1 associations are used by 35% of the participants.

As mentioned before, apart from the use of the annotations and highlights, many interesting association mechanisms are used by the participants. As in the physical document scenarios, various users (3.3%) tend to use separate documents to summarise or copy and paste the associated parts. According to many participants, *the new document used to describe the different associations serves as a database of related information or a starting point to remember some content of the linked documents at a later stage*. Five participants (2.8%) mentioned that they *prefer to read printed versions* of the documents and associate information in physical counterparts. Two of them have explicitly indicated the flexibility in annotating physical documents compared to digital documents. Four participants (2.2%) said that they *copy one part and paste it next to the other*. One of the two participants wrote: *“I repeat a paragraph in the other part of the document”*. Three other participants (1.7%) indicated that they rely only on their memories to remember the association.

3.4.2.2 Different Documents of the Same Document Type

In this scenario, 62.6% of the participants tend to associate information. Most participants have adopted various mechanisms to create associations between pieces of information in different documents of the same document type. These mechanisms include but are not limited to storing the linked documents in the same digital folder, using annotations and using separate documents for describing the intended associations.

Participants are mainly using the digital folder mechanism (63%) in order to create associations between entire documents. Some of the participants use consistent naming schemas while storing documents in folders

as shown in Figure 3.7. This finding confirms the finding of [81] which stated that participants use folders in order to organise related information. It is worth mentioning that previous research has shown that traditional hierarchical folder structures are ineffective and cognitively demanding [54, 61, 14]. The study of Golemati et al. [61] demonstrated the profound problem of hierarchical folders in desktop environments where participants could not remember the location of their documents in 17% of the retrieval tasks.

As in the previous scenarios, unidirectional and bidirectional associations are resulting from the use of annotations and highlights. Unidirectional associations of type U2 are established by 33.5%. Figure 3.6 illustrates an association of type U2 created by one of the participants. The U1 unidirectional associations are applied by 45.6% of the participants. Bidirectional associations of type B1 are established by 30.2% of the participants, whereas B2 associations are created by 20.1% of the participants.

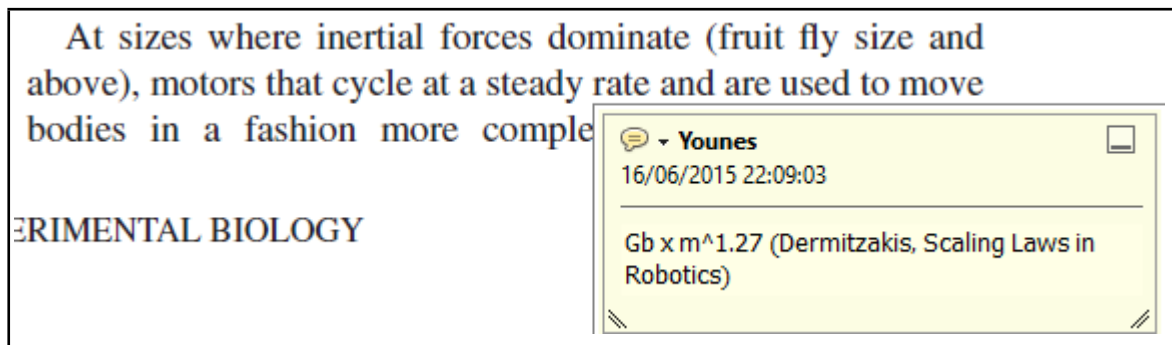


Figure 3.6: MDS: A U2 unidirectional association using an annotation

The other mechanisms (e.g. the use of separate documents) are applied by 22.6% of the participants. The use of a separate document to associate information is adopted by quite a large percentage of the participants with 12.7% of the participants using separate documents to summarise, describe or copy and paste the parts. *“I usually find it counterintuitive to annotate the different documents and prefer to create a separate document (Word outline) with comments, explanations to linked documents etc.”*, one participant explained. Another participant mentioned: *“Re-type or copy-paste the parts into a Word document and set the bibliography with APA style in each part and briefly describe how they are related”*. Two other participants wrote: *“I create another file that has the intended links, however, those tend to get lost and are not*

really used” and *“I usually create a third document divided into conceptual sections (one topic per section). I then copy-paste the relevant part of each document under the topic it refers to along with its reference in brackets”*. Two other participants (1.3%) have explicitly mentioned that, if possible, they copy one of the associated parts and paste it into the other associated document(s) (i.e. next to the other associated part).

As in the previous digital scenario (SD), the use of the physical counterpart mechanism is applied by five participants (3.3%). They prefer to associate information in the printed version of the intended documents. *“I prefer to work on printed material which I can also easily sort physically in addition to storing the files in the same folders”*, one justified their tactic. Three other participants (2%) rely on their memories to remember associations. According to them, over time they tend to forget their associations.

Five of the participants (3.3%) have indicated *the use of some external applications* (e.g. Evernote³, JabRef⁴ and Zotero⁵) to create associations between entire documents of the same digital type. The *bibliography reference managers* such as JabRef and Zotero are used by participants to make associations between entire documents in order to facilitate the creation of citations and bibliography while writing some scientific reports (e.g. journal or conference papers). One participant mentioned: *“I use my reference manager (Zotero) to indicate that one document is related to another. Related documents are available under the related tab on the document’s details panel”*. It is worth mentioning that most of the interviewees (12) did not indicate the use of bibliography reference managers in their answers to the online survey questions. Nevertheless, in the interviews, all of the interviewees have confirmed the use of these systems during the scientific writing activity. Further, *most of the interviewees did not consider the bibliography reference managers as association and linking tools*.

3.4.2.3 Different Documents of Different Types

In this scenario, 60.9% of the participants associate information across two or more documents. As in the previous scenario, storing associated documents in the same folder is the dominant mechanism adopted by

³<https://evernote.com>

⁴<http://jabref.sourceforge.net>

⁵<https://www.zotero.org>

66% of the participants. Figure 3.7 illustrates an annotated screenshot of a folder of one of the participants where linked documents are stored and consistently named.

Using annotations and highlights, unidirectional associations are established more than bidirectional associations. The U1 associations are created by 48.2% of the participants whereas U2 associations are established by 34.4% of the participants. 28.9% of the participants create B1 associations whereas B2 associations are created by 19.3% of the participants.

Another 20.8% of the participants are applying various other mechanisms similar to the aforementioned mechanisms in scenario MDS (e.g. the use of separate document, external applications or document physical counterparts). It is worth mentioning that those participants (20.8%) are a subset the participants (22.6%) who have mentioned the use of various association mechanisms other than the folder and annotations mechanisms in the previous scenario MDS. Some of them have mentioned various details about their mechanisms. One participant who uses the copy and paste mechanism mentioned: *“I grab and paste a screenshot of one document inside the other one (e.g. a screenshot of a PDF inside a Word or an Excel document)”*. Two participants who use the external application mechanism wrote: *“just putting some notes in Evernote and citing documents”* and *“create screenshots of relevant parts and use them in Evernote”*.

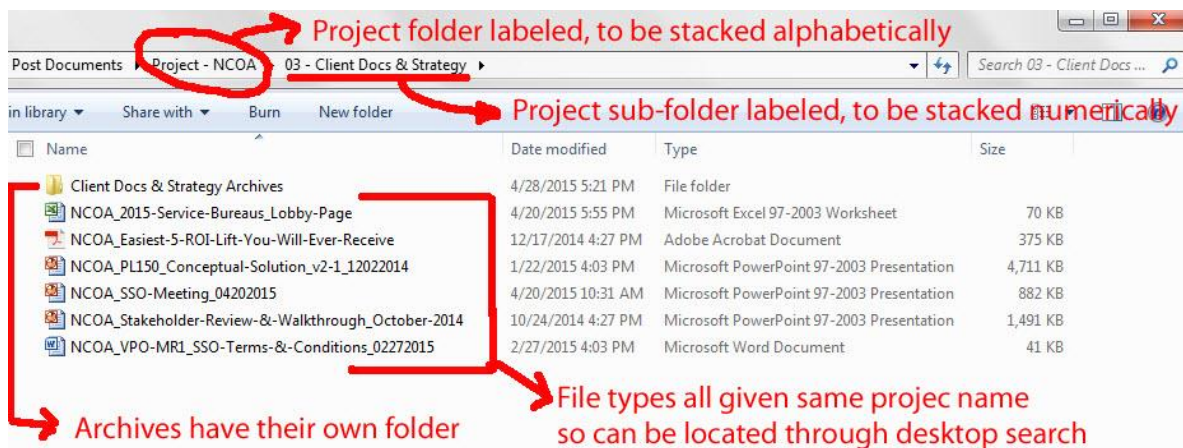


Figure 3.7: MDD: Linked documents are stored and consistently named in a folder

3.4.3 Associating Information Across Physical and Digital Documents

Almost half of the participants (54.2%) indicated that they have associated information across digital and physical documents. The collected data revealed that participants are using various mechanisms to associate information across digital and physical documents. Using annotations and highlights, participants tend to create either unidirectional or bidirectional associations between information across physical and digital media. Out of the 54.2% who associate information across physical and digital media, 21.7% of the participants tend to create bidirectional associations by highlighting the different parts and writing annotations next to each part (B1). Furthermore, bidirectional associations of type B2 are created by 24.8% of the participants. Unidirectional associations of type U1 are created by 25.5% of the participants, whereas unidirectional associations of type U2 are created by 35.6% of the participants.

Another 16.3% of the participants have declared the use of a wide range of mechanisms to associate the intended parts across physical and digital media. Some of them (3.8%) indicated that they tend to *print the digital document* to have the flexibility in annotating and archiving. Other users (2.3%) prefer to *write the information of the physical part into the digital document*. “*I type the part I need from the physical document in the digital one*”, one participant wrote. Another participant mentioned: “*I try to gather all relevant information in the digital world, and hence, I type the parts from the physical document in the digital document*”.

Other participants (4.6%) indicated the use of the separate document mechanism to associate information across physical and digital media. One of them uses a *separate physical document* whereas the rest use *separate digital documents*. The participant who uses the physical document mentioned: “*I make a list in a new physical document that lists all the names of the documents I should link together*”. One of the participants who uses a digital document explained: “*I create a summary file with the same title as the digital one*”.

Four other participants (3.1%) mentioned that they *scan the physical document or take a photo of it* using their smartphones. The scanned document or photo is then stored in the same digital folder as the digital document. “*I scan the physical document or page and store it in the same folder as the digital one, or paste it in the digital document whenever it*

is possible”, one participant explained. It is obvious that those four participants are also using the same folder strategy in combination with the scanning mechanism. One participant (0.7%) mentioned the use of post-it notes on the physical document in order to explicitly create associations.

3.4.4 A Deeper Look into the Information Association Mechanisms

Most of the information association mechanisms revealed by our user study are a result of different work practices. Moreover, each of them produces different types of associations (e.g. unidirectional or bidirectional). Furthermore, participants have different levels of satisfaction about their adopted association mechanisms. In this section, we discuss the nature of associations resulting from the different information association mechanisms. We further elaborate on the users’ satisfaction with their information association mechanisms used in digital scenarios (SD, MDS and MDD) as well as across digital and physical documents (DP). User issues and suggestions should help us to derive some design implications for the development of an ideal cross-document linking system. Finally, we discuss common work practices and how they shape the decisions regarding the use of a specific association mechanism.

3.4.4.1 Characteristics of the Associations

As mentioned earlier, each association mechanism establishes a different kind of association. We have already discussed some characteristics (bidirectionality and unidirectionality) of associations resulting from the use of annotations and highlights in the different scenarios. Two important aspects should be kept in mind while investigating the nature of an association; *the granularity of the associated parts* (e.g. fragments or entire documents) as well as *the attached references to the associated parts*. A bidirectional association enables a user to navigate from a source to a target document and vice versa. On the other hand, a unidirectional association only enables a user to navigate from a source to a target document. Some associations do not imply any traversal such as the associations created by the folder mechanism. In Table 3.3 we provide a summary of the characteristics of the resulting associations of the different association mechanisms.

Mechanism	Directionality			Associated information's granularity	
	Uni	Bi	Directed	Entire document level	Any level
Annotations & highlights	✓	✓	✓	✓	✓
Line & arrow drawings	✗	✓	✓	✗	✓
Common symbols	✓	✓	✓	✓	✓
Separate documents	✓	✓	✓	✓	✓
Physical folders	✗	✗	✗	✓	✗
Post-it notes	✓	✓	✓	✗	✓
Physical counterparts	✓	✓	✗	✓	✓
Copy & paste	✓	✓	✓	✗	✓
Digital folders	✗	✗	✗	✓	✗
External applications	✗	✗	✗	✓	✗
Writing physical parts into a digital document	✓	✗	✓	✗	✓
Scanning physical documents	✗	✗	✗	✓	✓

Table 3.3: General characteristics of the resulting associations

The use of the annotations and highlights mechanism enables users to create all types of associations at any level of granularity. Annotations written next to two linked parts in a bidirectional association should contain references to each other in order to enable a user to easily navigate between the linked documents. If annotations in one linked part do not contain reference(s) to the other part(s), the created association is categorised as a unidirectional association. The different document viewers have enabled users to write annotations at any place in a document (e.g. marginal annotations, annotations between the original document content or on top of a page). Users are further able to highlight almost any part of a document. Thereby, with annotations and highlights, users can associate information at word, paragraph, section and chapter level or even define associations between entire documents.

The drawing mechanism (i.e. lines and arrows), connecting information parts that are close to each other, produces bidirectional associations. Users are able to see all endpoints of the resulting associations. The fact that “most” of the drawings are attached to a single document limits the possibilities to create associations at some levels of granularity. With the drawing mechanism users can, for example, not establish associations between two chapters or two sections of different chapters in a document. As with drawings, the common symbols mechanism would normally produce bidirectional associations. In contrast to the drawing mechanism, common symbols support the linking at more levels of granularity. Users should be able to associate different chapters, sections or

even entire documents with the common symbols as long as they attach the right references to the linked parts.

The use of separate physical or digital documents for associating information enables the creation of unidirectional and bidirectional associations at any level of granularity. Some users write the correct references to all the linked documents which results in a bidirectional association. On the other hand, a unidirectional association can be established by neglecting the writing of a reference to one of the linked parts. Users can also establish associations at any level of granularity. Some users create associations between entire documents by summarising all the linked documents or writing only the titles of the linked documents. Others establish associations at a lower level of granularity by, for example, summarising or copying and pasting two important paragraphs from two different documents.

The use of the system folder as well as the bibliography reference managers enables users to create *undirected associations* between *entire documents*. The traversal between documents is not defined in both of the mechanisms. A system folder visualises its documents in a way (e.g. a list) that enables users to navigate to any document. Most bibliography reference managers enable the navigation not between documents but from a document to its system folder.

The associations resulting from physical archives as well as scanning mechanisms have more or less the same characteristics as the associations resulting from the use of system folders. The scanning of documents implies that the linked documents are stored in the same folder, whereas the system folders are an emulation of the physical archives. The use of the printed versions of digital documents (physical counterparts) mechanism to associate information implies the use of annotations or physical archives for associating information. Thereby, the associations resulting from this mechanism have more or less the same characteristics as the associations resulting from using annotations or physical archives. Last but not least, the associations resulting from the copy and paste, the writing of physical parts into digital documents and post-it notes mechanisms depend on the recording of the associated documents' references as well as the granularity of the information part that has been noted or copied (e.g. entire document or only a paragraph).

3.4.4.2 User Satisfaction with Used Association Mechanisms

Figure 3.8 illustrates the participants' satisfaction with the used mechanisms in the three different digital scenarios as well as across digital and physical documents. Please note that the percentages presented in Figure 3.8 are relative to the number of participants who associate information in the corresponding scenario (see Table 3.1). It is clear that many participants (57.7%) are satisfied with the mechanisms they use for associating information in a single document (SD). In contrast to the single digital document scenario, most participants are not happy or uncertain about the way they create associations in the other scenarios MDS, MDD and DP. In scenario MDS, 31.6% of the participants have indicated that they are not satisfied with the used mechanisms. In the same scenario, 32.2% of the participants are uncertain about the way they create associations. In scenario MDD, 40% and 28.3% are not satisfied and uncertain respectively. Finally, for scenario DP, 59.7% are not satisfied with the used mechanisms.

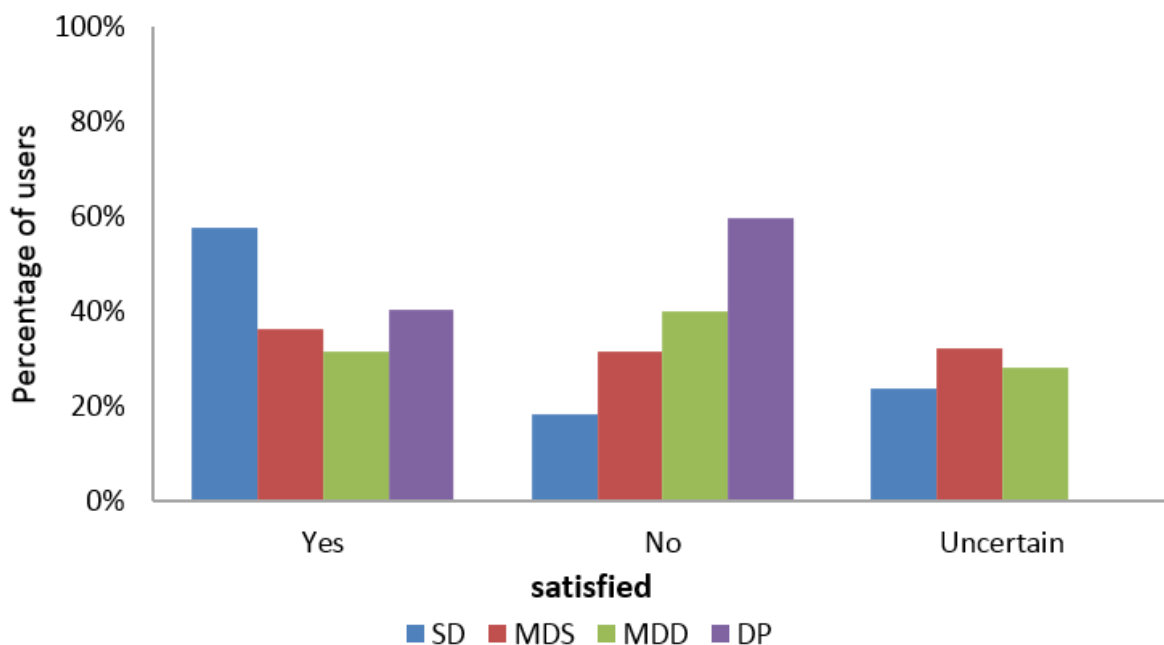


Figure 3.8: User satisfaction when associating information in digital documents and across digital and physical documents

Participants have given us numerous reasons for not being satisfied with their currently used association mechanisms. Some participants, especially those who reported the difficulty of creating associations in the different scenarios, have given us general complaints such as: *"I cannot*

create it”, “It is not efficient and mostly relies on my memory, which may fail in finding back the associations I have seen when I read the documents months or years before”, “I have to reread the documents again in order to remember the connection”, “It is not intuitive how to make the associations” and “It is hard to do it properly, no uniform tool exists”.

In the context of the different digital scenarios, most participants who use annotations and highlights to associate information complained about the *wasted time in creating and refinding the association* between the documents. Some of their complaints are: *“I have to read the annotation and then search for the related parts/documents. It would be nicer if I have something more visually like a link connecting the different parts to jump directly or pop-up related information from the other document”, “I still have to remember where I have written the annotation that expresses the relationship” and “It is time consuming to make a lot of annotations”.* One participant complained and suggested a better solution: *“It is not efficient if you have to write stuff down two times. Being able to make a direct link would be better”.*

Many of the participants who adopted the organisation of related documents with the system folders have given numerous reasons for not being satisfied with this mechanism. Some of their complaints were: *“common folder is not the most accurate”, “the structure [of the folders] is unclear, it is not organised well enough, causing confusion and waste of time” and “there might be a more efficient way to do it”.* Last but not least, participants who use separate documents as well as the copy and paste mechanisms were mainly concerned about the time wasted as well as the context of the association. *“Sometimes, in order to gather the linked information into a separate document, the information is simplified and it becomes difficult to understand the context, but also time consuming if you need to search and go back to the initial documents”,* one participant complained. Another participant complained: *“It requires a lot of extra work, lots of copying that results in redundancy, and most of the times it is hard to keep track of all the associations manually”.*

In the context of associating information across digital and physical documents, participants were mainly complaining about the time wasted in creating associations as well as retrieving and tracking of the linked documents. We think that most of the used association mechanisms in this context (DP scenario) require an effort from users, for example, creating annotations in the two different media (physical and digital) or scanning and archiving documents. The tracking of linked documents

in both media is further not an easy task. Some of the participants' complaints were: *"Printed documents do not tend to get lost or forgotten. Nevertheless, the annotations are lost or unreliable because I am not in the computer world to get help in finding them. Furthermore, path and name of the linked digital documents change, and one results deleting or updating annotations (if found) in the printed document"* and *"There is no easy way to retrieve the linked information"*.

3.4.4.3 The Need for a Linking Tool

After asking the survey participants about their issues with the used mechanisms, they were asked whether they feel the need for a tool or a mechanism to facilitate the linking (associating) of information in the different scenarios SD, MDS, MDD and DP. *Most of the survey participants (179) have indicated the need for a tool for associating information that easily supports the creation of associations between different digital document parts as well as the easy navigation between these document parts.*

In the context of the digital scenarios, 25.1% out of the 179 participants have shared their thoughts and suggestions for a suitable tool for associating information. 35 participants emphasised that a future tool must support the creation of hyperlinks. Some of them mentioned: *"It comes natural to me that such a system would resemble the mechanism of the Web"*, *"Conventional bookmarking or hyperlinking should do it"* and *"Hyperlinks seem to be an obvious choice [...]. Honestly, I do not know why this has not been done 20 years ago"*. Some participants were concerned about the usability of any future linking tool, *"It should be extremely quick and easy"*, *"A linking tool that is easy to learn"* and *"links to other document and jumping directly to the correct target paragraph"*.

Some participants were concerned about the inclusion of essential features in any future linking tool. Two of them were concerned about the integration of the linking mechanism in their workflow. The first participant mentioned: *"It would be dreamable if the linking mechanism would be available in the native document viewing program, and not that we are forced to use another application to link documents"*. The second participant wrote: *"It would be important to consider that the tool should work fine not only in the latest version of each software but also in the older versions"*. One of the participants who complained about the searching and refinding associations mentioned: *"[...] allowing me to link and then*

search for the associations". Other participants have given other suggestions such as: *"By creating an HTML page, in this page we can represent the name of all documents as a mindmap. Each part of this map (i.e. parent or child) will be linked to the appropriate document using a hyperlink mechanism. Our own notes can be added as comments near to the map parts"* and *"a tool that allows you to define the associations (e.g. cluster A; cluster B or cluster C) and show them in a different window next to the document, so that you can visualise the different elements in each cluster and go from one to the other without restrictions"*.

In the context of the DP scenario, 20% out of the 59.6% who are not satisfied with the used mechanisms have shared their thoughts and suggestions for a better linking mechanism across digital and physical documents. One of the participants who is familiar with the recent Anoto⁶ digital pen and paper technology suggested to exploit this technology in realising any future cross-media linking solutions. Other participants suggested the use of QR codes⁷ and RFID tags [139], *"by creating a link between them [means a physical and digital documents], for example indicating the position of the document in the physical space via RFID tags and in the digital space via a URI or the user's file system"*. Another participant suggested a totally different idea to *"automatically scanning physical documents and do text recognition on them. Then collect keywords from them to organise them on the computer together with the digital files"*.

3.4.4.4 User Work Practices

Most of the interviewees have mentioned the use of multiple documents while reading and writing not only in the physical space but also in the digital space. The use of *multiple screens* for multi-document viewing and associating information is a very common work practice for most of the interview participants. These findings are consistent with previous research findings [109]. According to the study participants, the simultaneous use of multiple documents enables them to easily annotate related information or summarise it in a separate document.

The annotation features of different document viewers are an important mean to establish associations by using annotations and highlights. Nevertheless, as complained by different participants, while users are able

⁶<http://www.anoto.com>

⁷<http://www.qrcode.com/en/index.html>

to create different kinds of annotations within their documents, they are not offered the necessary functionality to search, re-find or manage their annotations. According to various participants, most of the time they are forced to open their documents and search inside them to recall and find the previously created associations.

Another interesting work practice that we have discovered in our user study is the limited use of bibliography reference managers in reading activities. As mentioned before, participants are mainly using these applications for some scientific writing activities. We discovered that most of the users are not aware of the many interesting features (e.g. relate documents to each other) offered by these applications. Indeed, as most of the interviewees have indicated, these systems are not per se linking and association tools, but they can be seen as a layer on top of the system folder where users can group documents that reside in different system folders into one unified collection.

Last but not least, the advancements of the resolution of cameras embedded in smartphones enables various participants to scan and take photos of physical documents while associating information in the DP scenario. According to two interviewees, this practise does not require an effort in taking the photo, but it needs an effort for archiving and storing the taken photo. According to one of them, the required effort depends on the applications installed in the smartphone as well as the synchronisation of the smartphone and a user's main work station such as a desktop computer or a laptop.

3.5 Discussion and Design Implications

Our study revealed a number of interesting and important findings. It has shown that the activity of associating information forms an integral part of reading and writing activities of knowledge workers. Participants have adopted various mechanisms to associate information in both physical and digital spaces. They further associate information within and across different documents at any level of granularity. Study participants are associating information at the document, chapter, section, paragraph or even at word level.

In most of the investigated scenarios, users are mainly using annotations, folders and separate digital or physical documents to explicitly create the associations between documents. *Some of the association mech-*

anisms used in the physical space (SP and MP) are emulated with or similar to other mechanisms in the digital space (SD, MDS and MDD). The use of a physical folder mechanism (e.g. filing cabinets) in the physical space is, for example, emulated via the digital folder mechanism in the digital space. The use of separate documents to associate information is a common mechanism in both physical and digital media.

In addition, our study reveals that users tend to create different types of associations (i.e. bidirectional, unidirectional and undirected associations) between the associated parts. Nevertheless, regardless of the nature of the created associations, most users are not satisfied with their used mechanisms. Even though many hypermedia solutions exist in business and research, our study shows that *there is a lack of efficient and usable linking tools* that can be used by end users to associate information in the different scenarios. Most of the users have been forced to adopt different association mechanisms in order to overcome the lack of suitable linking tools. Based on their suggestions and the characteristics of their created associations, users do not only need systems that create associations between entire documents (e.g. system folder or bibliography reference managers), but also they ask for solutions that enable the creation of associations at any level of granularity. Furthermore, users need efficient association systems that go beyond the support of creating associations at different levels of granularity. An efficient association system should allow users to easily and efficiently organise, search and retrieve their created associations. Moreover, an efficient association system should take into account current user behaviour in associating information across documents. It can, for example, give users who use the separate document mechanism the option to automatically gather all the associated parts in a separate document. In the remainder of this section, we outline a number of important design implications for a future linking solution that we have derived from the collected data as well as from users' mentioned issues and suggestions.

DI1: Granularity of the associations: As mentioned earlier, the associations between documents exist at any level of granularity. A user should be able to establish “hyperlinks” at any level of granularity. The fact that document formats have different logical structures and document models (e.g. tree or linear structures) [57] should not create any barrier in supporting this feature in a future linking and association system. An association and linking application should rather enable different fragment identifiers (selectors or anchors) for the different document

formats. For example, a start and end index could be used for identifying selectors and anchors in plain text documents. In contrast, the XPointer [46] standard might be used to enable selections in some XML document formats such as OOXML or the OpenDocument format.

DI2: Bidirectional associations: As discussed before, many users prefer bidirectional associations over unidirectional associations. A future association and linking tool should not only enable the creation of bidirectional associations, but also allow users to seamlessly navigate from one source document to another. We believe that users who are used to create unidirectional or undirected associations will not complain about the support of bidirectional hyperlinks after they experience their advantages. Unidirectional hyperlinks are always criticised due to the inability to navigate in two directions [134, 122]. An update or delete operation on one endpoint of a unidirectional hyperlink will often produce a broken (dangling) hyperlink. Bidirectional hyperlinks are less exposed to such inconsistencies when updating or deleting any endpoint. This is due to the nature of bidirectional hyperlinks where an application should ensure the consistency of both endpoints.

DI3: Documents side by side: As discussed before, our study and a previous study [2] have both confirmed that knowledge workers need multi-document viewing and manipulation as part of their reading and writing activities. The use of multi-document viewing enables users to easily associate information across documents. It is worth mentioning that many systems have been built based on the side-by-side reading and annotating of documents such as [35, 34]. A future association and linking tool should enable the visualisation of documents side by side to give users the flexibility in creating “hyperlinks” between the visualised documents. A hyperlink between two documents that are visualised in a side-by-side manner can, for example, be created by simply using a drag-and-drop interaction.

DI4: Linking across physical and digital media: Information is available in different physical and digital media types. In fact paper and printed documents are still forming an important source of information. It is not sufficient to build applications that support the associating of information in digital media only but we should think about possibilities to support the information integration and association across physical and digital media [121]. As mentioned before, the interactive paper and digital pen seem promising in this context. A future linking application should overcome the limitation of existing cross-media applications by

supporting the seamless integration of printed materials and arbitrary digital media.

DI5: Management of the associations Giving the users the possibility to create associations between different documents should come along with the possibility to manage their associations. As discussed before, a future linking application should have an integral management component that helps user in managing and searching their associations. In fact, an optimal link management component should go beyond the simple search mechanism for associations. Instead, users should be able to filter and sort their associations based on many dimensions such as the types of the associated documents, context of associations or the time when they were created.

In the rest of this thesis, we take the initiative by presenting a cross-document link service that takes into account end user requirements for an ideal linking tool. Our link service addresses most of the aforementioned design implications for future linking systems. Moreover, our link service overcomes the shortcomings of existing link services by taking into account the fundamental requirements for an ideal link service presented in Section 2.8.

3.6 Threats to Validity

Many factors can jeopardise the validity of our user study. First of all, it is possible that the survey participants have different interpretations of what an association means depending on the applications they use or the activities they carry out. Furthermore, we have already discussed how some results of our study contradict with previous findings of O'Hara et al. [109] due to the advancement of document viewers. According to their findings, users did not like to annotate documents due to the difficulty of annotating documents and the inflexibility of interaction techniques. In contrast to their findings, we found that most of the users are using annotations to create associations between digital documents. We believe that existing applications might offer enhanced linking functionality and thereby future findings would then contradict with some of ours. Last but not least, our findings might only be generalised for the community of researchers. Other knowledge workers (e.g. secretaries) might have different interpretations of what an association means.

3.7 Summary

In this chapter we have presented a study that relies on a multi-case design approach for exploring user behaviour in associating information across digital and physical documents. We have collected data from 238 knowledge workers using an online survey. Furthermore, 12 participants of the online survey have been interviewed to gain detailed insights about their behaviour in associating information. Using the online survey, we have investigated user behaviour in associating information in six different scenarios. These scenarios are associating information within a single physical document (SP), associating information across multiple physical documents (MP), associating information in a single digital document (SD), associating information across multiple digital documents of the same document type (MDS), associating information across multiple digital documents of different document types (MDD) and associating information across digital and physical documents (DP). The collected quantitative data was analysed using descriptive statistics, whereas the qualitative data was analysed using informal coding.

Our study showed that most knowledge workers are associating information in the different information association scenarios. Information is associated more frequently in the single digital and physical scenarios (SP and SD). We have discovered twelve different association mechanisms applied by users in associating information in the different scenarios. The results showed that four out of the twelve different association mechanisms (i.e. digital folders, physical folders, separate documents and annotations & highlights) are used more frequently than the other mechanisms. We have discussed the nature of associations resulting from the twelve association mechanisms. For example, associations resulting from the annotations & highlights mechanism, are uni- and bidirectional. Furthermore, they associate information at any level of granularity. Our study further showed that many participants are not satisfied with their used mechanisms for associating information in the different scenarios. Most of the participants were concerned about the wasted time in creating and refinding their associations. A majority of the participants have further indicated the need for a tool for information association that supports the creation of associations as well as the navigation between the associated documents.

Based on the collected data and criticism of study participants about their used association mechanisms, we have derived and outlined a num-

ber of important design implications for a future linking solution. A future linking solution should enable users to establish hyperlinks at any level of granularity. Furthermore, it should allow users to create bidirectional hyperlinks and to seamlessly navigate these hyperlinks. In addition, a future linking solution should consider the side-by-side reading and visualisation of the linked documents and enable the linking across digital and physical spaces. Finally, users should be able to manage and retrieve their associations (hyperlinks).

4

A Dynamically Extensible Cross-Document Link Service

The survey presented in the previous chapter has illustrated the user need for a cross-document linking solution. In Chapter 2, we have already highlighted a number of limitations of existing linking solutions regarding the support of cross-document linking. Most existing linking solutions support the linking across a predefined set of document formats and are not extensible to support other existing or emerging document formats. Moreover, existing linking solutions are not extensible on the visual level (user interface). Last but not least, most existing approaches do not consider the support and integration of a user's preferred document viewers. Hence, users wishing to benefit from a specific link service are obliged to abandon their preferred document viewers.

In this chapter, we investigate an architecture for a cross-document link service that meets the user needs and overcomes the limitations of existing linking solutions. The presented link service architecture and its working prototype take into account the requirements for an ideal linking solution discussed earlier in Section 2.8, as well as the design implications outlined in Section 3.5. *Throughout the chapter, we motivate each design*

decision and indicate how the different link service components are used to realise a dynamically extensible cross-document link service. Note that whenever a design decision meets a requirement or a design implication discussed in Section 2.8 and 3.5, we refer to the corresponding requirement.

This chapter is structured as follows. In Section 4.1, we briefly elaborate on our proposed cross-document link service. Section 4.2 presents the general architecture of our link service. In Section 4.3, we outline the communication between the different components of the link service and in Section 4.4 we discuss the dynamic extensibility of our link service. In Section 4.5, we elaborate on the integration of document formats in the link service's link browser (user interface), whereas in Section 4.6 we discuss the integration of third-party document viewers with our link service. We critically discuss the presented link service in Section 4.7. Finally, Section 4.8 provides a summary of this chapter.

4.1 Proposed Solution

The presented cross-document link service is based on the general open cross-media annotation and link architecture presented in Section 2.6.3. Figure 4.1 illustrates a high-level conceptual schema of the presented link service. In brief, the link service contains six essential components: the link model, the link browser, the gateway, the communication component, the data storage and plug-in tracking components. The core link model is extensible via plug-ins to support arbitrary document formats (e.g. `DocFormat1`, `DocFormat2` and `DocFormat3` in Figure 4.1). The link browser is also extensible via plug-ins and is responsible for visualising the supported document formats (e.g. `DocFormat1` and `DocFormat2` in Figure 4.1). The gateway and communication components are essential for integrating document formats that are visualised using their own third-party document viewers (e.g. `DocFormat3` in Figure 4.1). The gateway component is also extensible via plug-ins to support arbitrary third-party document viewers. The plug-in tracking component is responsible for extending the link service via the different plug-ins by communicating with an online plug-in repository that is in charge of managing the different plug-ins. Furthermore, the plug-in tracking component is responsible for keeping track and managing the different plug-ins in the link service. The data storage component deals with the persistent storage of the selectors

and hyperlinks. It is worth mentioning that the used link model is capable to address any document via arbitrary resource identifiers (e.g. URI or URN). In other words, a user can use our link service to create hyperlinks between documents that are stored on their devices (e.g. desktops or smartphones), online (e.g. web pages) or in shared repositories. In the rest of this thesis, we discuss examples of creating hyperlinks between documents that are either stored locally (i.e. desktops) or online. We believe that the creation of hyperlinks in documents that are stored in shared repositories would raise many issues such as collaborative hyperlink editing, privacy and access rights that are out of the scope of this thesis. In the rest of this section, we further elaborate on our proposed solution that overcomes the limitations exposed by existing linking solutions.

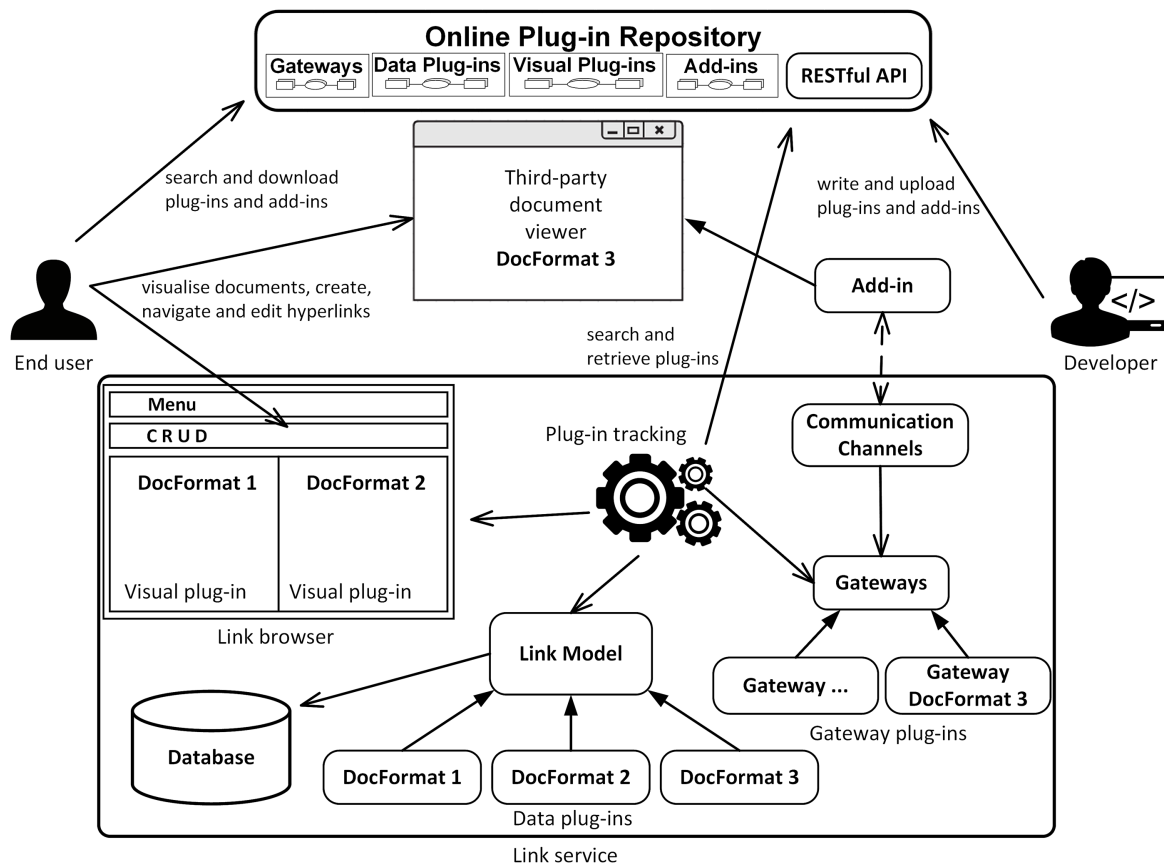


Figure 4.1: Conceptual schema of the presented link service

4.1.1 Link Browser

As suggested by Signer and Norrie [125, 126], our link service makes a clear separation between the link model (data level) and the link browser (visual level) in order to facilitate the necessary link service flexibility and extensibility. The link service provides end users with a link browser whose components are outlined in Figure 4.2. The link browser is used to visualise documents, create or navigate hyperlinks as well as to manage hyperlinks. In the default setting, the link browser visualises documents side by side (DI3), allowing users to easily create and navigate hyperlinks. The link browser further provides end users with visual handles in order to open and close documents, create, edit, delete or navigate hyperlinks. The link browser visualises a document in the left-hand side (e.g. `DocFormat1` in Figure 4.2) or right-hand side panel (e.g. `DocFormat2` in Figure 4.2) as a result of navigating a hyperlink or upon a user request to visualise a specific document. The document visualised on the right-hand side of the link browser might, for example, be a target of a hyperlink contained in a document visualised on the left-hand side or share no hyperlinks with the other adjacent document. We believe that the side-by-side visualisation of linked documents is close to many user's workspace settings where they concurrently read, compare and manipulate documents [2]. Recent systems have adopted the side-by-side visualisation of documents in order to help users to actively read and annotate their documents [34]. Note that the side-by-side visualisation of linked documents in the link browser is similar to the user interface configuration proposed by Bush for the Memex system [26]. Further, Signer and Norrie have also discussed a similar approach for visualising linked documents [125, 126]. It is worth mentioning that besides the default setting of visualising documents side-by-side, the link browser can also be configured to visualise a single document or more than two documents.

The link browser allows the creation of bi- and multidirectional hyperlinks between snippets of information in the supported document formats (DI2), something that is not supported by existing linking solutions. Bidirectional hyperlinks are preferred over unidirectional hyperlinks since they help to overcome the problem of broken hyperlinks. In order to illustrate the usefulness of bidirectional hyperlinks, imagine a document A that has a unidirectional hyperlink targeting document B. If document B has been deleted, then the hyperlink in document A will become broken. This can be prevented by using bidirectional hyperlinks.

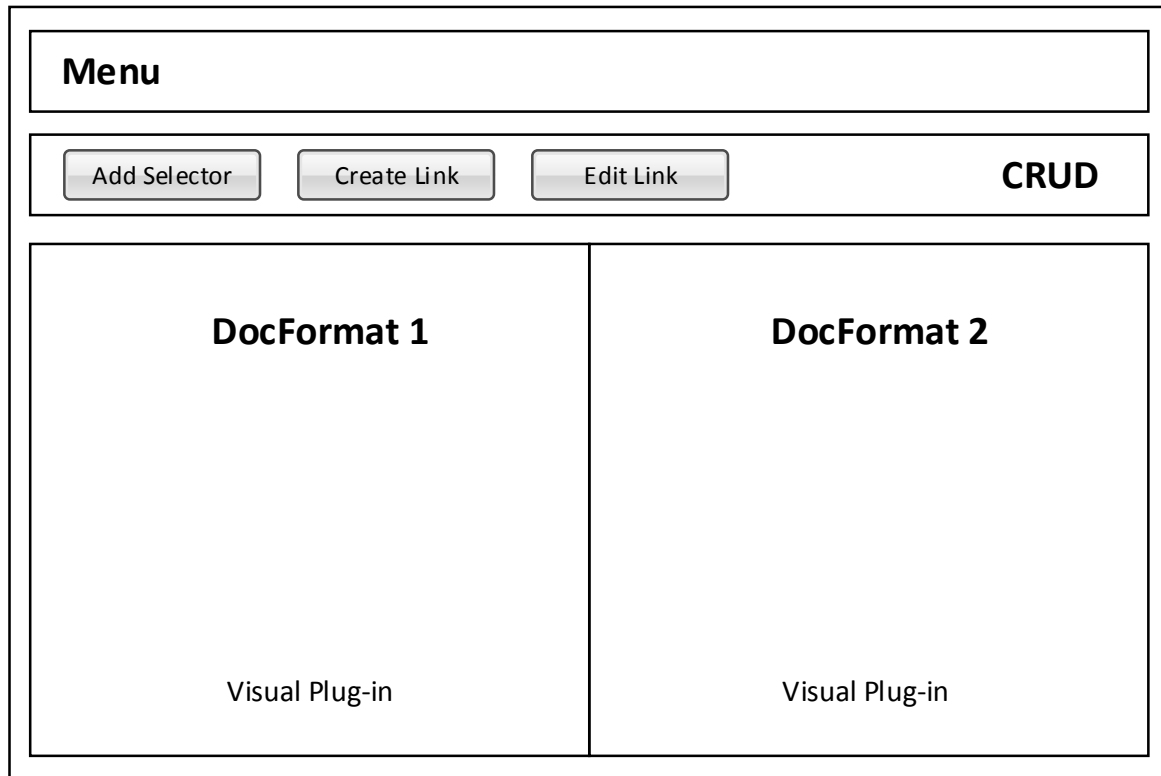


Figure 4.2: Link browser visualising documents side by side

If a user tries to delete a target document, they should be informed that the document is being linked by another document and that the deletion of the target document should be prevented or that the target should only be deleted together with the corresponding hyperlink.

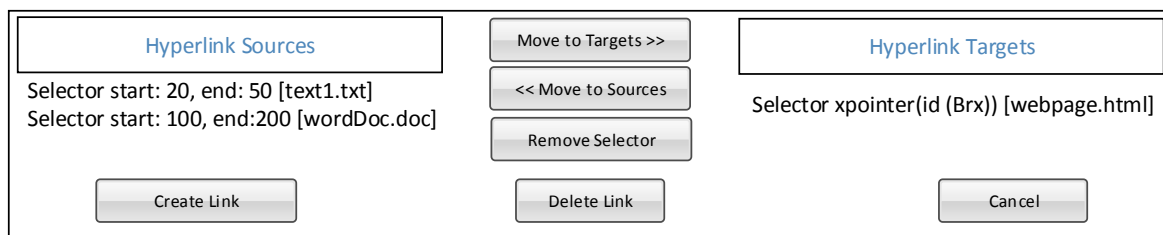


Figure 4.3: The hyperlink overview menu for editing a hyperlink's sources and targets

Multidirectional hyperlinks should enable users to create hyperlinks in cases where a document has a relationship with multiple other documents. It is common to find multiple web pages linked to a specific

Wikipedia¹ article. With the simple hyperlinks on the Web, none of these web pages (including the Wikipedia article) are aware of such an explicit relation since it has to be established via multiple autonomous unidirectional hyperlinks. In contrast, the support of multidirectional hyperlinks between related documents should enable users to be well aware of the relationships between the different documents. Multidirectional hyperlinks that are created via the link service can further be beneficial for desktop retrieval systems making use of the link service hyperlink metadata. For example, when a user searches for a document playing a role in a multidirectional hyperlink (i.e. one of the hyperlink sources or targets), a desktop retrieval system can use the hyperlink metadata in order to recommend the other documents participating in the multidirectional hyperlink.

A user can easily create bi- and multidirectional hyperlinks between their documents. After opening a document in the link browser, they can select parts of the document (DI1) and choose the option **Add Selector** to create a selection (selector) via the supported CRUD operations shown in Figure 4.2. The link browser then allows the user to choose (open) another document in order to create another target selector. The user might then either confirm the creation of a new bidirectional hyperlink by pressing the **Create Link** button or open more documents to create more selectors that will lead to the creation of a multidirectional hyperlink. A user can always choose to edit a hyperlink's sources and targets before confirming its creation. The link browser provides the user with an overview about the created selectors giving them flexibility in editing the hyperlink sources and targets. For instance, Figure 4.3 shows three different selectors made by a user; a selector in a text document (`text1.txt`), a selector in a Word document (`wordDoc.doc`) and a selector in an HTML document (`webpage.html`). A user can easily move the Word document selector from the hyperlink sources to the hyperlink targets leading to the multidirectional hyperlink illustrated in Figure 4.4, which is not supported by the three involved document formats or any existing linking solution. In order to create the multidirectional hyperlink, the user has to confirm its creation after finishing the editing of its sources and targets.

¹<https://en.wikipedia.org>

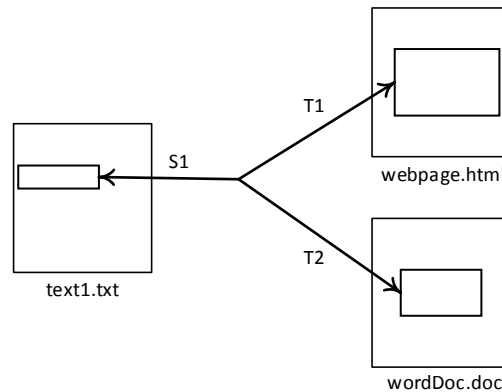


Figure 4.4: A multidirectional hyperlink between snippets of information in three documents of different formats

4.1.2 Link Browser Extensibility

In contrast to all existing linking systems that neglect the extensibility of their link browsers, we take into account that our link browser should be able to support the visualisation of existing as well as emerging document formats (R1 and R2). *We have defined a list of the necessary functionalities required to visualise and interact with any document in an ideal link service such as the opening of a document, the highlighting of a document's selectors, the creation of a selector or the navigation to a hyperlink target.* The complete list of functionality is detailed in Section 4.5.2. All this functionality is abstracted by the link browser and the link browser does therefore not make any assumptions about the document format to be visualised in its windows. For any document format to be supported by the link service and visualised in the link browser, we propose that a *visual plug-in* extending the link browser must be provided. The visual plug-in must implement all the abstract functionality required to visualise and interact with its resources (documents) and selectors.

4.1.3 Link Model

In order to support the linking between documents visualised in the link browser, the link service requires concrete information on how to address selectors and resources of a specific document format (DI1). Moreover, the flexibility and extensibility of the link service to integrate existing as well as emerging document formats asks for a general schema that defines selectors and resources of existing and emerging document

formats (R1 and R2). Unfortunately, for a number of reasons we cannot provide such a general schema. There exists a variety of document formats and standards including markup languages, WYSIWYG formats as well as the new trend for eBook standards (e.g. EPUB [41]). These document formats have different logical representations such as a linear document model, a flat pseudo-hierarchical document model, an unconstrained tree-like document model or a constrained heterogeneous tree-like document model as classified earlier by Furuta [57]. Furthermore, the document formats with similar document models might still show differences in terms of the granularity of the lowest level of atomic objects supported by the model. An atomic object in a model might, for example, be a text string representing a paragraph, a sentence or even an individual character. Last but not least, a multitude of media types such as text, images, sound or video clips are supported in different document formats. All these media types are informative and a selector within these media types can form the source or a target of a hyperlink. One might, for instance, have a hyperlink from a selection in an PDF document to a specific time span of a video clip that is embedded in an HTML document. All the aforementioned reasons and issues together indicate that it is impossible to make any prior assumptions about the types of linked documents and their selectors (R2 and DI1). That is one of the main reasons why most existing linking solutions show some limitations in terms of extensibility since they have adopted monolithic link models that only support a fixed number of document formats.

One possible solution is to provide an intermediate format (e.g. XML) and an addressing schema for every document format willing to benefit from our link service. The specific schema for a document format then can be used to convert the document format back and forth to the intermediate format. In this case, the link service should only know how to address resources and selectors of the intermediate format. However, this solution is very costly and not easy to implement considering the heterogeneity of document models. Furthermore, the definition of the addressing schemas for the different document formats is a tedious and complex task given the extensive specifications of some document formats (e.g. 5585 page specification of OOXML or 751 page specification of the OpenDocument format).

We believe that the link service requires a link model that is general and flexible enough to be used for defining resources and selectors of existing and emerging document formats. The chosen link model should

provide enough abstractions for resources, selectors as well as hyperlinks. One possible definition could be that hyperlinks might have one or more sources and one or more targets. The source could be a selector that is represented via an XPointer expression for tree-like documents or start and end indices for some document formats having a linear document model. The definition of sources and targets should be kept abstract and each document format willing to benefit from the link service should then provide a concrete definition of how parts of a resource can be addressed by extending the abstract hyperlink concepts via a *plug-in mechanism*. The definition of the document addressing parts can be achieved by using third-party document APIs, different programming language libraries for document formats or available implementations of some standards such as Document Object Model (DOM) libraries.

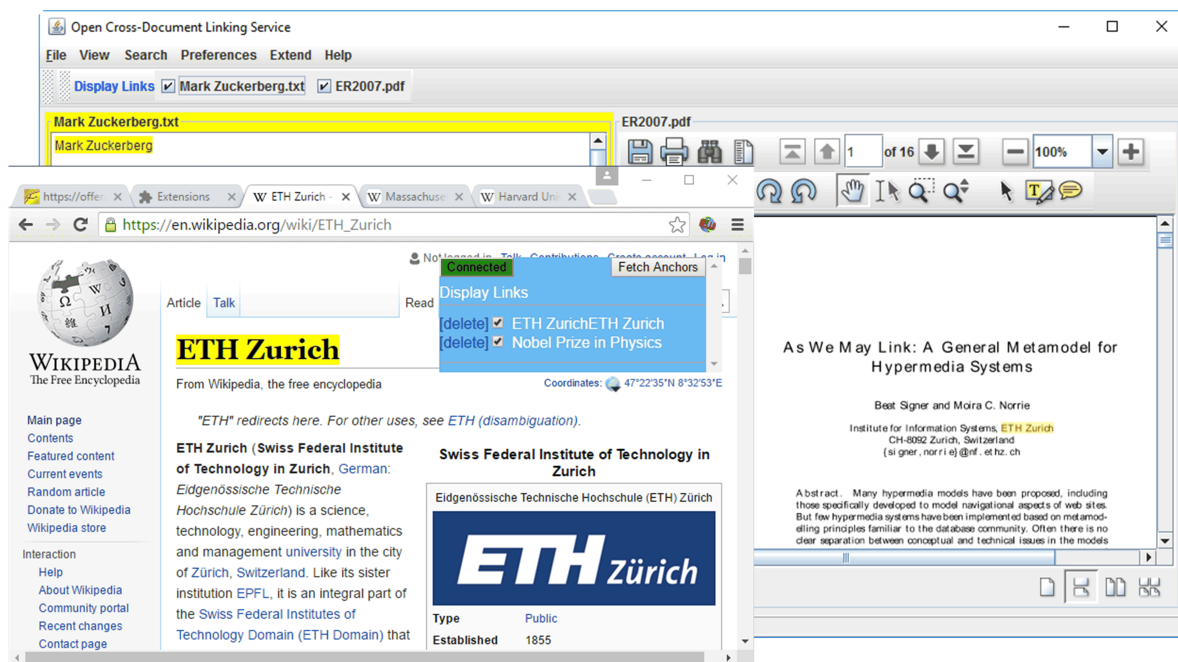


Figure 4.5: A bidirectional hyperlink between a PDF document visualised in the link browser and an HTML document visualised in the Google Chrome web browser

We have chosen the RSL metamodel presented in Section 2.4.2 as the link model for our cross-document link service. The RSL metamodel is based on the concept of linking arbitrary entities. A resource and selector concept is defined in RSL. The former defines a media type such as a complete document whereas the latter is attached to a resource and is used to address parts of that resource (DI1). The RSL metamodel has

already proven its flexibility and extensibility (R1 and R3) and served as a basis for a number of hypermedia applications [122, 121, 107]. A more elaborated motivation for using the RSL metamodel as the core link model as well as a comparison between the RSL metamodel and XLink standard is given in Section 4.5.1. For every document format that should be supported within our link service, a *data plug-in* extending the RSL abstractions and defining how to address selectors and resources of a given document format has to be provided. The data plug-in for a specific document format must provide the definition of its logical structure by extending the RSL resource and must further define how to create selectors within its structure by extending the RSL selector.

4.1.4 Integration of Third-Party Document Viewers

Our link service further addresses the challenge of seamlessly integrating third-party document viewers (R3). This gives users the flexibility to continue using document viewers they are familiar with. Furthermore, they can create hyperlinks between documents visualised in the link browser and documents visualised in their own third-party document viewers. For example, Figure 4.5 shows a bidirectional hyperlink created between a PDF document visualised within our link browser and an HTML document visualised in an external web browser. Moreover, users are able to create hyperlinks between documents visualised in different third-party document viewers. In the rest of this section, a discussion of how the link service can integrate existing and emerging document viewers is followed by some use cases explaining the creation of hyperlinks in the supported document viewers.

Nowadays, some features for extensibility are offered by most document viewers via their dedicated SDKs or APIs such as the Foxit Reader Plug-in SDK for the Foxit Reader PDF viewer or Microsoft's Office Developer Tools for Office applications. This enables developers and end users to support new features (e.g. visual handles) and customise the behaviour of the extensible third-party document viewer. Developers are able to build add-ins—sometimes also called extensions or plug-ins—for a document viewer by using the provided SDK. In order to extend a document viewer with an add-in, the add-in has to be installed in the document viewer according to the specification given by its vendor. Normally, an end user should be able to easily install an add-in and to benefit from its provided functionality.

We exploit extensibility features offered by third-party document viewers in order to integrate them in our link service. For every third-party document viewer to be integrated within our link service, an add-in for the third-party document viewer is required. We have defined a list of functionality (detailed in Section 4.6.1) that must be supported by a third-party document viewer add-in in order to be successfully integrated with our link service. In brief, an add-in must communicate with our link service about any selectors to be created or navigated. We do not limit the communication between the link service and third-party document viewers to a specific type of communication protocol, but rather support various communication channels including TCP sockets, WebSockets and a RESTful API. As discussed in Section 2.8, the support of a wide range of communication protocols should facilitate the integration of existing and emerging third-party document viewers in a link service (R4). The add-in should further provide visual handles (GUI actions) allowing users to create, navigate, delete and edit selectors (hyperlink sources or targets).

Similar to the document formats visualised in the link browser, the link service requires information in order to address resources as well as selectors of document formats integrated with their own third-party document viewers. The integration of a document format visualised in its third-party document viewer therefore also asks for a data plug-in defining how to address its resources as well as selectors. The data plug-in for a specific document format integrated with its third-party document format must provide the definition of its logical structure by extending the RSL resource and must further define how to create selectors within its structure by extending the RSL selector. Thereby, the structure of data plug-ins for document formats integrated in the link browser is the same as the structure of data plug-ins of document formats integrated with their third-party document viewers. *It is worth mentioning that if the link service has to support the visualisation of a document format A and a document viewer of the same document format (A), a single data plug-in defining how to address resources and selectors of the document format A is sufficient.*

Data plug-ins and add-ins are not sufficient for a successful integration of existing and emerging third-party document viewers in the link service. The fact that document formats of third-party document viewers have different logical structures as well as selectors and resources implies that messages exchanged between the link service and an add-in are different than messages exchanged between the link service and another add-in.

In order to illustrate the difference between the exchanged messages, assume that the HTML and PDF document formats are integrated in the link service via their external document viewers (e.g. Google Chrome and Adobe Acrobat Reader). HTML and PDF document formats have different representations of their selectors (e.g. XPointer-like expression for an HTML selector and a 2D rectangular shape as well as a page index for a PDF selector) and thereby a message exchanged between the link service and Google Chrome about the creation of a selector in an HTML document is different than a message exchanged between the link service and Adobe Acrobat Reader about the creation of a selector in a PDF document. *In order to cater for this challenge, we believe that a mediator component should form an integral part of the link service in order to abstract the messages exchanged between the link service and third-party document viewers.* This allows the link service to understand message types and structures from any add-in and perform the required actions. Add-ins further should be able to understand the type and structure of any message sent by the link service in order to perform the required task. We therefore propose a *gateway* component in order to facilitate the integration of third-party document viewers. The gateway component is the most essential component for integrating existing as well as emerging third-party document viewers. *We have defined a list of functionality (detailed in Section 4.6.2) that is necessary to translate any message exchanged with a third-party document viewer add-in.* All this functionality is abstracted in the gateway component. For every third-party document viewer to be supported in our link service, a *gateway plug-in* has to be introduced that extends the gateway component in order to translate the messages communicated between its corresponding document viewer add-in and the link service.

There is no doubt that our approach for integrating third-party document viewers is totally different from the one used in Sun's Link Service described earlier in Section 2.6.1, where third-party document viewers have to be rewritten in order to benefit from the link service. Moreover, in contrast to Microcosm [73], our approach for integrating third-party document viewers does not restrict the link service to a specific family of third-party document viewers. Any document viewer with an API or a dedicated SDK that allows access to its supported document formats, the development of new visual handles and the communication with other systems (i.e. our link service) can be integrated with our link service.

4.1.4.1 Communication Between the Link Service and Add-ins

In order to illustrate the communication between the link service and document viewer add-ins, we present two different scenarios for creating hyperlinks. In the first scenario, we explain the creation of the hyperlink depicted in Figure 4.5. In the second scenario, we explain the creation of a hyperlink between two different documents visualised in different third-party document viewers. In both scenarios, the user can confirm the creation of the hyperlinks using the **Create Link** button or use the hyperlink overview menu introduced earlier and shown in Figure 4.3 for editing and confirming the creation of the hyperlinks. The hyperlink illustrated in Figure 4.5 can be created in a few simple steps. Assume that the user is working simultaneously with the PDF document visualised in the link browser and the HTML document visualised in the Google Chrome web browser. The user selects **ETH Zurich** from the PDF document and chooses the option to create a selector from the link browser's supported CRUD actions. The selected PDF selector will be listed under the **Hyperlink Sources** in the hyperlink overview menu. Note that the user is given the flexibility to edit the hyperlink sources and targets and can for example move the PDF selector to the hyperlink targets. As illustrated in Figure 4.6, the user then selects **ETH Zurich** from the HTML document and chooses the option to create a selector from the GUI actions supported by Google Chrome add-in. The add-in then sends a request (in a platform-independent data interchange format) to the link service with information about the document and the selector. The link service asks the corresponding HTML gateway to translate the received message. In other words, the HTML gateway should transform the information about the HTML document into an RSL resource presenting the HTML document. Furthermore, the HTML gateway should also transform the information about the HTML selector into an RSL selector presenting the HTML selector. This means that the HTML gateway communicates with the HTML data plug-in in order to transform part of the received message into an HTML resource or selector. The link service then adds the HTML selector to the active hyperlink (the hyperlink being created). The HTML selector will be automatically listed under the **Hyperlink Targets** in the hyperlink overview menu. The hyperlink will be stored after confirmation by the user. When the hyperlink is saved, the link service asks both the PDF visual plug-in and Google Chrome add-in to update their documents in order to visualise the new hyperlink.

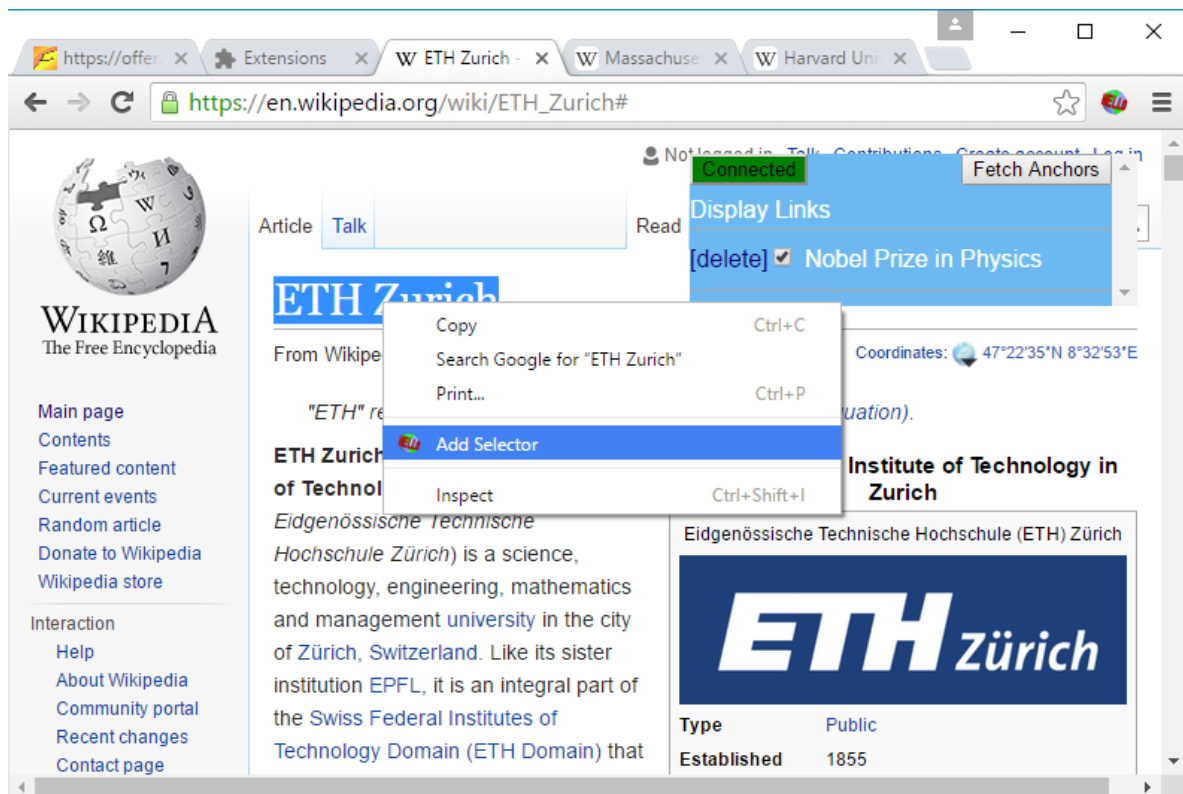


Figure 4.6: Creating a selector in an HTML document using a Google Chrome Add-in

In a similar way as described in the previous scenario, a user can also create a hyperlink between documents visualised in different document viewers. Suppose that besides Google Chrome also Adobe Acrobat Reader is supported by the link service. The user wants to create a hyperlink between a PDF document visualised in Adobe Acrobat Reader and an HTML document visualised in Google Chrome. They select parts of the PDF document and choose the option to create a selector from the GUI actions supported by Adobe Acrobat Reader add-in. The Acrobat Reader add-in then sends a request to the link service with information about the document and the selector. The PDF gateway is then asked to return the PDF resource and selector from the received message. The link service then lists the returned selector under the **Hyperlink Sources** in the hyperlink overview menu. The user then selects parts of the HTML document and chooses the option to create a selector from the GUI actions supported by Google Chrome add-in. In a similar way, the HTML gateway returns the HTML resource and selector from the message coming from its corresponding add-in and the HTML selector is listed under

the **Hyperlink Targets** in the hyperlink overview menu. After a confirmation by the user, a hyperlink between the two documents is created.

4.1.5 Plug-in Metadata

As mentioned before, data and visual plug-ins have to be provided for document formats to be visualised in the link browser, whereas data and gateway plug-ins are necessary for document formats to be visualised with their third-party document viewers (in addition to the add-ins). The multiple types of document format plug-ins that exist in the link service require a mechanism to differentiate between them in order to correctly use them in the link service. *We have defined a list of key/value metadata that should be included in the plug-ins. As detailed in Section 4.4.1, based on the type of the plug-in (i.e. data, visual or gateway) different metadata keys and values should be included in the plug-in.* Document format plug-ins must provide this metadata to be a valid extension for the link service. The plug-in tracking component exploits the plug-in metadata in order to identify them and correctly inject a plug-in in the link service. Furthermore, the plug-in tracking component will prevent any plug-in with invalid metadata to exist in the link service.

4.1.6 Users of the Link Service

4.1.6.1 End Users

End users can use the link service to visualise arbitrary document formats as well as to create, navigate and edit hyperlinks. They can also create hyperlinks between documents that are visualised in the link browser and documents visualised via their third-party document viewers. Furthermore, the link service is designed to allow end users to extend the link service on demand (R5). In order to facilitate the dynamic extensibility of the link service, an online plug-in repository that stores and manages the different types of plug-ins can be accessed by end users in order to search and download the different types of plug-ins for the different document formats (R1, R2 and R5). The online repository provides a RESTful API enabling the plug-in tracking component to keep track of available plug-ins and add-ins. In order to illustrate the link service's dynamic extensibility let us have a look at an example where the link service is extended to support a new document format.

The end user can install the link service in a few steps. The link service comes with support for a minimal set of document formats. Let us assume that the user wants to extend the link service to support the Word document format. In that case, the user can either open a web page of the online repository to search for plug-ins for the Word document format or use a simple user interface provided by the plug-in tracking component to search for them. The user finds that there are two options for the Word document format. The first option is an extension which visualises documents and their selectors in the link browser. This means that the extension consists of only two plug-ins, a data plug-in as well as a visual plug-in. Further, there exists an extension with visualisation support for the Microsoft Word application (R3). This implies that this extension consists of a data and a gateway plug-in as well as the Microsoft Word application add-in. In both cases, the user must install the data plug-in which has some metadata in order to be correctly identified and used by the link service components. The plug-in tracking component reads the data plug-in's metadata and the plug-in is downloaded via a secure shell protocol². *The plug-in tracking component then does the necessary work to inject the plug-in into the running link service (R5).* If the user wishes to visualise their documents with the link browser, the visual plug-in for the Word document format has to be installed. Similar to the data plug-in, the visual plug-in contains specific metadata and will be installed and injected into the link service based on the same mechanism (R5). After successfully installing the data and visual plug-ins, the user will see the Word document format appear in the list of supported document formats and can start visualising and creating hyperlinks between the different supported documents.

In the case that the user wishes to visualise their documents in Microsoft Word, they have to install the gateway plug-in and follow the instructions provided for the Microsoft Word add-in in order to extend Microsoft Word. The gateway plug-in is installed based on the same mechanism used for data and visual plug-ins (R5). When the data and gateway plug-ins as well as the Microsoft Word add-in have been successfully installed, the user should be able to create hyperlinks in their Word documents and link them to any arbitrary supported document format.

²https://en.wikipedia.org/wiki/Secure_Shell

4.1.6.2 Third-Party Developers

In most design decisions, we took into account that our link service should be extensible to support existing and emerging document formats not only by us but also by third-party developers. Therefore, *we have strived to design our link service in a way that allows any third-party developer to easily develop the different plug-ins or add-ins required for integrating a new document format into the link service.* First of all, third-party developers are not required to understand the communication between the different components of the link service. Second, the abstracted functionality for visual as well as gateway plug-ins consists of a minimal set of methods which are well documented. Developers are provided technical explanations as well as examples in order to help them in implementing the required plug-ins for integrating a new document format. Last but not least, we assume that most SDKs for third-party document viewers are documented and publicly available. A developer is expected to develop a data as well as a visual plug-in for documents that should be visualised in the link browser, whereas they have to develop a data, a gateway and a third-party document viewer add-in for documents that should be visualised in their third-party document viewer. When the development of the required plug-ins is completed, the developer should upload the different plug-ins to the online plug-in repository.

4.2 Architecture Overview

The general architecture of our open cross-document link service is illustrated in Figure 4.7. In contrast to most existing linking solutions that have been built as monolithic components, our link service offers a *plug-in architecture* for integrating different document formats as well as third-party document viewers. From a software engineering perspective, the plug-in architecture is well-known for its support of system extensibility (R1), modularisation, as well as configuration (R5) [92, 32, 97]. A more detailed discussion about the choice of the plug-in architecture and its benefits is presented in Section 4.4. In the following we briefly elaborate on the different link service components and a detailed explanation of each component is given in the following sections.

Most of the link service components are flexible and extensible to support the general goal of integrating existing as well as emerging document

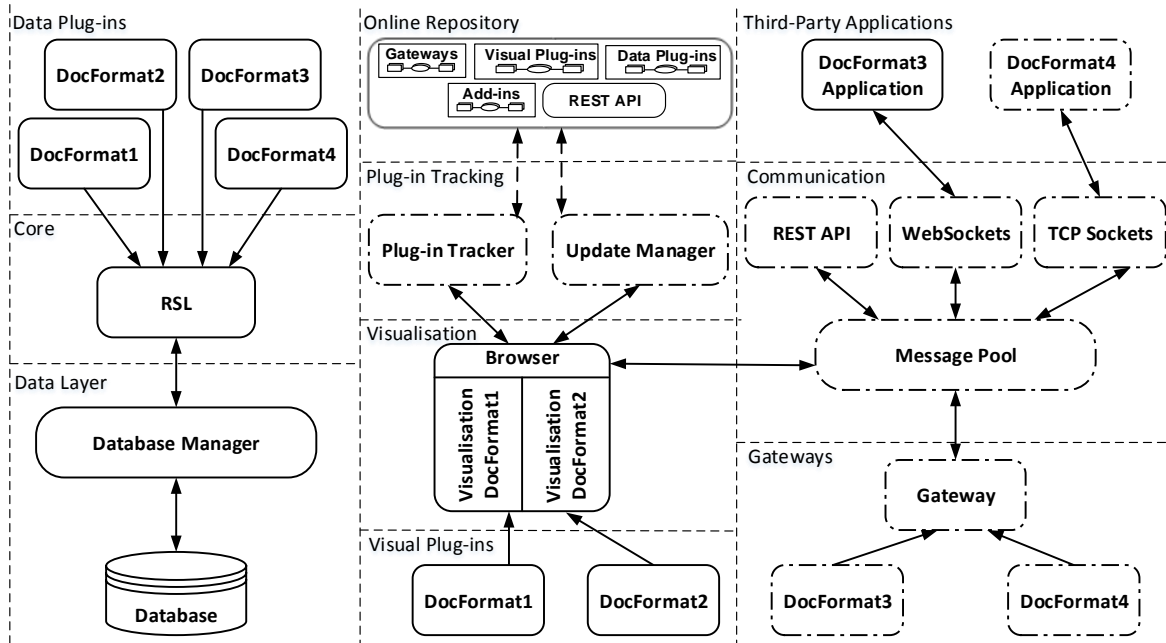


Figure 4.7: General open cross-document link service architecture

formats (R1). The core link service is extensible to support arbitrary document formats. The use of RSL as a core link model overcomes a number of limitations exposed by most existing linking solutions presented in Chapter 2 such as the limited set of supported document formats. The RSL metamodel provides the general abstractions that help the link service to address parts of any document format (DI1). As mentioned in Section 2.4.2, RSL supports bi- and multidirectional hyperlinks (DI2). As discussed earlier, for any document format that should be supported by the link service and visualised internally (in the link browser) or via its third-party document viewer, a data plug-in extending the RSL resource and selector has to be provided. The link service architecture further consists of a data layer that is in charge of storing any RSL metadata such resources, selectors or hyperlinks. The data layer is flexible to support different database management systems for the persistent storage of documents and hyperlink data. In the current implementation of the link service, the db4o³ object database is used for storing the system objects.

The visualisation component of our link service consists of a link browser to visualise the supported document formats side by side (DI3). In general, any visual plug-in extending the link browser has two main responsibilities. First, it has to render a specific document format and

³<http://www.db4o.com>

visualise any selectors that have been defined. Second, it should provide a visual handle for the basic create, read, update and delete operations for a specific document selectors.

The gateway and communication components play a major role in integrating third-party document viewers (R3). A document format gateway is responsible for launching the corresponding document viewer and handling the messages with the document viewer add-in. Messages are exchanged via any communication channel provided by the communication component (R4) and are managed via the message pool component. Similar to the visual plug-ins, a document viewer add-in should provide visual handles for the CRUD operations on document selectors.

The plug-in tracking component consists of a plug-in tracker and an update manager. These two components are responsible for keeping track of the installed plug-ins and for installing new plug-ins on demand (R5) by communicating with the online plug-in repository that manages data, visual and gateway plug-ins as well as different third-party document viewer add-ins.

4.3 Communication Between Link Service Components

In order to understand how the different components are communicating with each other and how the link service is actually working, we illustrate three main scenarios of interacting with the link service (i.e. the opening of a document, the navigation of a hyperlink as well as the creation of a hyperlink) as illustrated in the sequence diagram depicted in Figure 4.8. We explain the opening of a document and the navigation of a link scenario. Please refer to Section 4.1.1 and 4.1.4 for explanations of the third scenario; the creation of a hyperlink.

The end user can open any document in a format that is supported by the link service. Thereby, documents can either be stored in the link service database or in the local file system. When a document is selected to be opened, the browser retrieves supplementary metadata for the given document from the database via the core RSL component. The retrieved data contains information about the format of the document as well as its associated anchors. Note that an anchor in our link service contains information about a hyperlink source which is either

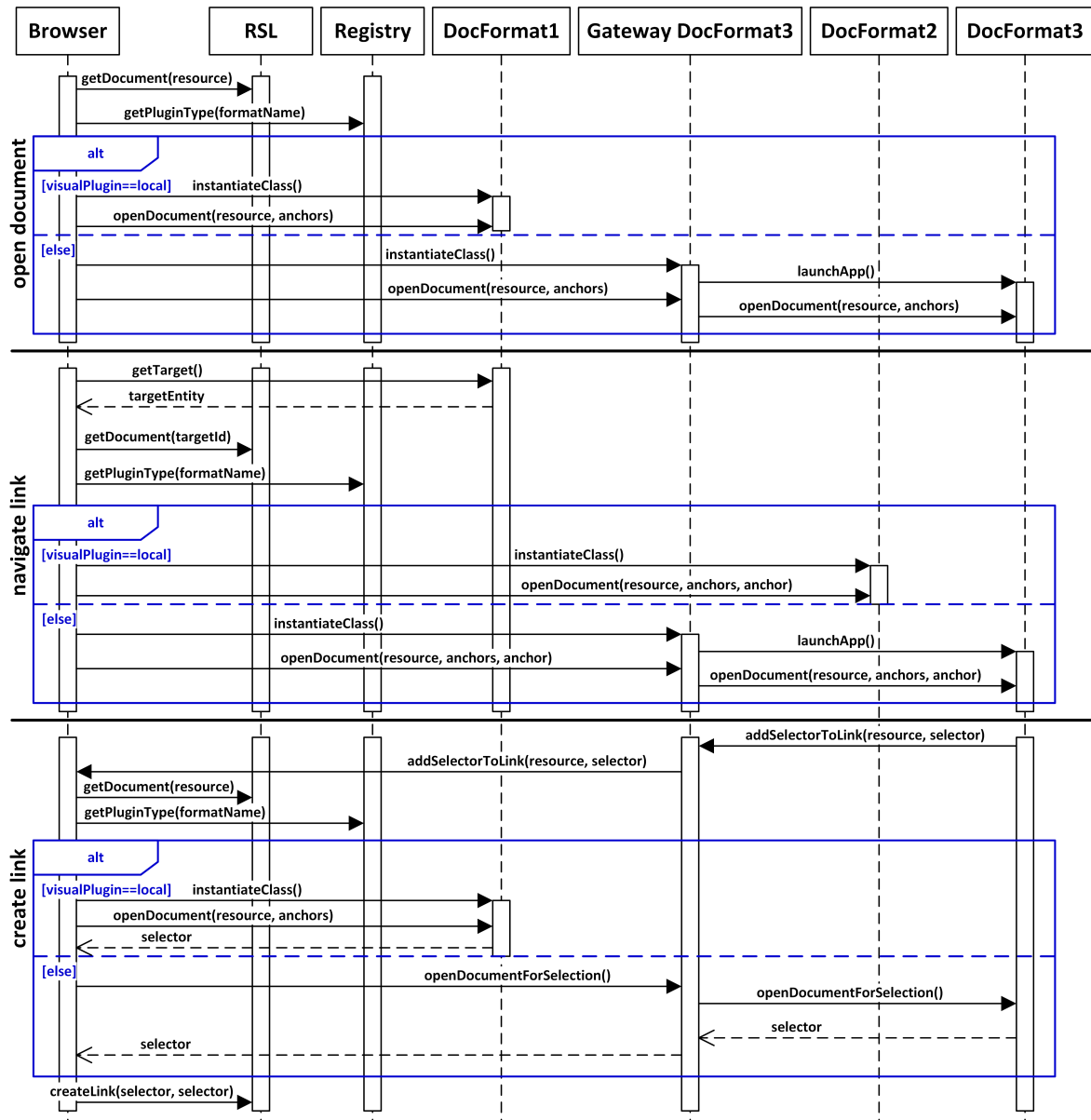


Figure 4.8: Communication among different cross-document link service components

a selector in a document or a complete document. It further contains information about all other sources and targets participating in the hyperlink. The browser then checks the type of plug-in that is installed for the given document format via a registry in the plug-in tracker component. The browser forwards a request to the corresponding visual or gateway plug-in in order to visualise the document in the link browser or to show it externally in a third-party document viewer. If the intended plug-in is a visual plug-in (DocFormat1 in our example), it visualises the

document and its anchors in a panel within the link browser. On the other hand, if the intended plug-in is a gateway (`Gateway DocFormat3` in our example), the gateway is asked to launch the corresponding document viewer. When a specific document viewer is launched, its add-in (`DocFormat3` in our example) should instantiate the connection with the link service via the supported communication channels. The link service then passes the document and its anchors to the gateway. The gateway transforms the data (i.e. RSL resource and selectors) into a message to be sent to its corresponding add-in. The message is represented in a data interchange format and contains information about the document and its anchors. The message further contains other essential information in order to inform the corresponding add-in to visualise the document. If the add-in communicates with the link service via a full-duplex communication channel (e.g. TCP sockets), the link service sends the message formed by the gateway to the add-in via the active communication session. Otherwise the add-in should pull the message from the link service. The add-in then asks its associated third-party document viewer to open the document and render existing anchors. The question is what should happen if a user opens a document with a supported third-party document viewer rather than with the link browser? In this case, the add-in forms a message containing information about the document and asks the link service to retrieve potentially stored data (i.e. anchors) about the currently visualised document. The link service communicates with the corresponding gateway in order to transform parts of the message (i.e. the data about the document) into an RSL resource. If any anchors have been defined for the currently opened document, the link service passes its anchors to the gateway. The gateway is expected to form a message containing information about the retrieved anchors. The message is then sent to the add-in and in this case, the add-in should update the currently opened document and visualise its anchors.

When a user clicks on a specific hyperlink target in a document that is visualised in the link service browser, the browser communicates with the core RSL component to retrieve the target document and its anchors including the exact target selector of the target document. The target document is then visualised with its anchors and the target selector of the followed hyperlink is highlighted in a different colour than the other selectors. The two documents can, for example, be visualised next to each other in the browser (`DocFormat1` and `DocFormat2`) (DI3) or one in the browser while the other document is rendered in a third-party document

viewer (`DocFormat1` and `DocFormat3`). In the case that a hyperlink is selected in a document that is visualised in a third-party document viewer, the add-in sends a message containing minimal information about the target (i.e. its ID) and requesting the link service to visualise the target document. The link service communicates with the corresponding gateway in order to retrieve the target ID from the received message. From there on the browser handles the request in the same way as in the previous scenario to either visualise the hyperlink target in the browser or in a third-party document viewer.

4.4 Dynamic Link Service Extensibility

As mentioned before, the link service has a plug-in architecture which is suitable for applications that should be extended by third-party developers [143, 33]. The plug-in architecture is nowadays one of the most widely used architectures for extensible and complex systems such as server applications, web browsers, integrated development environments (IDEs) and embedded systems [63]. The link service plug-in architecture is further supported by the use of an object-oriented programming language as well as the extensive use of design patterns that facilitate the evolution of systems [15].

We have used the Open Service Gateway initiative (OSGi) [72] for the development of the link service and realising its dynamic extensibility. The OSGi specification defines a dynamic modular system for the Java programming language and was originally designed for embedded systems. Various service applications (e.g. IBM Websphere) make use of OSGi and also the Eclipse IDE uses OSGi to enable the modularisation of its components and to support dynamic extensions via plug-ins. Conceptually, the OSGi framework consists of three layers, including the *module* layer, the *life-cycle* layer and a *service* layer. Each of these layers has a specific role in modularising an application. The module layer is responsible for packaging and sharing the application code. Each module of an application is called a *bundle* and is the same as a Java JAR file with some extra metadata in the form of a manifest file. Listing 4.1 illustrates some metadata contained in a manifest file of an OSGi bundle. The `Bundle-Name` and `Bundle-Version` provide information about the name of the OSGi bundle and its version, whereas the `Bundle-ManifestVersion` specifies the OSGi specification version

used to build the system. The `Import-Package` and `Export-Package` metadata explicitly define the exported and required packages for executing the bundle. The life-cycle layer is responsible for the management of specific modules at execution time. It controls when to install, resolve, start, activate, stop or uninstall a module. Finally, the service layer is responsible for the interaction and communication among an application's installed modules. Most of the components depicted in the link service architecture in Figure 4.7 are bundles. All document format plug-ins including the data, gateway and visual plug-ins are OSGi bundles but with extra metadata. Note that *the life-cycle layer is extensively used by the plug-in tracker of the link service for achieving the link service's dynamic extensibility by controlling the life cycle of any document format plug-in.*

```
1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: Core
4 Bundle-Version: 1.0
5 Import-Package: org.associations.collections, org.sigtec.exception
6 Export-Package: org.rsl.core, org.rsl.util
```

Listing 4.1: Some metadata in an OSGi bundle's manifest file

We decided to use OSGi for several reasons. First of all, a plug-in architecture does not necessarily guarantee the dynamic extensibility of a system. An extra mechanism is needed to provide and ensure the dynamic extensibility of any plug-in architecture. Fortunately, the OSGi specification supports the implementation of dynamic extensibility of the applications. That is one of the main features acquired by Eclipse and other well-known server applications from the use of OSGi. Second, OSGi enhances the modularisation of our link service. Aside from reducing the application complexity, OSGi offers a good mechanism for code sharing between different modules. OSGi modules do not arbitrarily share code as with Java JAR files but rather explicitly define export packages they want to share and import packages to be used. By this clear definition of exported functionality, an OSGi module cannot “misuse” any code of other modules. In other words, third-party developers of different document formats plug-ins cannot misuse any plug-in or component in the link service. Last but not least, our link service might provide different plug-ins (e.g. visual or gateway) for the same document format. Managing different versions of a module in a pure Java application often causes the so-called “JAR hell” problem [131], while the OSGi framework offers a mechanism for the versioning of modules and dependency resolution.

It is worth mentioning that before choosing the OSGi framework and the Java-based implementation of the link service architecture, we have considered to realise our link service based on the Open Web Platform⁴. The Open Web Platform implies the use of the plug-in architecture and the scripting approach [111] which produces extensible systems via the customisation of scripts [99]. Google Chrome and Firefox, for example, follow this approach of extensibility. For a number of reasons we did not realise the link service based on the Open Web Platform. There are currently only a few web-based open source libraries available for different document formats and most of them do not support the interaction with documents (i.e. violation of R1 and R2). Furthermore, most web browsers offer only limited support for communicating with third-party applications (e.g. only WebSockets) which violates the requirements R3 and R4 for an extensible link service.

4.4.1 Metadata and Online Repository

We have exploited the OSGi manifest file to correctly identify each document format plug-in. Aside from the specific OSGi metadata required by any OSGi bundle (Listing 4.1), different document format plug-ins must contain specific metadata to be a valid extension for the link service and to be correctly identified by the plug-in tracking component.

The **Extension-Name**, **Extension-Mime** and **Extension-Type** metadata is required for all types of plug-ins. This metadata provides information about the media type (e.g. text/html or application/pdf) supported by the plug-in, its name and its type (i.e. either a visual, a data or a gateway plug-in). A plug-in developer should maintain the consistency of the media type provided in a document format plug-in. In other words, a data plug-in and a visual plug-in for a given document format must have the same value for the media type. The same holds for data and gateway plug-ins in the case of a third-party document viewer integration.

The **Extension-Class** metadata should be included in visual and gateway plug-ins. As mentioned before, a visual plug-in for a specific document format must extend the abstract class provided by the link service visualisation component. Furthermore, each gateway has to implement the gateway interface. The link service can communicate with a visual plug-in in order to visualise documents or for the CRUD op-

⁴https://www.w3.org/wiki/Open_Web_Platform

erations on a link by instantiating the class that extends the abstract class in the visualisation component. On the other hand, the link service can communicate with a gateway plug-in to translate the exchanged messages by instantiating the class that implements the gateway interface. In order to correctly instantiate these classes, each class name combined with its corresponding package name (class location in its package) should be the value of the **Extension-Class** metadata included in the corresponding visual or gateway plug-in. Listing 4.2 shows some metadata of a visual PDF plug-in that is stored in its OSGi manifest file. In the visual PDF plug-in, the Java class `Pdf` with a class path `org.rsl.pdf.visual.Pdf` is the class extending the visualisation component of the link service. As also shown, the visual PDF plug-in makes use of some other modules (bundles or plug-ins) by importing their packages (e.g. importing the package `org.rsl.pdf.data` from its corresponding PDF data plug-in).

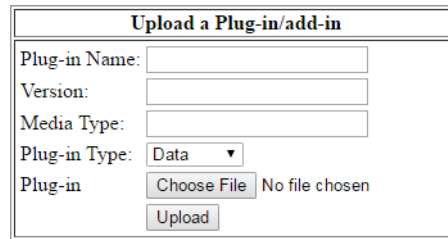
```

1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: Visual
4 Bundle-SymbolicName: org.rsl.pdf.visual
5 Bundle-Version: 1.0.0.qualifier
6 Import-Package: org.rsl.core, org.rsl.userInterface.util, org.userinterface.localvisualplugins,
7                 org.rsl.pdf.data
8 Extension-Name: pdf
9 Extension-Mime: application/pdf
10 Extension-Class: org.rsl.pdf.visual.Pdf
11 Extension-Type: visual

```

Listing 4.2: PDF Visual Plug-in Manifest' Metadata

The online repository is responsible for storing the different document format plug-ins as well as third-party document viewer add-ins. It provides a simple user interface for end users to search for the plug-ins. Furthermore, as illustrated in Figure 4.9, it provides an interface for third-party developers to upload the different document format plug-ins. The developer has to fill in the provided form by writing the plug-name, its version, the supported media type and the plug-in type (i.e. data, visual, gateway or an add-in). The online repository further provides a RESTful API to enable the link service to query information about the available plug-ins and add-ins. The RESTful API can be used to retrieve the plug-ins of a specific document format or a third-party document viewer. The plug-in tracking component of the link service uses the RESTful API of the online repository in order to give users with an overview of the available plug-ins and to install or update existing plug-ins.



The screenshot shows a web form titled "Upload a Plug-in/add-in". It contains several input fields: "Plug-in Name:", "Version:", and "Media Type:", each followed by a text input box. Below these is a "Plug-in Type:" label followed by a dropdown menu currently showing "Data". At the bottom, there is a "Plug-in" label, a "Choose File" button, the text "No file chosen", and an "Upload" button.

Figure 4.9: The interface used for uploading plug-ins and add-ins to the online repository

4.4.2 Plug-in Tracking

The plug-in tracker component makes use of the OSGi life cycle layer in order to extend the link service with new plug-ins. In brief, the plug-in tracker listens for any OSGi bundles (plug-ins) being started or stopped in the link service. The plug-in tracker will not allow a plug-in with missing or invalid metadata to be existent in the link service. Therefore, before extending the link service with a new plug-in, the tracker checks whether it is an extension based on our predefined metadata described in the previous section. If the plug-in is an extension, the tracker performs the necessary operations to integrate it in the link service.

In the case of a data plug-in, the tracker adds the media type to the list of document formats supported by the link model. In the case of a visual plug-in, the plug-in tracker checks whether the data plug-in of the same document format (media type) is already installed. If the data plug-in is missing, it communicates with the update manager to check if there is a data plug-in available in the online repository. If a positive reply is received from the update manager, the plug-in tracker asks the user a confirmation for installing the data plug-in. In the case that the data plug-in is missing or the user does not confirm the installation of the data plug-in, the visual plug-in will not be installed. If the data plug-in is already installed or installed after a user's confirmation, the plug-in tracker notifies the visualisation component that a new visual plug-in exists which in turn injects the new plug-in into the link browser. As a result, the user can see that the new document format has successfully been integrated in the link service and can start using it. Finally, if the plug-in is a gateway plug-in, the plug-in tracker will maintain the availability of its data plug-in with the same mechanism used when installing a visual plug-in and add it to the list of supported gateways.

In order to enable the dynamic loading and the discovery of valid document format plug-ins, the plug-in tracker component defines the different plug-in metadata as constants in a standalone Java class as illustrated in Listing 4.3. The plug-in tracker verifies the metadata for any (valid or non-valid) plug-in wishing to extend the link service. In order to do so, it first retrieves the plug-in metadata from its manifest file. If the plug-in failed to include one of the three required metadata (i.e. `Extension-Name`, `Extension-Mime` or `Extension-Type`), the plug-in will be rejected. If the plug-in is a visual or a gateway plug-in, the plug-in tracker ensures that the `Extension-Class` metadata is defined in the plug-in manifest file. Based on the plug-in type, each plug-in will be added to the registry with other plug-ins of the same type.

```
1 public static final String NAME_PROPERTY = "Extension-Name";
2 public static final String MIME_PROPERTY = "Extension-Mime";
3 public static final String CLASS_PROPERTY = "Extension-Class";
4 public static final String PLUGINTYPE_PROPERTY = "Extension-Type";
5 public static final String VISUAL_PLUGIN = "visual";
6 public static final String GATEWAY_PLUGIN = "gateway";
```

Listing 4.3: Plug-in metadata defined as String constants

4.5 Integration of Document Formats

In this section, we discuss the integration of document formats in the link browser, while in the next section we elaborate on the integration of third-party document viewers. We start by discussing the data plug-ins required for integrating any document format that is visualised in our link browser or via its third-party document viewer. We further motivate the use of the RSL hypermedia metamodel. After discussing the visual plug-ins required for integrating document formats in our link browser, we conclude this section with a concrete example for integrating the plain text document format.

4.5.1 Data Plug-ins

We have chosen the RSL metamodel presented in Section 2.4.2 to be the link model of our cross-document link service. The RSL metamodel overcomes a number of limitations exposed by the well-known XLink

standard presented in Section 2.4.1. RSL has proven its flexibility and extensibility in linking to a wide range of media types, whereas XLink is limited to a specific set of XML-based document formats. Thereby, we believe that the RSL abstractions (i.e. resource and selector concepts) are sufficient to integrate existing as well as emerging document formats. Furthermore, in contrast to the XLink standard, RSL supports overlapping hyperlinks. Last but not least, RSL goes beyond XLink by supporting user access rights for the different hypermedia application resources (e.g. documents, anchors, hyperlinks). It is worth mentioning that our link service currently supports user access rights on the hyperlinks.

For every document format that should be supported within our cross-document link service (i.e. visualised in the link browser or via a third-party document viewer), a data plug-in extending the RSL resource and selector concepts and defining how to address selectors and resources of the document format has to be provided. The data plug-in for a specific document format must provide the definition of its logical structure by providing an implementation of the RSL **Resource** and must further define how to create selectors within its structure by offering a specific implementation of an RSL **Selector**. Figure 4.10 illustrates four different PDF, HTML, plain text and XML data plug-ins for four different document formats. The PDF data plug-in defines its resource as a file path in the file system while its selector has been defined as a 2D shape. The HTML data plug-in defines its resources and selectors to be web URLs and XPointer-like expressions (e.g. Rangy⁵ selections) respectively. A selector for the plain text document format is defined as a tuple containing a start and end index (e.g. (s,e)) while its resources have been defined as in the PDF data plug-in. Finally, web URLs and XPointer expressions define XML resources and selectors respectively in the XML data plug-in.

4.5.2 Visual Plug-ins

As mentioned before, we ask for a visual plug-in extending the link browser for every document format to be integrated in the link browser. A visual plug-in for a document format must enable users to do all possible interactions (e.g. visualise a document, create a hyperlink or navigate a hyperlink). Therefore, in order to offer an extensible link browser as well as powerful visual plug-ins that help users in achieving their desired goals, in the visual component abstracted methods we take into account

⁵<https://github.com/timdown/rangy>

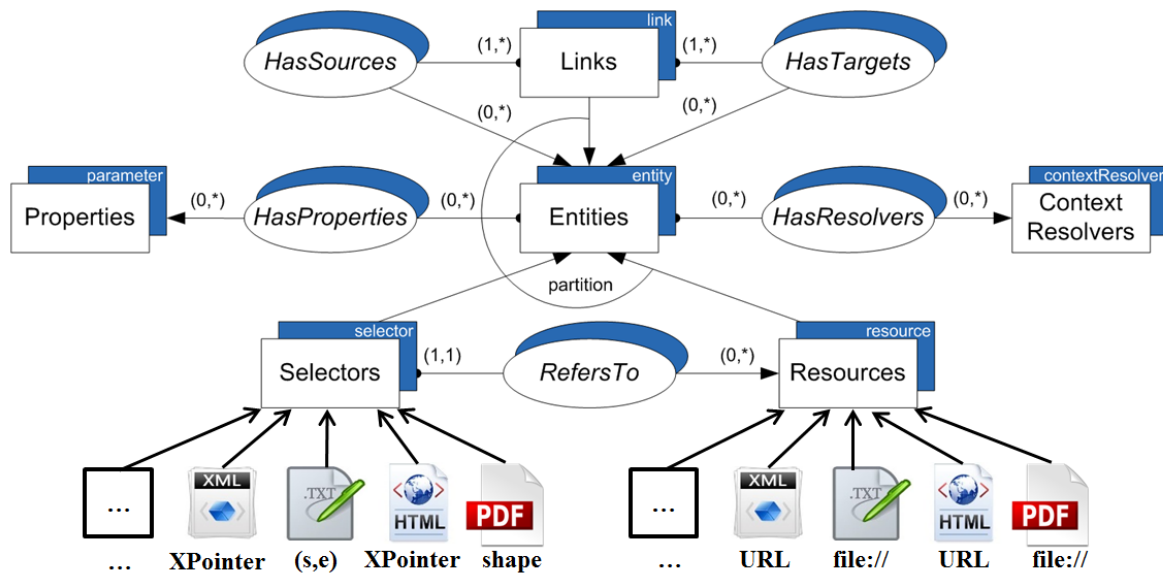


Figure 4.10: Different data plug-ins for different document formats

all possible end-user interactions with the link browser, including the following operations:

1. **Open a document** A user opens a document for
 - (a) **Reading:** They want to read the document in a reading mode (i.e. without the visualisation of the document anchors which is more suitable for reading activity [108], or as a result of
 - (b) **Hyperlink navigation:** in this case the user is “probably” interested in the specific hyperlink target contained in the document.
2. **Highlight document anchors** A user might be interested in reading a document without the highlighting of its anchors. However, they might be interested in visualising document anchors in order to navigate to other documents.
3. **Close a document** A user closes a document.
4. **Create selectors** A user selects parts of a document in order to create hyperlink sources or targets.
5. **Navigate to a hyperlink target** A user navigates to a specific hyperlink target.

6. **Update a selector** A user updates a specific document selector.
7. **Delete a selector** A user deletes a specific document selector.
8. **Update a document** A user updates a document (e.g. changes its URL or its file path).
9. **Delete a document** A user deletes a document (i.e. deleting an RSL resource).

The aforementioned interactions are abstracted in the visualisation component of the link browser. Listing 4.4 illustrates a number of abstract methods whereas the entire abstract class is included in full in Appendix B. The abstract `openDocument()` method enables the opening of a document in all the different scenarios (e.g. reading mode or as a result of a hyperlink navigation). The first parameter (`Object o`) of the `openDocument()` method enables a visual plug-in to visualise documents that are stored or not yet stored in the link service database. In a case where a document has to be visualised for the first time using the link service, the parameter `o` represents the document location (file) in the user's local storage. On the other hand, the parameter `o` represents a specific document format's resource when the document to be visualised is already stored in the link service database. In order to illustrate the difference between these two cases, Listing 4.5 shows the implementation of the `openDocument()` method in a PDF visual plug-in. The PDF visual plug-in checks whether the received object (PDF document) is an instance of `File` or if it is already stored in the link service database and being an instance of `PDFResource` (i.e. the class contained in the PDF data plug-in and extending the RSL resource to define PDF resources). In both cases, the visual plug-in visualises the document.

The second parameter of the `openDocument()` method is the set of anchors contained within a document. As discussed earlier, the anchor object in our link service contains information about a hyperlink source which is either a selector in a document or a complete document. It further contains information about all the other sources and targets participating in the hyperlink. Listing 4.6 presents the definition of the anchor object in the link service. The targets and other sources can also be either documents or selectors in other documents. The anchor object contains enough information about each target or other sources such as its media type, its ID in the system as well as the contained document

in the case of a selector. This provides developers the flexibility to build rich visual plug-ins. As illustrated in Figure 4.11, multidirectional hyperlinks can be presented with a pop-up menu in order to give the user the flexibility to navigate to any hyperlink source or target.

```

1 public abstract boolean openDocument(Object o, HashSet <Anchor> anchors, Anchor
    anchor);
2 public abstract void highlightAnchors();
3 public abstract HashMap <Entity, String> getSelections();

```

Listing 4.4: A number of the abstract methods in the visualisation component

The third optional parameter of the `openDocument()` method contains a specific anchor to be highlighted in the case that the document has to be visualised as a result of navigating a hyperlink. We recommend that a visual plug-in should not automatically highlight a document's selectors but rather wait for a user request to do so. The `highlightAnchors()` method has been proposed in order to give the users the flexibility in visualising the document selectors. The `getSelections()` method is intended to retrieve new selectors created by the user in a document visualised via a visual plug-in.

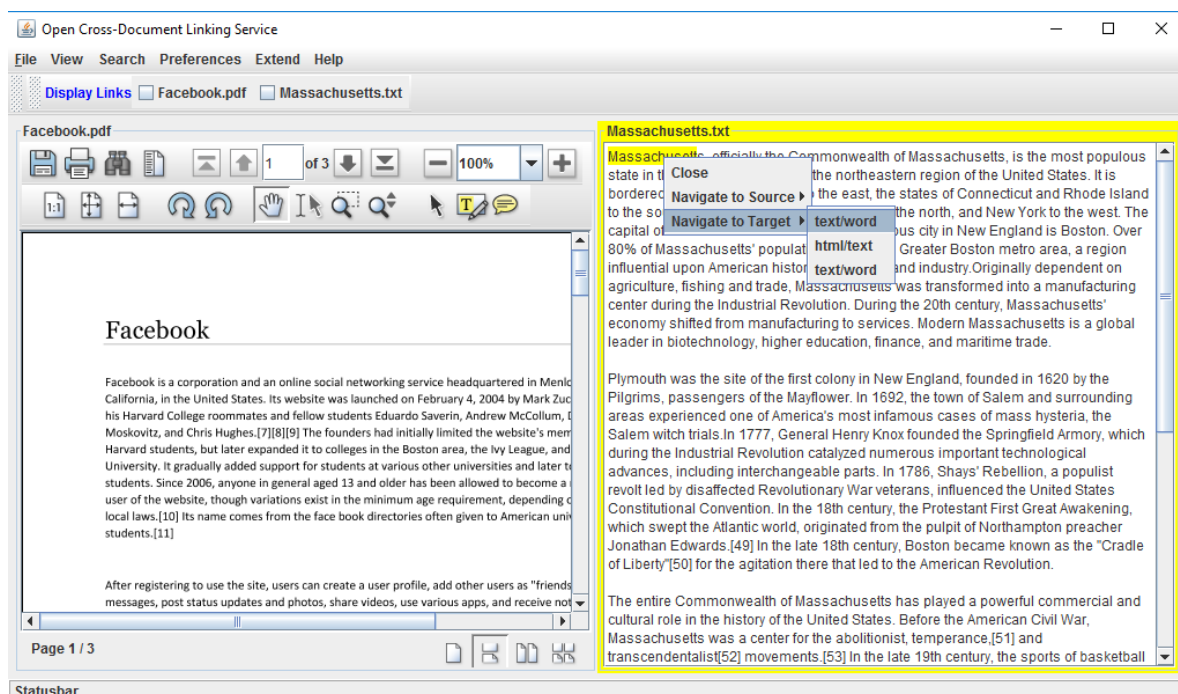


Figure 4.11: A rich visualisation of a multidirectional hyperlink by the plain text visual plug-in

Visual plug-ins are expected to extend our link browser by providing a concrete implementation of the abstract methods. The availability of many open source programming libraries or APIs for some document formats facilitates the implementation of visual plug-ins for the different document formats. For example, we have used the ICEpdf⁶ library to realise the PDF visual plug-in. The extensibility of our link browser is similar to the GUI extensibility of the Eclipse IDE where the user interface logic defines some extension points from some views in the GUI. Our approach is also similar to the Google Chrome extension mechanism where a Google Chrome add-in can choose to be either a browser action or a page action to extend the browser GUI.

```

1 @Override
2 public boolean openDocument(Object o, HashSet <Anchor> anchors, Anchor anchor) {
3     File file = null;
4     if (o instanceof File) {
5         file = (File) o;
6         this.selectedFile = file;
7     }
8     else if (o instanceof PDFResource) {
9         this.pdfResource = (PDFResource) o;
10        file = new File(pdfResource.getUri());
11    }
12    this.anchors = anchors;
13    this.anchorToHighLight = anchor;
14    return controller.openDocument(file.toString());
15 }

```

Listing 4.5: The implementation of the `openDocument()` method in a PDF visual plug-in

4.5.3 Requirements for Integrating a Document Format

As discussed, in order to integrate a document format with our link browser, a data as well as a visual plug-in for the corresponding document format have to be provided. Since our link service is based on the dynamic Java-based OSGi framework, both data and visual plug-ins must be implemented in Java. The implementation of a data plug-in for a given document format is not a complicated task since it only defines how to address the document format's resources (e.g. URI) and how to address its selectors (e.g. XPointer expression). The implementation of a

⁶<http://www.icesoft.org/java/projects/ICEpdf/overview.jsf>

visual plug-in for a given document format requires an open source Java-based library or a Java API for the document format in order to correctly visualise the document format. In the case that there is no available open source Java-based library or a Java API for a given document format, it is still possible to implement a Java library to visualise the document format from scratch. Nevertheless, we believe that the implementation of a Java library for a given document format is a complex and tedious task.

```

1  /**
2   * @param en : the selector or the entire document forming part of a link sources
3   * @param sources : other sources of the same link
4   * @param targets : targets of the selector, they are also targets of the sources
5  */
6  public Anchor(Entity en, HashSet <Entity> sources, HashSet <Entity> targets){
7      this.entity = en;
8      this.sources = sources;
9      this.targets = targets;
10 }

```

Listing 4.6: The definition of the anchor object in the link service

In order to successfully implement a visual plug-in for a given document format, its Java-based library or Java API should adhere to two different requirements. First, they must enable third-party developers to manipulate documents of the document format (e.g. create selectors or get selections). In the case that the open source Java-based library does not support document manipulation, it should be possible to extend the library to support this feature. Second, they must enable third-party developers to extend and customise the user interface that visualises documents of the document format. For example, a third-party developer should be able to add an item to the existing context menu enabling end users to navigate to hyperlink sources or targets. This feature should also facilitate the highlighting of a document's selectors. In the case that the open source Java-based library does not support the customisation of the user interface, it should then be possible to extend the library to support this feature.

4.5.4 Supported Document Formats

Our link browser currently supports the XML, plain text and PDF document formats as well as general multimedia content such as images via the corresponding data and visual plug-ins. Figure 4.12 illustrates the visualisation of a PDF and a text document as well as a bidirectional

hyperlink between them. We now elaborate on the plug-ins for the plain text document format. Please refer to Section 5.1 for more information about the integration of other document formats and multimedia content in our link browser.

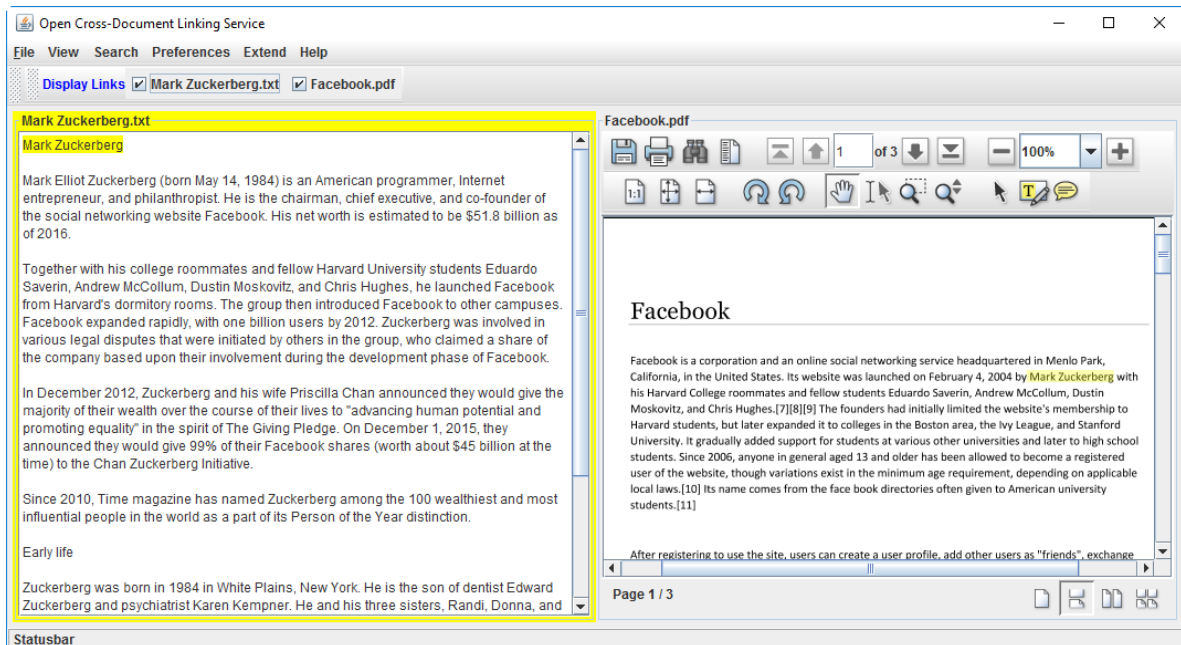


Figure 4.12: A bidirectional hyperlink between a PDF and a text document that are visualised in the link browser

The data plug-in for text documents defines its resources via the path and name of the documents in the user's local storage. Thereby, a user should be able to create bi- and multidirectional hyperlinks in text documents stored in their local storage. The selector within a text document is defined by a start and end index. In its manifest file illustrated in Listing 4.7, the text data plug-in declares its name, type and media type. It further exports its main package `org.rsl.text.data` to enable the corresponding visual plug-in to deal with text resources and selectors.

-
- 1 Import–Package: `org.rsl.core`
 - 2 Export–Package: `org.rsl.text.data`
 - 3 Extension–Name: `text`
 - 4 Extension–Mime: `text/plain`
 - 5 Extension–Type: `data`
-

Listing 4.7: Metadata in the manifest of the plain text data plug-in

As illustrated in Listing 4.8, the visual plug-in for text documents imports the main package of the data plug-in and defines the same value for its media type as the corresponding data plug-in. Using the **Extension-Class** metadata, it declares the Java class extending our link browser. The visual plug-in for text document relies on the Java Swing library. It uses a **TextPane** component to visualise arbitrary text documents. By using the **TextPane** API, the text visual plug-in creates and retrieves selectors. It further uses the **DefaultHighlighter** component of the `swing.text` package to highlight the selectors of text documents. Last but not least, as previously illustrated in Figure 4.11, the visual plug-in provides a context menu enabling users to navigate to hyperlink sources and targets.

```
1 Import-Package: org.rsl.core, org.rsl.userInterface.util, org.userinterface.localvisualplugins,
2   org.rsl.text.data
3 Export-Package: org.rsl.text.visual
4 Extension-Name: text
5 Extension-Mime: text/plain
6 Extension-Class: org.rsl.text.visual.TextPanel
7 Extension-Type: visual
```

Listing 4.8: Metadata in the manifest of the plain text visual plug-in

4.6 Integration of Third-Party Document Viewers

As discussed earlier, three things are required for integrating a third-party document viewer with our link service: a data plug-in for the corresponding document format, a gateway plug-in and an add-in for the third-party document viewer. The data plug-ins have been already presented in Section 4.5.1. In this section we elaborate on the add-ins, gateways and the communication between them. We end this section with a concrete example of integrating the Google Chrome web browser with our link service.

4.6.1 Third-Party Document Viewer Add-ins

The integration of documents visualised in their third-party document viewers in our link service asks for a mechanism that enables users to interact with the visualised documents in order to create, update, navigate

or search hyperlinks. The fact that different third-party document viewers have been developed in different programming languages for different platforms resulted in different SDKs and APIs for these applications. This also means that their add-ins have different architectural designs, programming languages or deployment procedures. Therefore, we only outline a number of guidelines for developers who wish to develop add-ins for third-party document viewers in order to integrate them with our link service. These guidelines are mainly concerned with the interactions required for the CRUD operations and the messages that have to be exchanged with the link service.

First of all, an add-in should provide a user interface that enables users to create, update as well as delete selectors. Furthermore, a document selectors should be visualised with a different colour than the document's original hyperlinks (e.g. HTML hyperlinks authored by the web page developer). Moreover, if a document is visualised as a result of a hyperlink navigation, the target selector has to be visualised with a different colour than the other document selectors. This enables users to easily spot and recognise the exact target of the navigated hyperlink. Furthermore, an end user should be able to enable/disable the visualisation (highlighting) of a document's selectors. In addition, enough information should be visualised about every selector's targets (e.g. name of the target document and its media type) as well as other hyperlink sources (e.g. name of the document and its media type). As mentioned earlier, targets of a selector as well as other hyperlink sources can be visualised with a pop-up menu in order to give the user the flexibility to navigate to any target document. Moreover, we recommend that an add-in should provide some search functionality for selectors and their targets. For example, a user might use a Google Chrome add-in to search for selectors in an HTML document that have targets to only other HTML documents.

Last but not least, an add-in should communicate with our link service through any communication channel provided by the link service. The JavaScript Object Notation (JSON)⁷ format has been chosen to represent the messages exchanged between the link service and add-ins. The JSON data interchange format facilitates the integration of any document viewer in a platform-independent manner. It is worth mentioning that other data interchange formats such as XML could be used in the link service. A developer of a third-party document viewer add-in should provide handles to marshal and unmarshal the JSON messages exchanged

⁷<http://www.json.org>

with the link service. The structure of the different messages and their essential parts are further detailed in Section 4.6.3.1.

4.6.2 Gateway Plug-ins

The gateway component contains an interface that provides the abstract methods needed to translate any message exchanged with a third-party document viewer add-in. Message translation simply means the marshalling and unmarshalling of Java objects. JSON messages sent by an external third-party document viewer add-in are unmarshalled into Java objects by the corresponding gateway and Java objects are marshalled to JSON objects by the gateway to be sent to its corresponding external third-party document viewer add-in. For a document format that has to be integrated with its third-party document viewer, a gateway extending the gateway component has to be provided. In a gateway plug-in, the developer has to provide a class that implements the gateway interface.

Two things are worth mentioning here. First, the gateway component can easily be extended by any document format gateway since third-party developers are not required to understand the different communication channels and processes among the link service components. Second, the link service does not provide a general JSON representation for messages that should be exchanged with external third-party document viewer add-ins, but it rather asks developers to form these objects. For multiple reasons, we provide developers the freedom to marshal and unmarshal the objects and send as much information (in JSON objects) as they want from the link service to third-party document viewers. First of all, the objects to be marshalled or unmarshalled represent information about documents and selectors in a specific document format. This information is introduced by the data plug-in of the document format and we cannot anticipate what information the object contains. Therefore, it is impossible to provide a general JSON object that is valid for all document formats. The link service treats all objects as entities which is the general representation of RSL resources and selectors. Nevertheless, third-party developers are aware that entity objects received by the gateway from other link service components must be instances of the document format. Therefore, they are able to retrieve any information they want from the different objects. Second, it enables developers to send any information to external third-party document viewer add-ins to provide rich hyperlink visualisations.

The gateway interface methods support all possible end-user interactions with third-party document viewer documents as well as other important functionality that is required by the link service. The possible end-user interactions are the following:

1. **Open a document in a third-party document viewer** A user wants to visualise a document in its third-party document viewer. The user can choose the document from the stored documents in the link service or follow a hyperlink in another document (i.e. in this case, the document is the target of the hyperlink). The user probably wants to visualise a document in a reading mode or with highlights.
2. **Create selector(s) in a document** A user selects parts of a document in order to create hyperlink sources or targets.
3. **Navigate to a hyperlink target** A user follows a hyperlink in a document visualised in its third-party document viewer.
4. **Update a selector** A user updates a specific document selector.
5. **Delete a selector** A user deletes a specific document selector.
6. **Update a document** A user updates a document (e.g. changes its URL or its file path).
7. **Delete a document** A user deletes a document.

Listing 4.9 illustrates a number of abstract gateway methods. The entire **Gateway** interface is included in Appendix C. The **openDocument()** method should enable the opening of a document in all the different scenarios. The first parameter of the function presents the document to be opened. Note that the first parameter of this method is different from the first parameter of the **openDocument()** method of the visual plug-ins. A user will use visual plug-ins to visualise documents that are not yet stored in the link service in order to create hyperlinks. As previously shown in Listing 4.5, the **openDocument()** method of a visual plug-in should accept a general parameter (i.e. Java Object) to deal with new documents (objects) and documents that are already stored in the link service (i.e. RSL resources). This is not the case with external documents visualised in their own third-party document viewers. The user does not

need the link service to visualise documents that are not yet stored in the link service (i.e. documents having no hyperlinks). A document that is stored in the link service is an instance of the RSL resource and thereby the `openDocument()` method of the gateway should only deal with RSL resources. The second and the third parameters of the `openDocument()` method are similar to those in the `openDocument()` method of the visual plug-ins. The `openDocument()` method should return a JSON message to be sent to the corresponding add-in. The JSON message should contain enough information about the document and its anchors.

```
1 public abstract JSONObject openDocument(Resource res, HashSet <Anchor> anchors,  
2   Anchor entityToHighlight);  
3 public abstract Resource getResource(JSONObject command);  
4 public abstract long getTargetEntityID(JSONObject command);  
5 public abstract void launchApp();
```

Listing 4.9: Some of the abstract gateway methods

In most cases, JSON messages sent by a third-party document viewer add-in contain information about the document being interacted with. Therefore, the link service asks the corresponding gateway to transform the JSON data about the document to a Java object (i.e. RSL resource). The `getResource()` method takes a JSON message coming from the corresponding add-in and returns the exact RSL **Resource** (document). Let us illustrate the use of the `getResource()` method with an example. A user opens an HTML document in the Google Chrome browser. They select a button in the Google Chrome add-in in order to retrieve the defined anchors (if any) from the link service for the document. The add-in sends a JSON message with information about the document (e.g. its URI) to the link service requiring its anchors. The link service can only deal with RSL resources and therefore it passes the JSON message to the `getResource()` method of the corresponding gateway. The developer of the gateway knows the exact JSON message structure sent by the corresponding Google Chrome add-in. Thereby, they can easily unmarshal (part of) the JSON message to an RSL **Resource**. The link service then checks whether there are any anchors defined for the document. Using the `openDocument()` method, the link service passes the document and its retrieved anchors to the gateway. The returned JSON message is then forwarded to the Google Chrome add-in. The add-in should finally update the document being opened and visualise its anchors.

When a gateway constructs a JSON message for opening a document in the corresponding third-party document viewer, it must send the ID of each selector's target or other hyperlink sources. It should send enough

information about each of them such as their media type or the contained documents (in the case of selectors) in order to provide rich hyperlink visualisation. When the user navigates to any hyperlink source or target, the add-in will use its ID value to form a JSON message. The add-in sends a request (i.e. the JSON message containing the target ID) to the link service. The link service then asks the corresponding gateway to retrieve the ID value of the intended target via the `getTargetEntityID()` method. The link service uses the returned ID in order to retrieve the target document and then visualises it.

4.6.3 Communication Channels

In order to support a large range of third-party document viewers, our link service communication component supports various communication protocols (R4). Full duplex communication protocols are the optimal communication channels for our link service since the link service and third-party document viewer add-ins both send (push) messages to each other. Thereby, the communication component of the link service offers two channels for the full duplex communication; TCP sockets as well as WebSockets. A RESTful API is further offered by the communication component in order to be used as a fallback solution for third-party document viewer SDKs not offering full duplex communication. JSON messages coming from third-party document viewers through different communication protocols are managed centrally via the message pool component. The message pool also keeps track of all active third-party document viewer add-ins and their sessions in order to forward the JSON messages produced by different gateways to the correct communication protocol.

Messages coming from third-party document viewer add-ins are forwarded to the user interface component via the message pool. Before this can happen, the messages have to be unmarshalled to Java objects. Therefore, the message pool asks the document format gateway to unmarshal the message. The message pool can identify the correct gateway by using the media type defined by the communication session in the handshake process. Moreover, when a message has to be sent from the link service to the external document viewer add-in, the message pool asks the document format gateway to marshal the message (Java object) to a JSON message before forwarding it to the correct communication protocol with the active session. Thereby, the communication protocols

are only responsible for sending and receiving messages without any further processing.

4.6.3.1 Messages

As mentioned before, there are various types of interactions with the link service as well as the third-party add-ins (e.g. creating a selector, navigating to a hyperlink target or opening a document). In order to correctly execute these interactions, each message exchanged between the link service and add-ins must specify the type of action expected to be executed. For instance, a user might have clicked on a specific hyperlink target visualised in the third-party document viewer and is waiting for visualising the target document or a user might have updated a selector in the third-party document viewer and is waiting for a confirmation from the link service.

Each JSON message coming from an add-in must contain a **command** key with one of the predefined request values. These predefined request values are **updateResource**, **updateSelector**, **deleteResource**, **deleteSelector**, **addLinkAnchors**, **isOpen** and **showTarget**. The last request value (i.e. **showTarget**) must be contained in JSON messages that inform the link service that a user clicked on a hyperlink target in a document visualised in the third-party document viewer. As mentioned earlier, the JSON message containing the **showTarget** request value should also include the ID value of the intended hyperlink target. When a user opens a document in its third-party document viewer and asks the add-in to retrieve the document anchors from the link service, the add-in must include the **command** key with the **isOpen** value in the JSON message. The **isOpen** value informs the link service to retrieve the document anchors and returns them back to the add-in. The **addLinkAnchors** informs the link service that a user selected one or more selectors in a document. The link service then asks the corresponding gateway to return the selector(s) contained in the JSON message. The selector(s) will be added to the hyperlink overview menu shown earlier in Figure 4.3. The rest of the request values inform the link service that a user updated or deleted a document or a selector. The link service communicates with the corresponding gateway plug-in to return the **RSL Resource** or **RSL Selector**. The link service then updates or deletes the document or the selector and sends a notification to the add-in.

Communication channels are responsible for checking the request values coming from add-ins. Based on a request value, a communication channel forwards the message along with the add-in media type to the message pool component which in turn handles the request by communicating with the corresponding gateway.

Messages formed by a gateway and sent to its corresponding add-in should also contain request values. However, we do not force developers to use a specific key and value since the link service will not process these messages. Developers are given some flexibility in naming their request keys and values. For instance, a JSON message returned by the `openDocument` method of a Microsoft Word gateway can contain `Wordcommand:openWordDocument` while another JSON message returned by the same method of a Google Chrome gateway can contain `command:openWebpage`. The important thing that developers should assure is that their corresponding add-ins understand the requests and can process the messages. In our example, this means that the Microsoft Word add-in has to know that `Wordcommand:openWordDocument` is a request for visualising a document with its anchors while the Google chrome add-in must know that `command:openWebpage` is a request for visualising an HTML document with its anchors.

4.6.4 Requirements for Integrating a Document Viewer

As discussed, in order to integrate a third-party document viewer with our link service, a data plug-in, a gateway plug-in and an add-in for the third-party document viewer have to be provided. With our proposed approach for integrating third-party document viewers, a given third-party document viewer can be integrated with our link service when four conditions are met. First of all, the third-party document viewer should be extensible via add-ins (sometimes called add-ons or plug-ins). Second, the SDK of the third-party document viewer must enable third-party developers to manipulate documents in order to create or get selectors within its supported documents. Third, the SDK of the third-party document viewer must enable third-party developers to extend and customise the third-party document viewer's user interface. This feature should enable third-party developers to provide visual handles in the developed add-ins. For example, a developed add-in can customise the context menu of the third-party document viewer by adding a new item (icon)

that enables end users to navigate to hyperlink sources or targets. In addition, this feature should facilitate the highlighting of a document's selectors. Finally, the SDK of the third-party document viewer must support the communication to other third-party applications (i.e. our link service).

4.6.5 Supported Document Viewers

Our link service currently supports three different document viewers including Google Chrome, Microsoft Word and Microsoft PowerPoint. Two add-ins have been developed for Google Chrome. The first add-in enables the linking to arbitrary web pages (i.e. HTML selectors) while the second add-in enables the linking to timespans in YouTube videos. As explained in Section 5.1, the second add-in illustrates the flexibility of our link service in supporting not only document formats but also other media types. We elaborate on the different plug-ins developed for the integration of the HTML document format. Please refer to Section 5.1 for more information about the integration of the other document viewers.

4.6.5.1 Google Chrome Add-in

Google Chrome is extensible via add-ins. It provides a powerful API for developing add-ins using the HTML5, CSS and JavaScript web technologies. A Google Chrome add-in is a zipped file containing JavaScript, HTML, CSS, images and any other essential files needed to add extra functionality to the Google Chrome web browser. An add-in must contain a manifest file that provides information about the extension such as its name, version and the Google Chrome browser capabilities (e.g. permissions to access specific websites) that it might use.

The developed add-in for Google Chrome is written using a combination of HTML5, CSS3 and JavaScript. HTML5 and CSS3 are used to realise the add-in user interface visualised on the right hand-side of the Google Chrome web browser as previously illustrated in Figure 4.6, whereas JavaScript is used to implement the logic of the add-in and to communicate with our link service. The add-in makes use of the `WebSocket` JavaScript object in order to connect to our link service. It further uses Rangy⁸, a cross-browser JavaScript library, to create, get and

⁸<https://github.com/timdown/rangy>

highlight selectors within web pages. The add-in customises the Google Chrome context menu by adding new commands for creating selectors and navigating to hyperlink sources and targets.

4.6.5.2 HTML Document Format Plug-ins

Two different plug-ins have been developed for the HTML document format, a data and a gateway plug-in. The HTML data plug-in defines its resources with URIs. The selector within an HTML document is defined with an XPointer-like expression as implemented by the Rangy library. On the other hand, the gateway plug-in implements all the methods of the gateway interface. It forms JSON messages to be sent to the corresponding Google Chrome add-in. It also unmarshalled the JSON messages coming from the corresponding add-in to HTML resources and selectors.

4.7 Discussion

The presented cross-document link service goes beyond the simple annotation concept offered by most annotation tools where only the reading, creating, saving, updating and retrieving of annotations is supported while support for bi- and multidirectional hyperlinks between document content is missing. Furthermore, existing as well as emerging document formats can be integrated in our link service regardless of their document models. Moreover, in contrast to some existing link services, our link service offers a flexible mechanism for integrating third-party document viewers. To the best of our knowledge, the presented link service is the first prototype to introduce flexibility and extensibility on the model as well as the information visualisation layer as proposed by Signer and Norrie [126, 125]. We believe that our extensible cross-document link service is a future-proof linking solution for arbitrary document formats and multimedia content types. Its dynamic extensibility further allows third-party developers and end users to support any document format and multimedia content types without the intervention of the link service provider. To address the preferences of end users, the integration of different document formats and multimedia content types can either happen within the link browser or in their preferred third-party document viewers. The presented dynamically extensible link service further deals

with updates for document format specifications or third-party document viewers by providing a mechanism for maintaining different versions of the same document format plug-in. We believe that the presented cross-document link service forms an ideal platform for investigating innovative forms of cross-media linking.

Five features were essential in achieving the presented prototype of a cross-document link service. First, our cross-document link service uses an external link representation and storage like early hypermedia systems such as Sun's Link Service [112] and Microcosm [73]. This implies that there are no changes necessary to the specification of existing document formats in order to integrate them with our link service. Second, through generalisation and the treatment of hyperlinks as first-class objects in the core link service (RSL), each document format to be supported can extend the RSL resource for its own logical document model definition and specialise the RSL selector with a definition of its selector. Third, the proposed visual plug-in extensibility of the link browser enables the integration of existing as well as emerging document formats in the link browser. Fourth, the proposed mechanism for integrating third-party document viewers enables the integration of extensible third-party document viewers without any required changes to its core. Finally, the dynamic modular OSGi framework enables the dynamic extensibility of the link service via the plug-in tracking mechanism as well as the plug-in metadata.

The presented link service overcomes the limitations of existing link services since it takes into account the requirements for an ideal link service discussed earlier in Section 2.8. Moreover, we believe that our link service meets end-user requirements since it takes into account most of the design implications outlined in Section 3.5. In the following, we discuss how our link service meets the different requirements and design implications.

1. Requirements for an ideal link service

- (a) **R1 flexible and extensible link service architecture** Most of the link service components are extensible to integrate existing and emerging document formats as well as third-party document viewers. The link service's link model is based on the RSL metamodel which is flexible and extensible to support any media type. The link browser is also extensible via visual plug-ins.

- (b) **R2 support multiple document formats** Due to the flexibility and extensibility of our link service, we have already supported the linking across six different document formats. Thanks the extensibility of the link browser via visual plug-ins and the extensibility of the gateway component.
- (c) **R3 easy integration of third-party document viewers** We have proposed a simple mechanism to integrate extensible third-party document viewers in our link service. No changes are required to the core of extensible third-party document viewers willing to benefit from our link service features. By using the SDKs of third-party document viewers, add-ins that communicate with our link service via their corresponding gateways about selectors to be created or highlighted can easily be developed. We have already integrated three different document viewers with our link service.
- (d) **R4 flexible communication channels** The link service has a flexible and extensible communication component which currently supports three different communication channels. WebSockets and TCP sockets can be used for full duplex communication channels whereas a RESTful API can be used as a fall back solution for third-party document viewer SDKs not offering full duplex communication.
- (e) **R5 customisable link service** An online repository managing the different plug-ins has been developed. End users are able to customise the link service by installing only the document formats that are really needed.
- (f) **R6 plug-in versioning** The plug-in tracking component keeps track of any updates to the already installed plug-ins by communicating with the online repository. The plug-in tracking allows end users to update any existing document format plug-in by installing a new version of the plug-in.

2. Design implications for a linking solution

- (a) **DI1 granularity of the associations** The link model (RSL) of our link service enables to the linking between documents at any level of granularity. A data plug-in of a specific document format can define its selectors to be a paragraph while another can define its selectors to be single words. Currently most of the

link service's supported document formats allow users to establish hyperlinks in their documents at any level of granularity. For example, the data plug-in for plain text defines its selectors as start and end indices. In other words, a user can establish hyperlinks in the supported text documents at any level of granularity (e.g. at character, word, sentence, paragraph or document level).

- (b) **DI2 bidirectional associations** Thanks to the RSL hypermedia metamodel, the link service not only supports bidirectional hyperlinks but also multidirectional hyperlinks.
- (c) **DI3 documents side by side** Our link browser visualises documents side by side. It also gives the user the flexibility to disable this feature and visualise only one document on the entire user interface.
- (d) **DI5: management of the associations** Currently the link service provides users some simple search features enabling them to search for specific hyperlink targets and sources.

4.8 Summary

In this chapter, we presented a dynamically extensible cross-document link service which overcomes the shortcomings of existing link services and meets end-user requirements for a linking solution. The link service offers a plug-in architecture where different document formats as well as third-party document viewers can be integrated via a plug-in mechanism. The link model of our link service is based on the RSL hypermedia metamodel. The RSL metamodel has been introduced to overcome some limitations of existing hypermedia models and systems and to be general and flexible enough to be used in evolving hypermedia systems. RSL supports advanced hyperlinks such as bi- and multidirectional hyperlinks. In our link service, new document formats are supported by implementing data plug-ins that extend the RSL resource and selector concepts and contain information on how to address resources (documents) as well as selectors attached to documents in a given document format. The visualisation component of our link service consists of a link browser for visualising the supported document formats. The user interface further offers the necessary GUI actions to perform the basic CRUD operations on hyperlinks. For each document format to be visualised in the link browser, a visual plug-in has to be implemented. A visual plug-in for

a given document format needs to visualise documents as well as their selectors and has to provide the necessary functionality to create, delete and update selectors. Our link service's link browser currently supports the visualisation of three different document formats including plain text, PDF and XML.

The presented link service further addresses the challenge of seamlessly integrating third-party document viewers in order to enable users to continue using their preferred third-party document viewers while being able to link the different documents. The concept of an extensible gateway component that facilitates the communication between our link service and third-party document viewers has been introduced. For every document format to be integrated via its third-party document viewer, a specific document format gateway extending the gateway component has to be provided. Moreover, an add-in for the third-party document viewer needs to be developed in order to enable users to highlight and create selectors in documents visualised in the third-party document viewer. The add-in should communicate with the link service about selectors to be created or highlighted via the corresponding gateway. A data plug-in extending the RSL metamodel and containing information on how to address resources and selectors of the documents visualised in the third-party document viewer also has to be provided. An add-in of a third-party document viewer can communicate with the link service by using any communication channel provided by the flexible and extensible communication component of our link service. The link service currently supports the linking to documents visualised in three different third-party document viewers including Microsoft Word, Microsoft PowerPoint and Google Chrome.

The presented link service allows third-party developers to support any document format or third-party document viewer by developing the required plug-ins. Moreover, it allows end users to support any document format by installing the required plug-ins from an online repository, without the intervention of the link service provider or third-party developers. Last but not least, the presented link service exploits the dynamic modular framework (OSGi) for its dynamic extensibility as well as the management and tracking of different plug-ins.

5

Evaluation

In this chapter we present three evaluations of our link service. In a first technical evaluation, we validate the extensibility of our link service by discussing its support for a number of document formats and various third-party document viewers. In a second technical evaluation, we further validate the extensibility of our link service by investigating the potential for integrating other existing document formats and document viewers with our link service. In the third evaluation, we investigate the usability of the presented link service in an end-user study.

The chapter is structured as follows. In Section 5.1 we present our first evaluation, whereas the second evaluation is presented in Section 5.2. In Section 5.3, we discuss the results of the end-user study. A conclusion and a summary of the different evaluations is provided in Section 5.4.

5.1 Supported Document Formats and Viewers

Our link service's extensible plug-in architecture allowed us to integrate a number of document formats and document viewers as well as two multimedia content types. The link service currently supports the linking of PDF, XML, plain text, HTML, Word and PowerPoint document formats

as well as images and YouTube videos. The link browser is further able to visualise PDF, XML and plain text documents as well as images via visual plug-ins. The HTML document format as well as YouTube videos have been supported by developing two different add-ins for Google Chrome. The integration of Microsoft Word and Microsoft PowerPoint with the link service via two different add-ins allows the linking to Word and PowerPoint documents. In the remainder of this section we elaborate on the integration of the different document formats, document viewers and multimedia content types.

5.1.1 PDF Document Format

The data plug-in for PDF documents specifies its resource via the path and name of the document in the user's local storage. The selector within a PDF document is defined by a page index and a rectangular area within a page. The PDF data plug-in defines its media type `application/pdf` in its manifest file.

The visual plug-in for PDF documents relies on the existing ICEpdf¹ open source Java library for visualising PDF documents in the link browser. Normally, the ICEpdf library visualises PDF documents in a JFrame. Therefore, we had to customise the ICEpdf library in order to be able to extend the link browser and visualise the PDF documents within a JPanel. The visual plug-in further uses the ICEpdf library API to get and create selections of rectangular areas in PDF documents. It is also worth mentioning that we have customised some methods in the ICEpdf library in order to be able to create interactive hyperlinks. The visual plug-in successfully implements all the abstract methods provided by the link service visualisation component.

5.1.2 XML Document Format

Similar to the PDF data plug-in, the XML data plug-in defines its resource via the path and name of the document in the user's local storage. A selector within an XML document is defined via DOM ranges. Note that there are also some libraries such as XInclude² that use XPointer, but these libraries are targeting XML inclusions. The XML visual plug-in

¹<http://www.icesoft.org/java/projects/ICEpdf/overview.jsf>

²<https://www.w3.org/TR/xinclude>

extends the `StyleEditorKit` that forms part of the `javax.swing.text` library for a better visualisation of XML documents. Furthermore, it uses the `javax.xml.parsers` library for reading XML documents. Last but not least, the XML visual plug-in applies the `org.w3c.dom` library to retrieve and highlight nodes and ranges within an XML document.

5.1.3 Images

In order to illustrate the flexibility and extensibility of our link service, we support the linking to areas within general multimedia content types such as images and YouTube videos. We support the visualisation of images within our link browser by using a visual plug-in. Figure 5.1 illustrates a bidirectional hyperlink between a PDF document and a JPEG image. The visual plug-in for images makes use of the Java Swing library in order to render the images. The visual plug-in further extends the Swing `JComponent` object in order to enable users to create rectangular shapes within images. The data plug-in for images defines an image resource via its path and name in the user's local storage. A selector within an image is defined as rectangular shape.

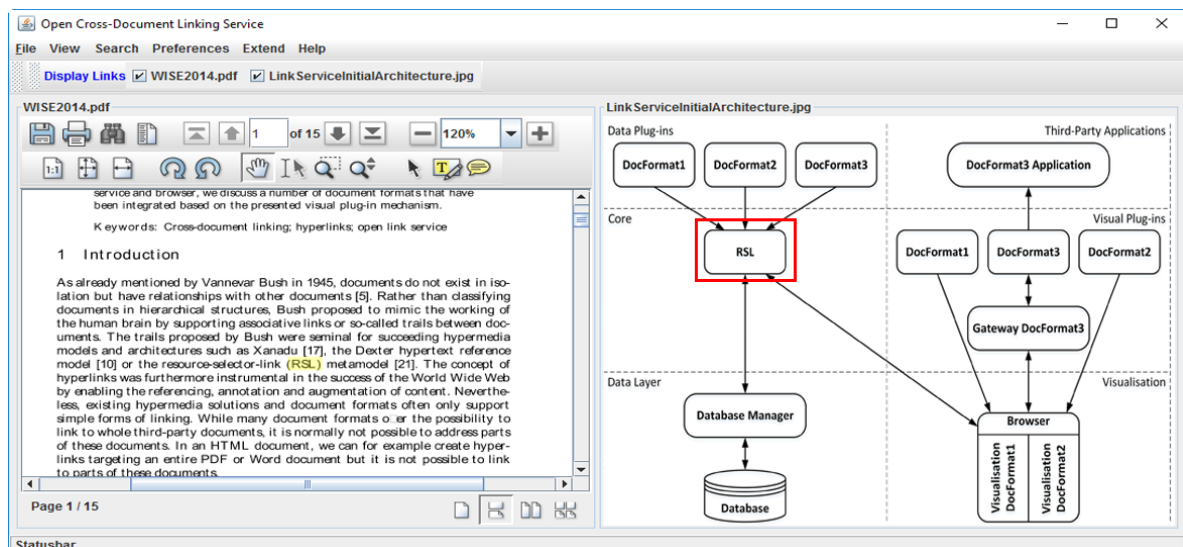


Figure 5.1: Bidirectional hyperlink between a PDF document on the left and a JPEG image on the right

5.1.4 Microsoft Word

Microsoft office applications (e.g. Microsoft Word) are extensible via add-ins. Add-ins can be developed either by using C# or JavaScript API. In contrast to the former C# API, JavaScript-based add-ins are platform independent. In the beginning, we have developed a JavaScript-based add-in. Even though the add-in provides most of the required functionality, it had a number of limitations (e.g. cannot customise Microsoft Word context menu) due to the limited capabilities of the JavaScript-based API. Therefore, we had to develop another add-in using the C# API. In the following, we elaborate on the JavaScript-based add-in and its limitations before describing the second C# add-in.

5.1.4.1 JavaScript-based Add-in

A JavaScript add-in can either be a content or a task pane application. A content application implies that the add-in, including its UI, will reside in the document itself, whereas a task pane application resides next to the document in a sidebar. The add-in for integrating Microsoft Word should not hinder a user's interactions with their Word documents. Therefore, we have chosen for the less obtrusive add-in by using the task pane application.

Normally, a simple JavaScript-based add-in includes two basic components, an XML manifest file and a web page. The XML file contains various required settings while the web page implements the logic of the add-in. The JavaScript API allows a developed task pane add-in to interact with a Word document's content to, for example, retrieve any selections. In order to preserve the privacy of users as well as the security of the host office application, a JavaScript-based add-in runs in a sandboxed web browser instance. Moreover, the runtime environment of the JavaScript-based add-ins facilitates their installation and uninstallation process since they do not require any executable file or dynamic link library (.dll).

The add-in is written using a combination of HTML5, CSS3 and JavaScript. HTML5 and CSS3 are used to realise the add-in user interface visualised in the task pane, whereas JavaScript is used to implement the logic of the add-in and to communicate with the link service. The JavaScript logic is heavily based on the asynchronous programming paradigm. The add-in user interface, shown in Figure 5.2, allows the

user to connect with the link service. The user can also list the available anchors and navigate to any specific hyperlink target.

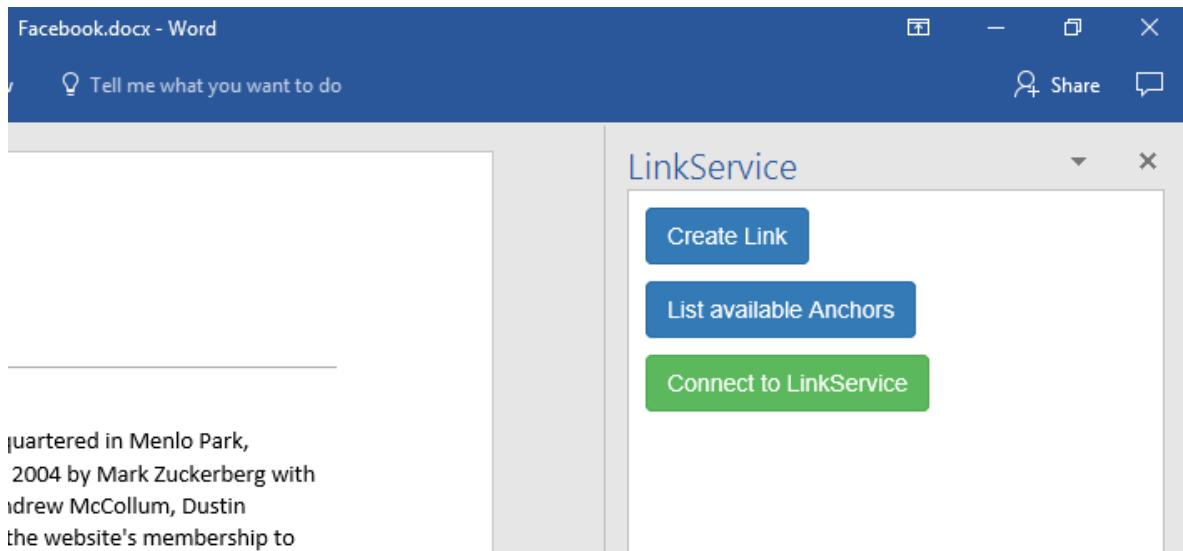


Figure 5.2: Microsoft Word user interface of the JavaScript add-in

The add-in makes use of the `WebSocket` JavaScript object in order to connect to the link service. Once the connection with the link service is successfully established, the selectors of the visualised document are retrieved which enables the user to update, navigate and delete them. It is worth mentioning that a selector in a Word document is represented by using the Office bindings which can be used in Word as well as Excel applications to reference a text selection in a document. In order to correctly position a selector with a Word document, a save operation has to be performed after the creation of a new selector.

As mentioned earlier, the add-in has a number of limitations. First of all, the highlighting of selectors is not possible with the current JavaScript API. When a user navigates to a specific selector, the add-in forwards the cursor next to the target selector without being able to highlight it. Furthermore, with the JavaScript API is not possible to customise the context menu of Microsoft Word. One last drawback is that the add-in cannot be loaded in Microsoft Word unless there is a document opened. This limitation prevents any request from the link service to open a document in Microsoft Word when Microsoft Word is closed.

5.1.4.2 C# Add-in

The C# API enables add-ins to customise almost every aspect of Microsoft Word. A developed add-in can customise the context menu, create selections, highlight selections, create toolbar menus and create multiple new windows within Microsoft Word. Using the API, we were able to develop a rich add-in that overcomes all the limitations of the previously described JavaScript add-in. As illustrated in Figure 5.3, the add-in provides a user interface on the right-hand side of the visualised documents. It enables users to connect to the link service via a TCP socket connection as well as to highlight or disable the highlighting of selectors. The add-in further allows users to create selectors by customising the context menu of Microsoft Word and adding the **Add Selector** command to it. By using the customised context menu, users can further navigate to hyperlink sources or targets.

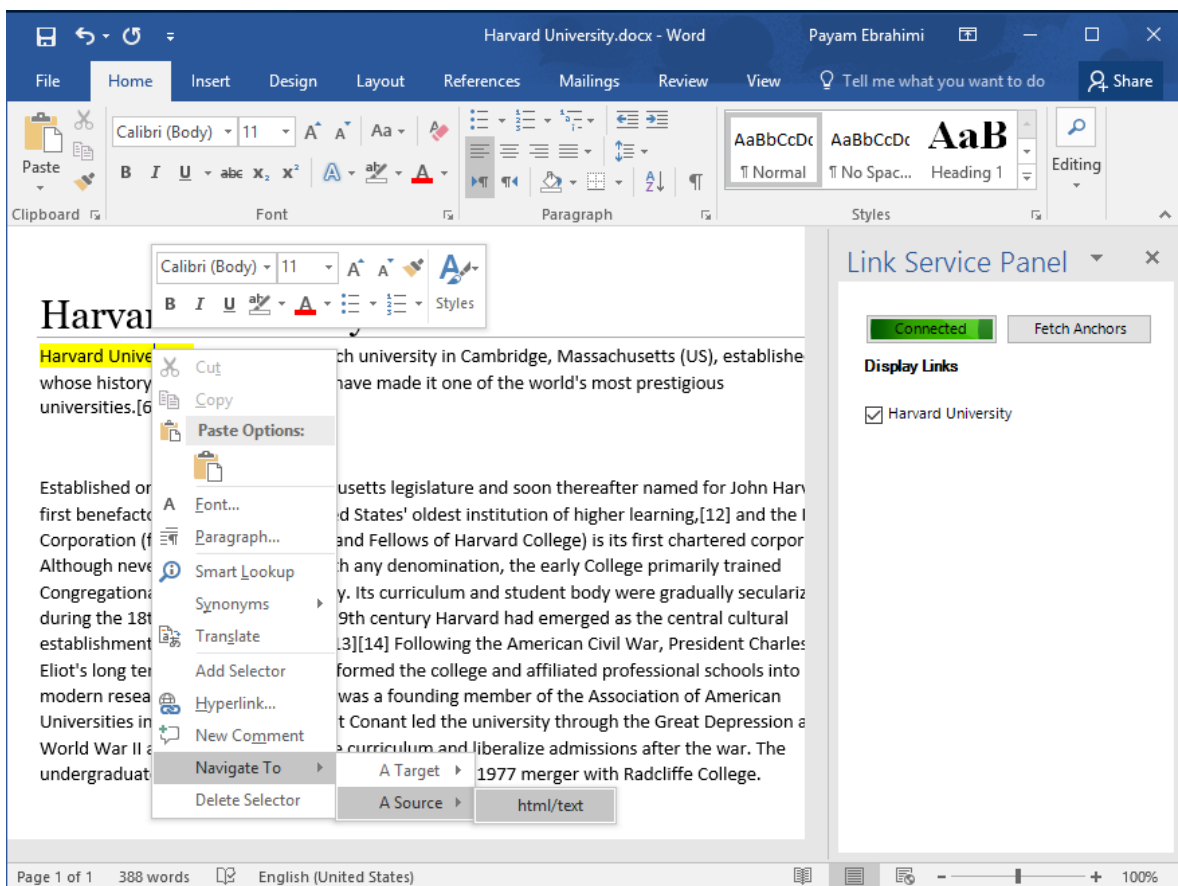


Figure 5.3: Microsoft Word user interface of the C# add-in

5.1.4.3 Word Document Format Plug-ins

Two different plug-ins are installed in the link service for the Word document format; a data and a gateway plug-in. The data plug-in defines its resources via the document name and path in the user's local storage. The selector within a Word document is defined as an XPointer-like expression. On the other hand, the gateway plug-in implements all the methods of the gateway interface. It forms JSON messages to be sent to the corresponding add-in. It also unmarshalled the JSON messages coming from the corresponding add-in to Word resources and selectors.

5.1.5 Microsoft PowerPoint

5.1.5.1 Microsoft PowerPoint Add-in

We have integrated Microsoft PowerPoint with a JavaScript-based add-in which has the same limitations of the Microsoft Word JavaScript-based add-in. The interface of the add-in is further identical to the Microsoft Word add-in interface illustrated in Figure 5.2. One thing that should be noted is that the Office bindings which reference text selections cannot be used in Microsoft PowerPoint. We gave the user the possibility to link to a specific slide or a range of slides (e.g. from slide 1 to slide 4).

5.1.5.2 PowerPoint Document Format Plug-ins

The data plug-in for PowerPoint identifies a PowerPoint resource via its path and name. It further defines a PowerPoint selector through an XPointer-like expression. The gateway plug-in for PowerPoint implements all the methods of the gateway interface. It forms JSON messages to be sent to the corresponding add-in. It also unmarshalled the JSON messages coming from the corresponding add-in to PowerPoint resources and selectors.

5.1.6 Google Chrome

In Section 4.6.5, we have already discussed the integration of the HTML document format that is visualised in Google Chrome. In this section, we discuss the integration of YouTube videos visualised with Google Chrome. We support the linking to timespans within YouTube videos

based on a simple add-in for Google Chrome as well as a data and gateway plug-in for our link service.

5.1.6.1 Google Chrome Add-in

Figure 5.4 illustrates the user interface of the add-in. The add-in communicates with our link service via WebSockets. The developed add-in only works if the YouTube website is completely loaded. After a YouTube page is loaded, the add-in calls the YouTube player which is embedded in an HTML `div` element called `player-api`. After grabbing the YouTube player, the add-in is able to access the HTML5 `<video>` object which offers several functions and attributes. Using these functions our add-in is able to create and play timespans within YouTube videos.

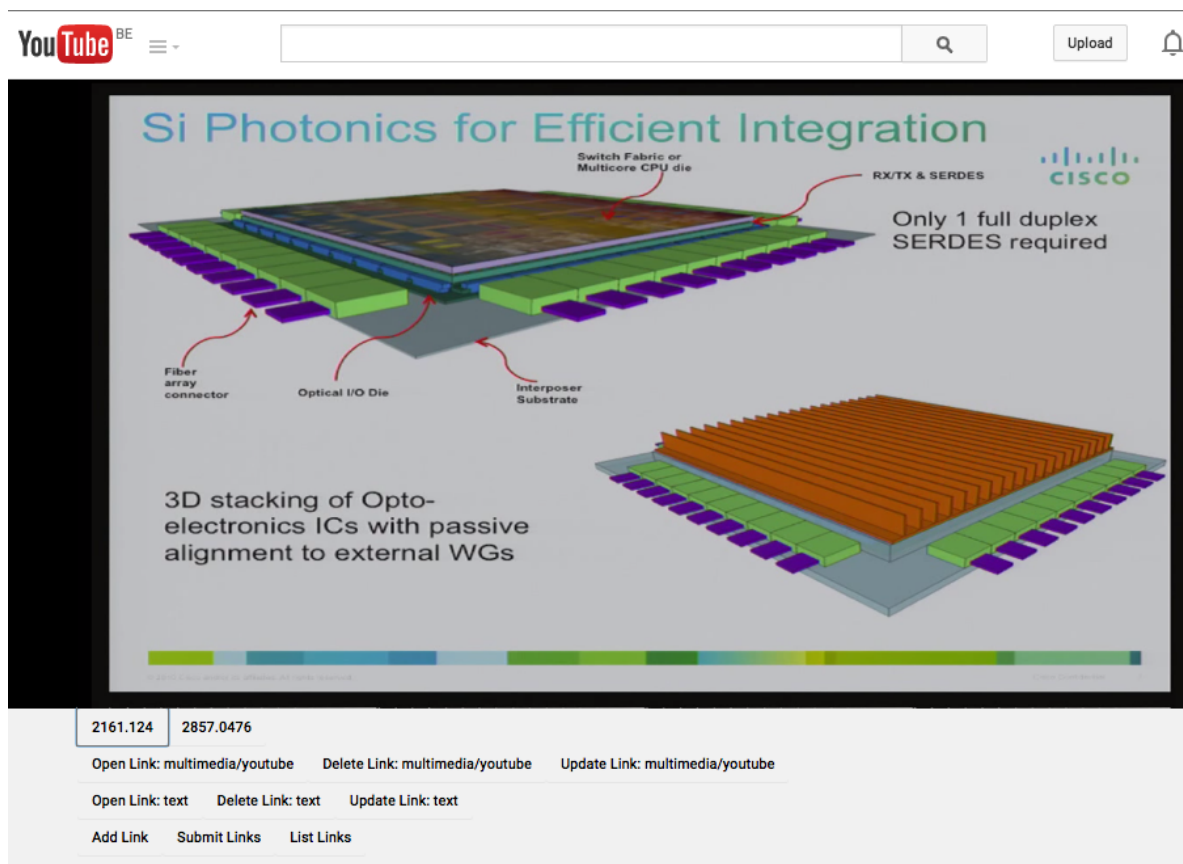


Figure 5.4: Google Chrome add-in enabling the linking to timespans within YouTube videos

5.1.6.2 YouTube Video Plug-ins

As mentioned, we have developed two different plug-ins for YouTube videos; a data and a gateway plug-in. The data plug-in defines a YouTube resource via its URI while the selector is defined using a start and end time. On the other hand, the gateway plug-in implements all the methods of the gateway interface. It forms JSON messages to be sent to the corresponding add-in. It also unmarshalled the JSON messages coming from the corresponding add-in to YouTube resources and selectors.

5.1.7 Discussion

The integration of a number of document formats and third-party document viewers serves as an assessment of our link service's extensibility. The flexibility of RSL enabled us to define a wide range of different resource-specific selectors such as timespans, XPointer-like expressions or rectangular shapes. The Java Swing library further allowed us to integrate some document formats and multimedia content types such as XML and images. We believe that by using the Java Swing as well as other existing Java libraries we can also integrate other document formats such as the HTML document format in the link browser. In contrast to XML and plain text which have been integrated based on the Java Swing library, the integration of the PDF document format was achieved by using the ICEpdf open source Java library. Even though the used library was not sufficient to integrate the PDF document format and support the required functionality, we were able to extend the library in order to develop a rich visual plug-in for the PDF document format.

Our proposed approach for integrating third-party document viewers proved its flexibility and extensibility for integrating third-party document viewers. For example, the flexible communication channels of our link service allowed Google Chrome to communicate via the WebSocket communication channel. On the other hand, it enabled the C# add-in for Microsoft Word to communicate with the link service via TCP sockets. The development of add-ins for the different third-party document viewers depends on the available SDKs for the third-party document viewers. For instance, we managed to develop two different add-ins for Microsoft Word by using the two available APIs.

5.2 Integration of Existing Document Formats and Third-Party Viewers

We now present a technical evaluation regarding the extensibility of our link service. In this evaluation, we investigate whether a number of existing document formats and third-party document viewers can be integrated with our link service.

5.2.1 Methodology

As discussed in Section 4.6.4, a given third-party document viewer can be integrated with our link service when four different requirements are met. The document viewer must be extensible via add-ins. Furthermore, its SDK must enable document manipulation in order to facilitate the creation of selectors. Moreover, its SDK must enable the customisation of the user interface of the third-party document viewer in order to enable the introduction of new GUI actions or the highlighting of a document selectors. Finally, its SDK must enable the communication with our link service. In the light of these four requirements, we investigate seven different third-party document viewers including Adobe Acrobat Reader, Foxit Reader, OpenOffice, the Opera web browser, the Firefox web browser, iBook Author and Sumatra PDF.

As discussed in Section 4.5.3, a given document format can be integrated and visualised in our link service by using an open source library or a Java API for the document format. In the absence of libraries for the document format, one can develop an entire library for the document format. In order to successfully integrate a document format in our link service, its open source library or Java API must meet two requirements. First, it must enable the manipulation of documents in order to facilitate the creation of selectors within documents. Furthermore, it must enable the customisation of the user interface in order to enable the introduction of GUI actions or the highlighting of selectors. In the light of these two requirements, we investigate whether OOXML, ODF, EPUB and IBA can be integrated and visualised in our link service. Please note that we only investigate existing libraries for the aforementioned document formats. In the case where there are no available libraries for an investigated document format, we will not discuss the development of libraries to support its integration in our link service.

5.2.2 Results

5.2.2.1 Third-Party Document Viewers

Table 5.1 summarises the results of our investigation of different third-party document viewers. In the table, we use the ✓ symbol to illustrate that a feature is supported or that the third-party document viewer can be integrated with our link service. As shown, most investigated third-party document viewers can potentially be integrated with our link service. Adobe Acrobat Reader is extensible via plug-ins using its Acrobat SDK³. Plug-ins for Adobe Acrobat Reader can be developed using either C, C++ or the JavaScript programming language. The Acrobat SDK allows developers to build rich and interactive plug-ins for Adobe Acrobat Reader. It further allows them to manipulate PDF documents and extend or modify Adobe Acrobat Reader toolbars and enables the communication with other third-party applications (e.g. our link service) via TCP sockets or WebSockets when C++ is used or via WebSockets when JavaScript is used.

Similar to Adobe Acrobat Reader, Foxit Reader is also extensible via plug-ins that are developed using the Foxit PhantomPDF SDK⁴. Foxit PhantomPDF plug-ins are based on the C++ programming language. Foxit PhantomPDF enables developers to modify and extend the existing Foxit Reader user interface. It further offers two different modules for manipulating and adding extra annotations to PDF documents. The communication with third-party applications can be realised by using TCP sockets or WebSockets.

OpenOffice⁵ or Apache OpenOffice is a well-known open source office suite that was mainly developed for reading and manipulating the OpenDocument Format (ODF). OpenOffice is extensible via add-ins that can be developed using many programming languages such as Python, C++ and Java. A third-party developer can create a rich OpenOffice add-ins that manipulates a document's content, modifies the existing OpenOffice user interface and communicates with other third-party applications using various communication channels (e.g. TCP sockets and WebSockets).

³<http://www.adobe.com/devnet/acrobat/sdk/eula.html>

⁴<https://www.foxitsoftware.com/products/sdk/phantompdf-plugin>

⁵<https://www.openoffice.org>

Document Viewer	Extensible	Document Manipulation	User Interface Customisation	Communication	Possible Integration
Adobe Acrobat Reader	✓	✓	✓	✓	✓
Foxit Reader	✓	✓	✓	✓	✓
OpenOffice	✓	✓	✓	✓	✓
Opera	✓	✓	✓	✓	✓
Firefox	✓	✓	✓	✓	✓
Sumatra PDF	✗	✗	✗	✗	✗
iBook Author	✗	✗	✗	✗	✗

Table 5.1: Investigating the integration of seven third-party document viewers with our link service

Similar to Google Chrome, most web browsers such as Opera⁶ and Firefox are extensible via add-ins. The Opera web browser can easily be extended via add-ins⁷. In general, the structure and development of its add-ins is similar to Google Chrome add-ins. Using a JavaScript background page, an Opera add-in can communicate with our link service via a WebSocket communication channel. Furthermore, the developer can manipulate HTML documents and modify the existing Opera user interface menus (e.g. context menu).

By using the Firefox add-ons SDK⁸, a third-party developer can develop rich add-ons for Firefox. A Firefox add-on is created based on various web technologies such as JavaScript, HTML5 and CSS3. Firefox add-ons can manipulate HTML documents or modify and enrich the existing Firefox user interface. Using HTML5 WebSockets, an add-on should be able to communicate with our link service.

Most existing EPUB readers such as Sumatra PDF⁹, a free PDF and EPUB reader, are not extensible via plug-ins or add-ins. As discussed in Section 2.5, IBA is based on the EPUB standard but offers more interactive features. iBook Author which is the reader for IBA documents offers a nice feature for extensibility. It allows end users as well as developers to build rich and interactive HTML widgets that can be embedded within IBA documents. For example, a user can create a widget to render a sequence of images and interact with them using a swipe gesture. Nevertheless, iBook Author does not allow the development of plug-ins or

⁶<http://www.opera.com>

⁷<https://dev.opera.com/extensions>

⁸<https://developer.mozilla.org/en-US/Add-ons/SDK>

⁹<http://www.sumatrapdfreader.org/free-pdf-reader.html>

add-ins to manipulate documents or communicate with third-party applications.

5.2.2.2 Document Formats

Table 5.2 summarises the results of our investigation regarding four different document formats. In the table, we use the ✓ symbol to illustrate that a feature is supported or that the document format can be integrated in our link browser. The (✓) symbol means that there is only limited support for a given feature or that the document format can be integrated in our link browser after some enhancements to its programming library. As shown, there are no Java-based libraries available for the IBA document format. Therefore, it is only possible to integrate IBA in our link browser after the development of a Java library for the IBA document format. In contrast to IBA, there exist many Java-based libraries for the OOXML document format. Apache POI¹⁰ and doxc4j¹¹ are the most well-known open source Java libraries for creating and manipulating OOXML documents. Even though both libraries do not support user interface customisation, they can be extended to support this feature. Therefore, we are convinced that these libraries can be used to develop a rich visual plug-in for OOXML for our link browser.

Document Format	Java Library	Document Manipulation	User Interface Customisation	Possible Integration
OOXML	Apache POI	(✓)	(✓)	(✓)
	doxc4j	(✓)	(✓)	(✓)
ODF	Apache ODF	x	x	x
	jOpenDocument	x	x	x
	Odf4j	x	x	x
EPUB	Epublib	x	x	(✓)
IBA	—	—	—	x

Table 5.2: Investigating the integration of four different document formats with our link browser

The Apache ODF Toolkit¹² is a well-known and documented Java library providing an easy-to-use API for reading and creating ODF documents. For many reasons we believe that this library is not suitable for developing a visual plug-in for the ODF document format for our

¹⁰<https://poi.apache.org>

¹¹<http://www.docx4java.org/trac/docx4j>

¹²<https://incubator.apache.org/odfdom/index.html>

link browser. The library does not provide an API to create annotations (i.e. selectors) within ODF documents. Furthermore, it does not provide handles to create GUI actions. Similar to the Apache ODF Toolkit, the jOpenDocument¹³ Java library for ODF enables only the reading and creation of ODF documents. Nevertheless, it is not possible to create selectors inside ODF documents or to add new GUI actions. Odf4j¹⁴ is another library that is written in the Java programming language and provides functionality for reading and manipulating ODF documents. Unfortunately, Odf4j does not support the customisation of the user interface or the creation of selectors within documents.

Epublib¹⁵ is a well-known open source Java library for reading and creating EPUB documents. The current version of the library does not support the manipulation of EPUB documents (i.e. adding selectors) or adding custom GUI actions. Nevertheless, we believe that it can be extended to support the missing features and that it can potentially be used to create a visual plug-in for the EPUB document format.

5.2.3 Discussion

The presented evaluation shows that besides our supported document formats and document viewers, other existing document formats and document viewers can be integrated with our link service. Unfortunately, changes are required for most existing open source Java libraries in order to develop visual plug-ins for their corresponding document formats. Moreover, recent document formats such as the proprietary IBA document format owned by Apple Inc.¹⁶ cannot be integrated with our link service. In contrast to the open source Java libraries that require enhancements, most existing third-party document viewers can seamlessly be integrated with our link service. Unfortunately, a few recent document readers for recent document formats (e.g. EPUB and IBA) cannot be integrated with our link service. The reader for IBA is a proprietary document format reader while EPUB readers are not yet powerful enough to offer extensibility features via SDKs.

In this evaluation we did not discuss the effort and time required for integrating a given document format or a third-party document viewer.

¹³<http://www.jopendocument.org>

¹⁴<https://wiki.openoffice.org/wiki/Odf4j>

¹⁵<https://github.com/psiegman/epublib>

¹⁶<http://www.apple.com>

The time required to integrate a given document format or a third-party document viewer with our link service depends on the used library or SDK. A visual plug-in for a document format might be developed in a week while another visual plug-in for another document format might take up to a month if changes are required to the used library. The same holds for developing add-ins. For example, we were able to realise the JavaScript-based add-in for Microsoft Word within a week, whereas it took us three weeks to realise the C# add-in for Microsoft Word.

5.3 End-User Evaluation

5.3.1 Goal

The main goal of our third evaluation was to evaluate the usability of the link service in terms of ease of use, satisfaction and the quality of the interactions.

5.3.2 Methodology

We evaluated the usability of our link service by means of both qualitative and quantitative evaluations in order to get a better understanding of the end-users experience and to gain more feedback [28]. According to previous research [62, 91], using a mixed-method approach is more effective than using a single-method approach. The qualitative evaluation consists of semi-structured interviews with the participants of the evaluation. On the other hand, in the quantitative evaluation we have used the well-known Computer System Usability Questionnaire (CSUQ) [88] which measures the end-user satisfaction with the usability of computer systems. CSUQ is included in Appendix D and it contains 19 different questions relying on a 7 point Likert scale (1 = strongly disagree, 7 = strongly agree). The 19 questions evaluate four different usability aspects. Questions 1 to 8 evaluate the ease of use (SYSUSE). Questions 9 to 15 evaluate the information quality (INFOQUAL) such as error messages or the documentation on how to use the system. Questions 16 to 18 evaluate the interface quality (INTERQUAL). The last question (19) evaluates the overall satisfaction (OVERALL). We have chosen the CSUQ questionnaire in our usability evaluation due of its accepted reliability. An alpha coefficient exceeding 0.89 has been proven for all

the four different parts (i.e. SYSUSE, INFOQUAL, INTERQUAL and OVERALL).

5.3.3 Population

We were mainly interested in knowledge workers' (researchers) point of view and experience when using our link service. Fourteen researchers working either on their Master's or PhD theses have participated in our usability evaluation. The participants' age ranged from 23 to 37 years ($M=27$), including 7 males and 7 females to keep gender balance. We have intentionally selected all participants from non-Computer Science specialisations in order to avoid any biased results. All participants are studying at the Vrije Universiteit Brussel¹⁷ (VUB) and were recruited through the "VUB experiment participant pool"¹⁸ Facebook page which was mainly created for recruiting students to participate in research experiments and evaluations conducted at the VUB.

5.3.4 Setup

Before conducting our study, we have prepared the link service and supported only one document format (i.e. plain text) and one third-party document viewer (i.e. Microsoft Word). In the online repository, we have uploaded plug-ins for the PDF document format as well as plug-ins and an add-in for Google Chrome. Note that the Google Chrome add-in was not uploaded (published) to the Google Chrome Web Store¹⁹ since it requires special validation from Google. We have further prepared some documents (i.e. two 'plain text', two PDF and two Word documents) in order to be used by the participants for creating and editing hyperlinks. Each document contained at least two pages. Moreover, the set of available documents consisted of two online Wikipedia articles (HTML documents). We started our evaluation by briefly explaining the context of our research and the objectives of the study to the participants, including the concept of a hyperlink, hyperlink sources and targets and the integration of document formats and third-party document viewers with our link service. We further explained the different functionality supported by the link service. After the explanations, the participants

¹⁷<http://www.vub.ac.be/en>

¹⁸<https://www.facebook.com/groups/VUB.participant.pool>

¹⁹<https://chrome.google.com/webstore/category/extensions>

were asked to use the link service and to perform a number of tasks. Each participant had to create, navigate and delete a number of bi- and multidirectional hyperlinks between text and Word documents. Furthermore, they had to enable and disable the highlighting of document selectors. The participants also had to extend the link browser to support the PDF document format and to integrate Google Chrome. They then have been asked to create and navigate a number of bi- and multidirectional hyperlinks between all the supported document formats. The average time for completing their tasks was 15 minutes. After finishing their tasks, the participants had to answer some demographic questions and to fill in the CSUQ questionnaires. This was followed by a semi-structured interview focusing on their perception and subjective satisfaction.

5.3.5 Results

In general, we received promising feedback about the usability of our link service as illustrated in Figure 5.5. Table 5.3 summarises the overall sample CSUQ and shows that the overall user satisfaction (OVERALL) was assessed very positively with a high mean and a small deviation. Nine of the participants provided us some encouraging comments about our link service such as: *“It is very easy to use”*, *“This application will definitely help me in doing my literature review”* and *“I do not know how it [the link service user interface] could be simpler”*.

Despite the fact that most participants had no knowledge about the concept of bi- and multidirectional hyperlinks in advance, the collected data from both the questionnaire and the interviews confirms the ease of creating and navigating both types of hyperlinks. Nevertheless, six participants were confused about the concept of navigating to a hyperlink’s sources and targets. For them, if they want to navigate a hyperlink that exists in a document then all other documents participating in the hyperlink (regardless of being in the hyperlink’s sources or targets) are targets of that hyperlink. Out of these six participants, four participants suggested to remove the **navigate to Source** button from the supported actions in the link browser and from the different add-ins of third-party document viewers. According to their suggestions, all hyperlink sources and targets have then to be listed under **navigate to Target**. Out of the total population, four of the participants were not satisfied with the hyperlink overview menu illustrated earlier in Figure 4.3 when editing

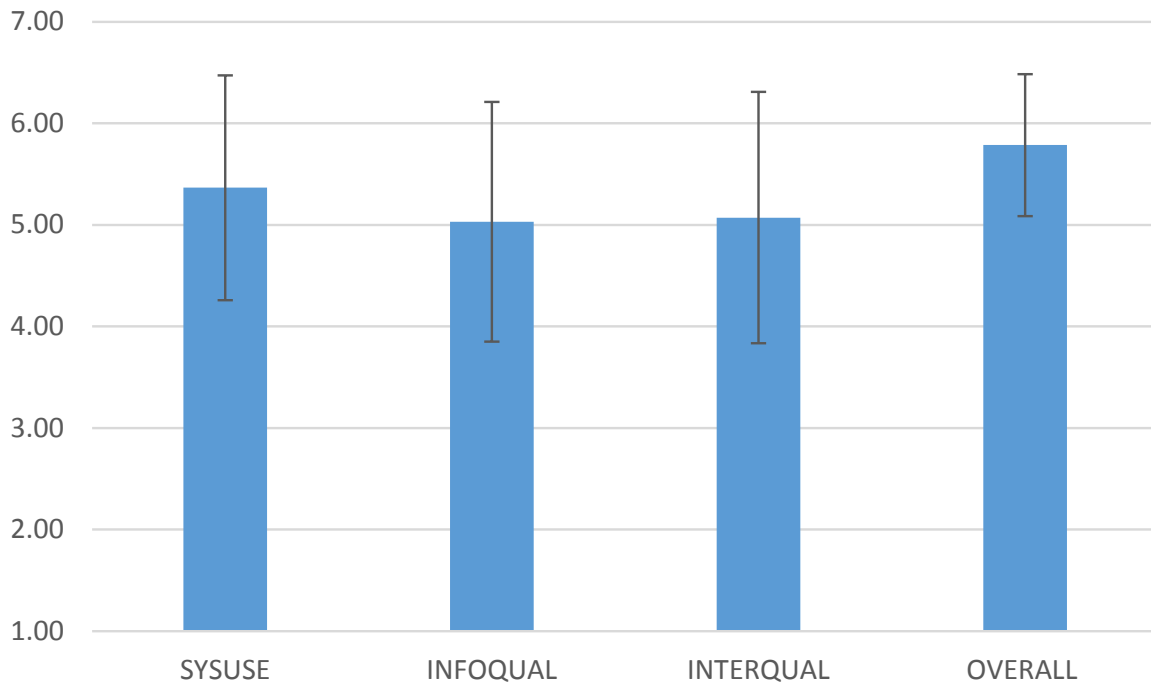


Figure 5.5: Results of the CSUQ questionnaire

their hyperlinks. All of these four participants mentioned that the used descriptions of the different selectors (e.g. XPointer-like expressions or start and end indices) are too technical. Instead of using an XPointer-like expressions for selectors within web pages they, for example, suggested to show (part of) the selected paragraph or statement.

One participant complained about installing Google Chrome add-in. According to this participant, they are not a Google Chrome user and therefore not aware that it can be extended via add-ins. Moreover, for this participant it was not easy to see the main menu in Google Chrome and to do the required steps for installing the add-in. The rest of the participants (thirteen) confirmed that it was very easy to extend the link service to support a new document format (PDF) and a new third-party document viewer (Google Chrome). *“It was only one single click to support the PDF document format; it is very easy to do it”*, one participant mentioned. All these thirteen participants emphasised the ease of extending Google Chrome with its add-in by following the provided guidelines illustrated in Figure 5.6. It is worth mentioning, that all of these thirteen participants had prior knowledge on how to extend Google Chrome with new add-ins from its online repository.

Subscale	Statistical Indices		
	Mean	Median	Std deviation
SYSUSE	5.37	5.5	1.11
INFOQUAL	5.03	5	1.18
INTERQUAL	5.07	5	1.24
OVERALL	5.79	6	0.70

Table 5.3: Summary of overall sample CSUQ

Out of the fourteen participants, ten participants have confirmed that the link service functionality is very intuitive and that they could become experts in using it after the creation of the first hyperlinks. All the participants indicated that the interactions provided by the link service are simple, easy to use and effectively helped them to perform all the required tasks. Moreover, most of the participants (twelve participants) have confirmed that the naming of the interactions were consistent across the different user interfaces (i.e. link browser, Google Chrome add-in and Microsoft Word add-in). One thing to note is that we noticed that some users (four participants) were confused while using the context menu of Microsoft Word to search for our add-in's command for creating a new selector in Word documents. By default, the Microsoft Word context menu has a command named **Hyperlink** which enables user to establish hyperlinks to external entire third-party documents. Some of the users initially thought that the **Hyperlink** command is the one that they should use to create the selector. After using the **Hyperlink** command, they noticed that the link service did not react by adding the intended selector to the hyperlink sources or targets and therefore they had to try again to create the selector by searching for the right command (i.e. **Add Selector** command).

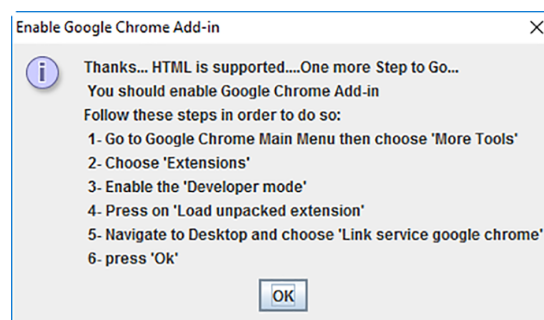


Figure 5.6: Guidelines for extending Google Chrome with its add-in

Out of the fourteen participants, ten participants have confirmed that the link service functionality is very intuitive and that they could become experts in using it after the creation of the first hyperlinks. All the participants indicated that the interactions provided by the link service are simple, easy to use and effectively helped them to perform all the required tasks. Moreover, most of the participants (twelve participants) have confirmed that the naming of the interactions were consistent across the different user interfaces (i.e. link browser, Google Chrome add-in and Microsoft Word add-in). One thing to note is that we noticed that some users (four participants) were confused while using the context menu of Microsoft Word to search for our add-in's command for creating a new selector in Word documents. By default, the Microsoft Word context menu has a command named **Hyperlink** which enables user to establish hyperlinks to external entire third-party documents. Some of the users initially thought that the **Hyperlink** command is the one that they should use to create the selector. After using the **Hyperlink** command, they noticed that the link service did not react by adding the intended selector to the hyperlink sources or targets and therefore they had to try again to create the selector by searching for the right command (i.e. **Add Selector** command).

Most of the participants (eleven participants) have confirmed that the information provided by the system (e.g. explanations for buttons for the mouseover events or the detailed examples provided in the **Help** menu) was clear and helpful. Some of the participants (three) who prefer explanations via videos, suggested to provide a two-to-three minutes demo explaining the different link service features. Two of the four participants who were confused about the command for creating selectors in Microsoft Word suggested to add some extra information in the Microsoft Word add-in guiding users when creating their first selectors in Word documents.

5.3.6 Discussion

The presented user study revealed that end users were satisfied with our link service. However, we must take into account their valuable remarks regarding the technical naming used in our link service such as hyperlink sources and XPointer expressions. In a future revision of the link service, we will consider listing the hyperlink sources under the hyperlink targets

as suggested by some of the participants. Moreover, we plan to create a demo in order to help beginners in using our link service.

We preferred to do this user study in order to get some general feedback helping us in enhancing the usability of our link service. In the future we might conduct additional studies to evaluate other aspects of the link service such as the usefulness of our link service as well as the efficiency of the hyperlink management using our link service.

5.4 Summary

In this chapter we have evaluated the extensibility of our link service in two different technical evaluations. In the first evaluation, we have integrated a number of document formats with the link browser via the corresponding data and visual plug-ins. Furthermore, we have integrated the general image multimedia content type in the link browser through the necessary data and visual plug-ins. We have further integrated three third-party document viewers in our link service. In a second evaluation, we have further validated the extensibility of our link service by investigating whether a number of existing document formats and third-party document viewers could be integrated with our link service. The evaluation revealed that our link service is flexible and extensible to integrate various existing document formats and third-party document viewers.

We have then evaluated the usability of our link service in an end-user study. By using quantitative and qualitative methods, we have collected data from fourteen knowledge workers who had the chance to use our link service. The collected data revealed that participants were satisfied with the usability of our link service user interface. We have further received a number of constructive remarks from the participants which might help us to further enhance the usability of the presented link service.

6

Use Case

In this chapter we further illustrate the usefulness of the hyperlinks created between different digital documents using our link service. We present a document discovery and retrieval framework that exploits the implicit relationships between documents based on the similarity of their content and metadata in order to improve the retrieval process. An innovative aspect of our framework is the combination of implicit and explicit hyperlinks between documents in the retrieval as well as in the visualisation process. As discussed, the visualisation of implicit and explicit hyperlinks in the document discovery tasks might further encourage users to create explicit hyperlinks. Please note that the purpose for presenting this case study is to present a potential application that benefits from our link service's hyperlinks. Furthermore, our case study discusses end-user feedback regarding the benefit of using our link service's hyperlinks in the document discovery tasks. Note that we were neither interested in developing new clustering algorithms for exploiting the implicit relationships between documents nor discussing new visualisation techniques for document retrieval tasks.

We start by providing an overview of different existing document discovery and retrieval systems. We then present our framework and briefly discuss some of its features, architecture and implementation. We further elaborate on the use of implicit and explicit hyperlinks in document

discovery tasks. Finally, we elaborate on some end users feedback that we got for the presented framework.

6.1 Document Retrieval Systems

As mentioned in Section 2.2, Bush proposed his visionary system in order to overcome the shortcomings of hierarchical classification structures. Bush's trails (associative hyperlinks) mimic the working of the human mind. Nowadays, web hyperlinks which were mainly inspired by Bush's trails are instrumental for document discovery and retrieval on the Web. In fact, Google's original document search was mainly based on the PageRank algorithm [87] which ranks documents based on incoming and outgoing hyperlinks. Unfortunately, the most common tool that we use for managing our documents is still the traditional desktop file explorer. Despite the underlying architectural differences between operating systems, the traversal of hierarchically nested folders is the standard way of locating documents. While this method is easy to follow, there have been many critical comments about the usability of hierarchical file structures [61, 54, 14], including the intensive cognitive load for basic tasks such as the classification and retrieval of documents [18]. A study by Golemati et al. [61] has demonstrated the profound limitation of hierarchical folders in desktop environments where users could not recall the file path of their documents in 17% of the retrieval tasks.

Many approaches for novel user interfaces have been proposed to overcome some limitations of the traditional desktop file browser. One approach is the use of 3D visualisations instead of the original 2D visualisations. Cone Trees [117] and Data Mountain [116] are good examples of visualisations that use 3D techniques. The focus+context technique for visualising large hierarchical structures is another approach for enhancing desktop document retrieval. The focus+context technique [16] provides two kinds of document views; a focus view and a context view. The former offers a visualisation of documents with a high level of details, while the latter provides a view of document's structure at an abstract level. Flip Zooming [16] and the hyperbolic file browser [85] are based on the focus+context technique. Another known approach for enhancing desktop document retrieval is the space-filling technique which is mainly developed to make use of the entire available screen space. Good

examples of this approach are Tree-map [80] and Sunburst [130] which visualise information hierarchies by using nested 2D rectangles.

Besides approaches for novel user interfaces, there are a number of studies and systems working towards exploiting document metadata for enhancing the document retrieval. Placeless Documents [48] addressed the fact that documents usually refer to several topics and should not belong to only one place. DeFiBro [102] is another recent system that exploits document metadata for document clustering and delivers an alternative visualisation.

An interesting approach for enhancing the document discovery and retrieval is the use of the Memex vision. SEMEX [27] is a well-known PIM system supporting access to different types of information objects (e.g. documents) based on a repository of objects and their associations. SEMEX applies a reconciliation algorithm [48] that automatically associates objects based on their metadata. Nevertheless, users are not given the possibility to add any association between information objects. Moreover, SEMEX simply displays the information objects in a nested list and users have to go through long lists when navigating information. iMapping [74] is another recent system that makes use of an ontology which supports associative hyperlinks between data objects. iMapping offers many interesting features such as the associations between different data objects and a zoomable user interface. Nevertheless, it is not evident how different documents as well as their metadata and content can be supported in the system.

6.2 Motivation

Existing approaches for document discovery and retrieval show a number of shortcomings. Systems with novel user interfaces neglect the content of a document and are limited to a predefined visualisation. Furthermore, some of them stick to the traditional hierarchical tree representation (e.g. Flip Zooming). 3D visualisations are less efficient than 2D interfaces [40] and document retrieval in a 3D visualisation requires more time than in a 2D visualisation [39]. There is no doubt that systems making use of document metadata go beyond the systems with novel user interfaces. Nevertheless, due to multiple reasons, existing document metadata is not always sufficient for document discovery and retrieval. First, system-created metadata contains basic information (e.g. time of

creation or media type) that does not add any semantics to different documents. Second, it is a big burden for users to manually add metadata to their documents. Last but not least, sometimes users use document names that reflect the context of the document and barely add semantics about a document itself. Systems trying to apply the Memex vision have many limitations. First of all, some of them rely on document metadata (e.g. SEMEX) to automatically create the associations between information objects. Second, they are not flexible enough to allow users to create their associations between documents. Last but not least, they are limited to a predefined visualisation (e.g. list).

We believe that an enhanced solution for document discovery and retrieval should not neglect the fact stated by Bush that documents do not exist in isolation but that they are related to each other [26]. The relations might be explicitly established (e.g. using our link service) or be defined implicitly based on the similarity of content or some metadata. As discussed, there have been little to no frameworks exploiting explicit hyperlinks to enhance document discovery and retrieval tasks. We believe that our link service's hyperlinks would be beneficial and valuable in enhancing the retrieval and discovery of digital documents. The link service's bi- and multidirectional hyperlinks would enable the easy retrieval of all linked documents regardless of whether they have been defined as a source or target of a hyperlink.

Furthermore, while some previous research has exploited implicit hyperlinks between documents using document metadata in order to enhance the document discovery and retrieval, to the best of our knowledge there are no desktop document discovery and retrieval solutions that exploit implicit hyperlinks in a document's content by, for example, using clustering algorithms.

6.3 Enhanced Document Retrieval and Discovery

Our proposed framework for document retrieval and discovery is unique in the sense that it overcomes a number of limitations of current document retrieval systems. An overview of the most important features of our framework is provided in Table 6.1. Next to each feature, we mention the document and/or framework components that have been used to realise the feature. The strength of our framework is that it exploits

the combination of implicit and explicit hyperlinks between documents. In contrast to previous research that only used document metadata to discover the implicit links between documents [102], we are using both a document's content and metadata in order to discover implicit links based on a clustering algorithm. The content of a document should tell us everything about a document and how it can be distinguished or related to other documents. It is worth mentioning that we also exploited the a document's content and metadata to support important features such as synonym-based search and word stems. Moreover, we make use of explicit document hyperlinks that have been created via our cross-document link service. Further, our framework offers an extensible architecture that not only supports multiple visualisations, but also enables the integration of arbitrary document formats at a later stage.

Search Feature	Document or Framework Component
Exact keyword matching	Index of metadata (title and filename) and content
Synonym-based search	Index of metadata (title and filename), content and WordNet lexical database
Multiple search criteria	Advanced search engine
Implicit linking via clustering	Clustering based on document content and metadata
Suggesting relevant documents	Clustering engine and explicit link engine
Multiple visualisations of a search result	Extensible visualisation engine

Table 6.1: Important features offered by the framework

6.3.1 Use of a Document's Content and Metadata

In our framework for enhanced document retrieval, a document's content and metadata serve three main purposes. First, they are used to build an inverted index to support full-text search. The inverted index is normally stored as a hashmap which enables a fast lookup for a word, a document or a document page. A second purpose for using the content and metadata is to discover the implicit links between different documents. In order to do so, we use a clustering mechanism to group similar documents into the same category (cluster). Figure 6.1 illustrates how our approach can visualise documents in distinct clusters with different colours using a schemaball visualisation. The effectiveness of document

clustering reduces time and effort in search tasks as has been proven in previous research [118]. Moreover, the clustering of documents based on their similarity is appealing since it minimises the scale of search results by organising documents into different categories rather than having the user to scroll through a long list of documents. Thereby, users can have a broad overview of the search results, which is beneficial in terms of reducing search time when retrieving and finding documents. Clustering helps the user in typical scenarios, for example when they encounter documents with the same name, since they can easily distinguish them by selecting the category. We use the k-means algorithm [89] and the number of clusters is determined by using the rule of the thumb.

The third purpose for using the content and the metadata is to support search based on synonym matches as well as keyword stems. By doing so, not only documents that contain the search keyword or its stems can be returned but also documents that contain keywords with the same meaning as the original search keyword. For example, if the user searches for documents with the keyword “computer”, documents containing the keyword “calculator” will be returned in addition to documents containing any form of the keyword “computer” (e.g. computation or computing). We make use of the well-known WordNet lexical database [101] from which we query a list of synonyms for every search keyword and then execute the search with the generated list of keywords.

6.3.2 The Use of Explicit Hyperlinks

As described earlier, apart from a document’s content and metadata, our framework makes use of the explicit hyperlinks created by users via our link service. These explicit hyperlinks are an excellent resource for enhancing the document retrieval and discovery. The existence of an explicit hyperlink between two documents A and B literally means that to some extent these two documents are related to each other. Therefore, if document A belongs to the result of a given search query, it is likely that document B may also be of interest to the user. As illustrated in Figure 6.2, we use this assumption to enhance our search results by adding document B to the preliminary search result if it is not already forming part of it.

In our framework, an *explicit link engine* has been implemented in order to query the explicit hyperlink metadata from our cross-document link service. This has been done via an RSL API offered by our cross-

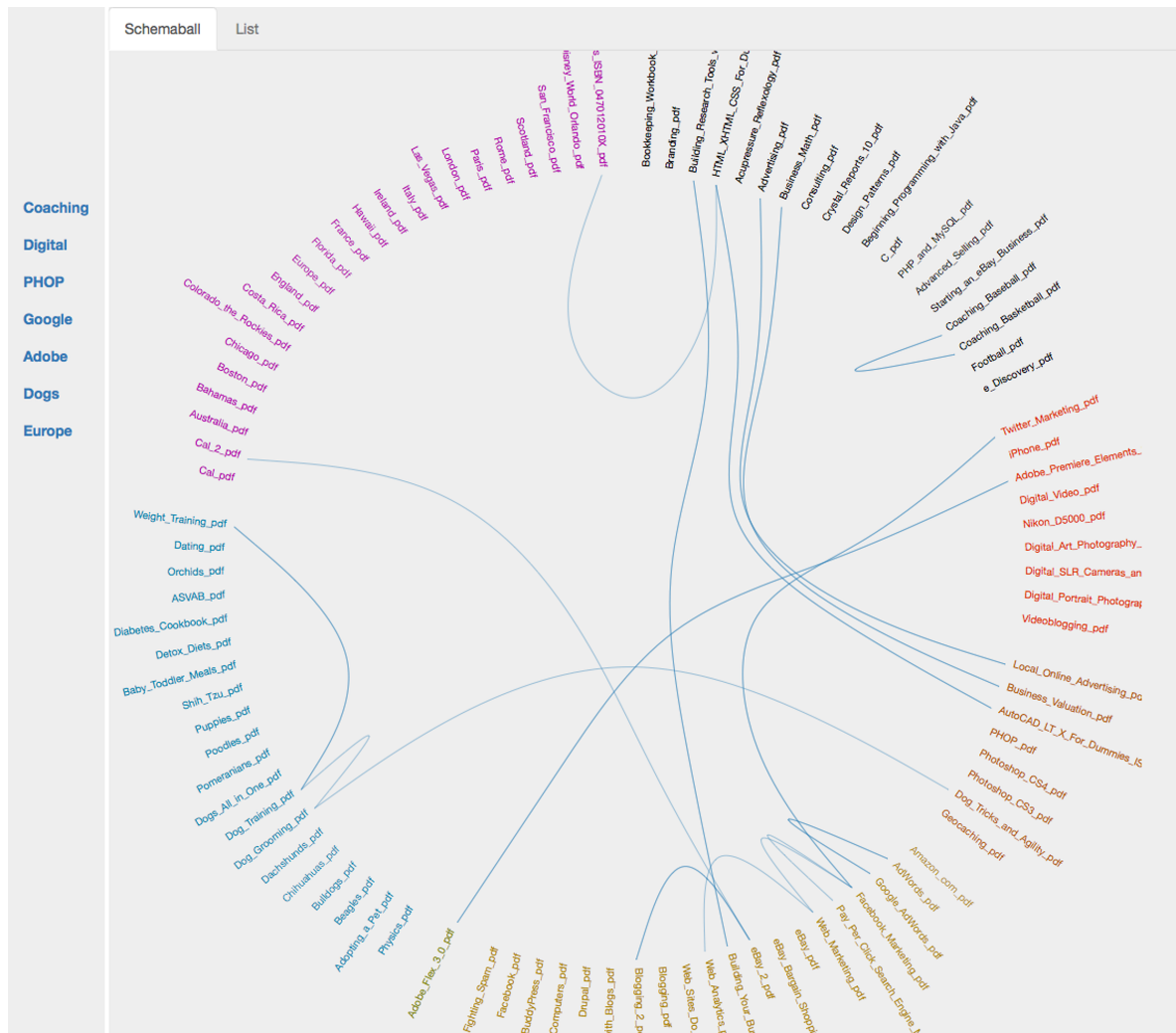


Figure 6.1: Schemaball visualisation of a search result, based on [136]

document link service. Listing 6.1 illustrates a number of methods of the RSL API that can be used to retrieve resources (documents) and linked documents. The explicit link exploiter enquires explicit hyperlinks for every document contained within the preliminary search result. The explicit link exploiter then augments the search results with retrieved documents that have explicit hyperlinks to any document in the preliminary search result.

```

1 public Entity getEntity(String name);
2 public Resource getResource(String name);
3 public Resource getResourceByURI(String uri);
4 public HashSet<Resource> getLinkedDocuments(Resource res);
5 public HashSet<Resource> getLinkedDocuments(String uri);

```

Listing 6.1: A number of methods offered by the RSL API

Document A and B which have an explicit hyperlink between them do not necessarily have to belong to the same cluster. According to the cluster hypothesis [137], documents that are similar to each other tend to belong to the same cluster. In unsupervised clustering algorithms (e.g. k-means algorithm), documents are grouped in clusters based on the document vector similarity (i.e. textual values as well as metadata). It is possible that the explicit hyperlink between document A and B is established in a specific context but they are content-wise not similar to each other. The clustering of documents in our framework is therefore done after augmenting the search result with documents having explicit hyperlinks with any document in the preliminary search result.

When visualising the search results, we take into account the existing explicit hyperlinks between the documents. In the schemaball visualisation shown in Figure 6.1, the explicit hyperlinks between documents are visualised as connected curves. In an ordinary list visualisation (see Figure 6.4), a document's explicit hyperlinks are shown next to **Have links to** at the end of each search entry.

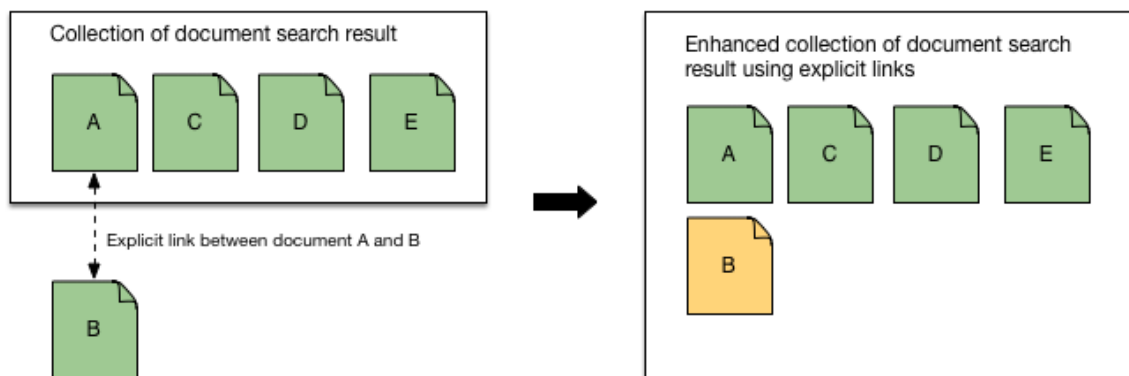


Figure 6.2: Document B which has an explicit hyperlink to document A is added to the search result if it is not already forming part of it, based on [136]

6.3.3 Support for Multiple Visualisations

Our framework is extensible to support different visualisations. Every visualisation has its strengths and weaknesses and we did not want to restrict our framework to a single visualisation. Normally each visualisation provides a different perspective of the documents, and hence, users

can switch between various visualisations according to their needs. Currently, we have implemented two different visualisations; a schemaball visualisation and an ordinary list visualisation. In the schemaball visualisation, we have the chance to minimise the necessary space as well as to demonstrate the explicit hyperlinks and document clusters on a single plane as shown in Figure 6.1. Users can view some information about the document including a text snippet that contains the matching results and some metadata by hovering over the document name. Moreover, they can filter documents by clicking on one of the cluster names shown in the left panel. Figure 6.3 illustrates the filtering of documents using the “digital” cluster name.

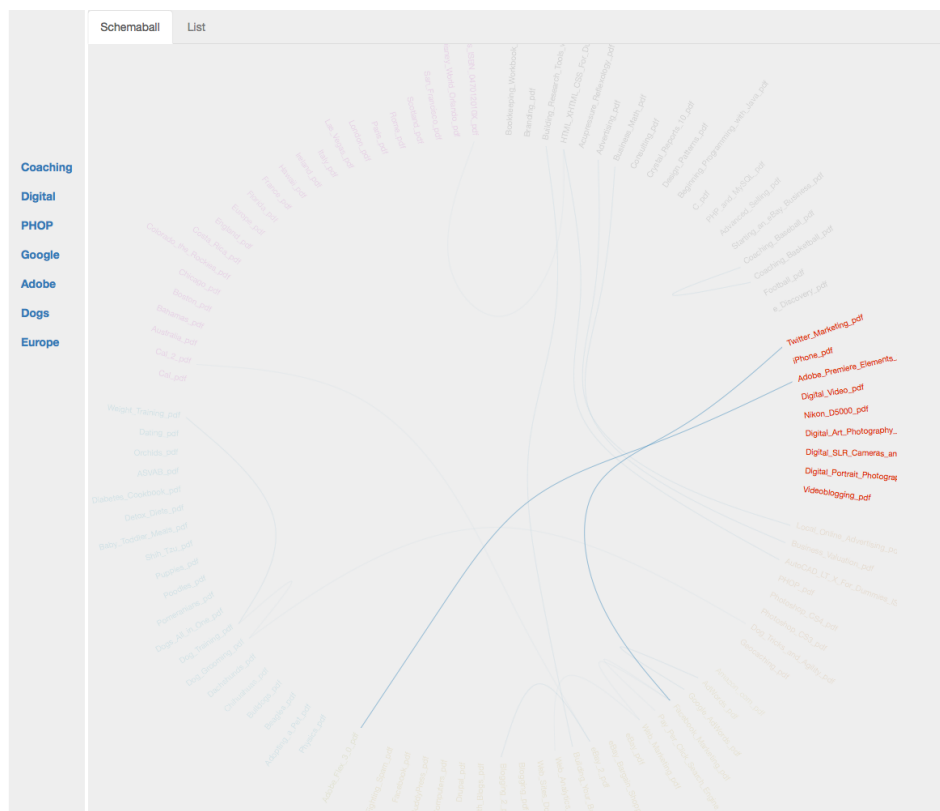


Figure 6.3: Search result filtered by cluster name, based on [136]

In our second visualisation, we can show richer textual information than in the schemaball, including the file name, snippets of the document and the target document of an existing explicit hyperlink. Moreover, users do not need any extra interaction (e.g. hovering) to see such details. Figure 6.4 shows a search result using the ordinary list visualisation. Please note that, in both visualisations we visualise the cluster name in the left panel as shown in Figure 6.1 and Figure 6.4. When a user clicks

on a cluster name, only the documents in the corresponding cluster will either be highlighted in the schemaball visualisation or listed in the list visualisation. It is worth mentioning that we automatically name the clusters by using the name of the document that is closest to the centroid.

Alternatively, we can use the most important keyword in the cluster to define the name of the cluster

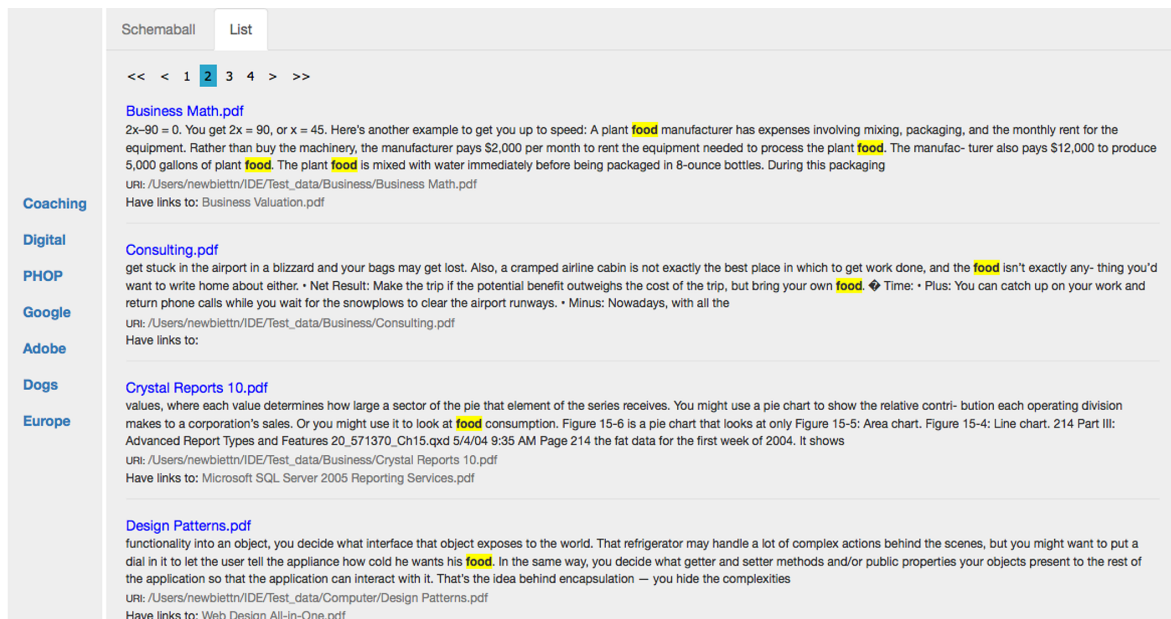


Figure 6.4: Ordinary list visualisation of a search result, based on [136]

6.3.4 System Architecture

The general architecture of our document retrieval framework is illustrated in Figure 6.5. The architecture consists of three main components. The first component is responsible for handling the search queries entered by the user. A second component is responsible for the document clustering. The third component collects the documents returned by a search query and publishes the results in a neutral data representation which can then be further processed by any visualisation engine.

In order to illustrate the communication between the different components, we present a scenario of a user who would like to search for a document. Note that we refer to the numbers depicted in Figure 6.5. The user can enter a search query using the provided search interface (1). The search interface which is shown in Figure 6.6 enables the user to query

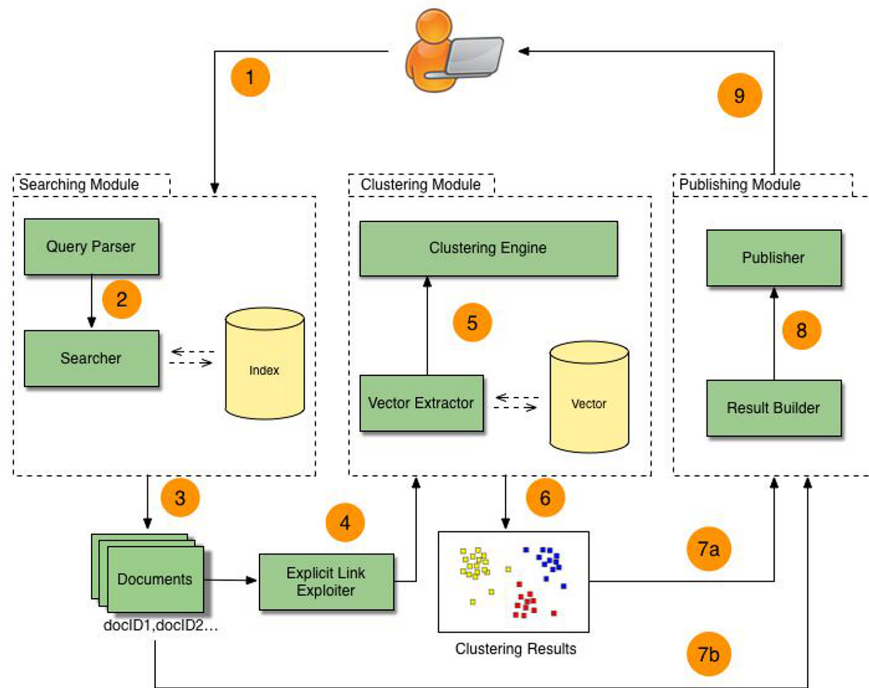


Figure 6.5: General architecture of the document retrieval framework, based on [136]

documents based on multiple search criteria. A user can search with keyword synonyms or any form of a keyword contained in a document’s metadata or content. Furthermore, the user can retrieve documents via the name of the author or the date of creation. In contrast to existing systems (e.g. Mac Finder) that enforce a logical AND operator between the different search criteria, we give the user the freedom to use logical AND as well as OR operators between different search criteria. For example, a user can search for a document which is authored by “Ted Nelson” OR contains any form of the word “compute” in its filename. Furthermore, the user can search by using the wildcard operator. The keyword “comput*” can for example be used to search for documents that contain “computer” or “computing” in the filename.

The search query is forwarded to the searching module in order to be parsed (2). If the user opts for a search using synonyms, the search module will communicate with the WordNet database in order to retrieve all the synonyms of the original keywords. Both the keywords entered by the user and the retrieved synonyms are combined to search over all documents in the file system. The index of the document’s metadata and

The screenshot shows a web interface titled "The Novel Search". At the top right, there is a yellow button labeled "To add search rules". Below the title, there is a toggle switch labeled "Switch between AND/OR search" with "AND" selected. The main area contains five search rules, each with a "Delete" button (a red square with a white 'X') on the right:

- Rule 1: Content contains keyword1
- Rule 2: Content contains keyword2
- Rule 3: Date created after keyword3
- Rule 4: Author contains keyword4
- Rule 5: File name contains keyword5

At the bottom left is a blue "Search" button. At the bottom right, there is a yellow button labeled "To delete the search rule".

Figure 6.6: Enhanced search form, based on [136]

content only contains word stems. Therefore, we take the stems of the search keywords in order to be able to search over the index. It is worth mentioning that searching with author name or creation date stems does not make sense and the stemming algorithm is not used in this case. The search module searches over the index and a list of matching documents is returned based on the user query (3). The search result is then passed to the explicit hyperlink exploiter engine. As described earlier, the explicit hyperlink engine retrieves other documents that have any relationships with documents contained in the search result (4) by querying the explicit link metadata via the API offered by our cross-document link service. The search result is augmented with any additional retrieved documents and forwarded to the clustering module. The vector extractor component of the clustering module returns matching vectors based on the list from the vector repository (5). The matching vectors are read and text mining is performed in order to cluster the documents (6). When the module finishes its task, it returns the final clustering results. Both the search result (7b) as well as the clustering results (7a) are then passed to the publishing module where the search result is finally formatted in JSON and ready to be processed for further visualisations.

6.3.5 Implementation

The browser (user interface) of our document retrieval framework has been realised based on the latest web technologies in order to provide a platform-independent solution. The powerful D3.js¹ visualisation library has been used to visualise the document search results. An advantage of

¹<http://d3js.org>

the D3.js library is that it can run on most web browsers without any configurations.

The core logic of our framework is mainly based on the cross-platform Java programming language. We have used a number of open source libraries such as Apache Tika², Apache Lucene³ and Apache Mahout⁴. The Apache Tika library is the de facto open source document parsing library that supports text extraction of a large number of document formats. The indexing of documents as well as the support of search queries over the index have been mainly realised via Apache Lucene. Last but not least, the Apache Mahout library has been used for the data mining and clustering of documents.

A last thing we have to emphasise is the publishing module. In order to support as many visualisations as possible, the publishing module provides a RESTful API which produces JSON data that can be further processed by different visualisation engines in order to properly present the search results.

6.4 User Evaluation

The presented framework has been evaluated in an end-user study. The main goal of the evaluation was to evaluate the usability of the application in terms of ease of use, satisfaction and usefulness. We were mainly interested in getting feedback about the use of implicit and explicit hyperlinks as well as the multiple visualisations. In the evaluation, we were not planning to make any comparison with existing document retrieval systems (e.g. Windows search system or Mac Finder) in terms of time performance. However, we have intentionally chosen participants with different operating systems preferences (5 Mac users and 5 Windows users). This was done to avoid biased results, especially if users are satisfied with the document retrieval mechanism offered by their operating system. We were mainly interested in knowledge workers' point of view who are working with documents as part of their daily tasks. Hence, all of our 10 participants (5 female) were researchers working either on their Master's or PhD theses. Since the main goal of our evaluation was usability testing, it was not necessary to use the personal document

²<https://tika.apache.org>

³<https://lucene.apache.org/core/>

⁴<http://mahout.apache.org>

collections of our participants. Instead, we prepared a large document collection beforehand. We decided to choose the ‘For Dummies’⁵ book series as test data. These books cover a wide range of topics from accounting, photography, computing to blogging. Moreover, the content of the book collection is not too abstract and users will have the freedom and flexibility to easily discover the collection. The collection of the books was categorised in multiple folders and sub-folders. Moreover, in order to minimise the required time for the evaluation, we have used the cross-document link service to establish various explicit hyperlinks between documents that are distributed in the folders of the collection. Last but not least, for our evaluation we turned the synonyms-based search feature off.

We started our evaluation by briefly explaining the objectives of the study to the participants, including the concept of explicit and implicit document linking. We further introduced the book collection to the user and gave them approximately ten minutes to explore the data. After the participants felt familiar with the dataset, we told them about the explicit hyperlinks that we had already created between the documents beforehand. We then introduced our document retrieval framework and asked them to perform different document retrieval tasks by using the framework. After finishing their tasks, the participants had to complete a questionnaire followed by a semi-structured interview. The questionnaire contained closed 5- point Likert scale questions to assess different usability criteria (e.g. ease of use) for the application in general, the implicit linking and the explicit hyperlinks.

In general, we received promising feedback about our document retrieval framework as illustrated by the results shown in Figure 6.7. We below elaborate on some the feedback we received about implicit and explicit hyperlinks but for detailed explanations about the results of the user evaluation please refer to [136]. Participants were delighted to see the relationships between documents in terms of the explicit and implicit hyperlinks. Most of them normally deal with many journal and conference papers that are distributed over various folders, making it difficult for them to organise and classify their documents. One user commented: *“It is really a great application because I can find the relationships between documents, especially when you have to do your thesis. You have read a lot of papers and you want to make an integrated literature review.*

⁵<http://www.dummies.com>

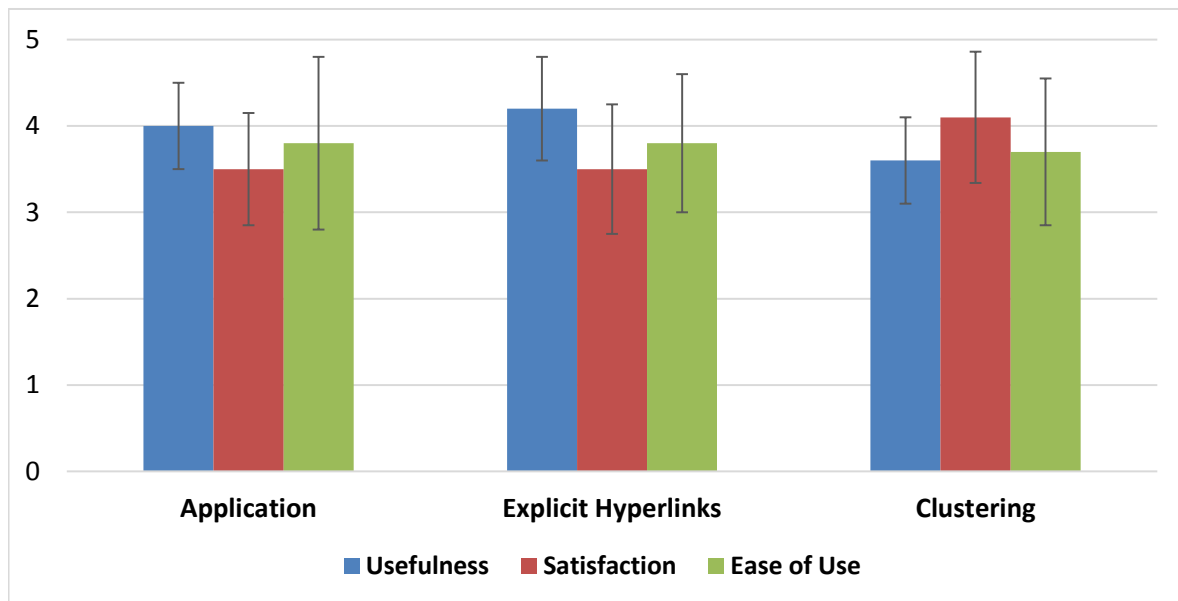


Figure 6.7: Some results of the user evaluation

Suggesting related documents as well as showing me what documents I linked with each other makes my life much easier. You do not have any application that does that ”. Some participants have emphasised the usefulness of suggesting relevant documents via the implicit and explicit hyperlinks. One participant said: *“I download a lot of papers and I do not memorise the titles. This feature [flexible search over content and metadata] combined with suggesting relevant documents would definitely help me to find my documents”.*

Despite the fact that all participants had no knowledge about the concept of explicit hyperlinks in advance, the collected data confirms the contribution of the explicit hyperlinks to document retrieval. Whereas the score of the usefulness (see Figure 6.7) indicates that explicit hyperlinks are very much beneficial to all participants, the score of the ease of use and satisfaction gives us a hint about the unfamiliarity with the concept. One user mentioned: *“It is very beneficial for doing literature review”.* Another participant explained: *“The link visualisation is very interesting. It shows me the connections between the documents that I would probably create over time. Normally, I had to write annotations to link documents or memorise the relations between documents by heart. Creating and visualising the links will help me to quickly remember the relation between documents and their content”.* Nevertheless, because we

had created the explicit hyperlinks by ourselves, one user expressed their confusion about the explicit hyperlinks and another participant commented: *“I do not know if these links are useful. I did not create them myself. Probably, I will be glad to see them when I create them”*. Another user wondered about the scalability of the explicit link visualisation. In fact, the scalability of the visualisation is an issue for any large data set, but for a future version of the visualisation we foresee some advanced settings where users can choose to visualise the explicit hyperlinks for a specific document, cluster or all the documents.

6.5 Summary

We have presented a framework for enhancing the retrieval and discovery of documents. The presented framework has showed the importance of the explicit hyperlinks created by our cross-document link service. Based on a clustering algorithm in combination with our link service, our document retrieval framework exploits implicit as well as explicit document relationships in order to improve the retrieval and discovery process. While we currently support two different search result visualisations, the framework might be extended with arbitrary third-party visualisations. An initial end-user evaluation of the presented framework has further revealed that users are keen on exploiting the implicit and explicit relationships between documents as part of the document retrieval process. The framework can further be extended to suggest new “explicit hyperlinks” between documents for the users based on the results of the document clustering (i.e. the implicit hyperlinks).

7

Conclusions and Future Work

In this final chapter we summarise and discuss the results of this dissertation. We further discuss our research contributions as well as some limitations of the presented work. We finally present some future work and new research directions. This chapter is structured as follows. The presented work is summarised in Section 7.1. In Section 7.2 we discuss our answers to our research questions, our contributions as well as the limitations of our work. We conclude the chapter by presenting potential future work in Section 7.3.

7.1 Summary

In this dissertation we have presented a user study that explores the user behaviour in associating information within and across digital as well as physical documents. We have further introduced a dynamically extensible cross-document link service which enables the linking across existing and emerging document formats as well as third-party document viewers.

7.1.1 User Study

The presented user study has a multi-case design that consists of an online survey combined with interviews with participants of the online survey. In the online survey, we have collected data from 238 participants and afterwards 12 participants out of the 238 participants have been interviewed. Our study revealed a number of interesting findings. First of all, it showed that most knowledge workers are associating information within and across digital and physical documents. Furthermore, it revealed a number of association mechanisms applied by participants to associate information within and across documents. These mechanisms include but are not limited to the use of physical folders, post-it notes, digital folders and line drawings. Moreover, we found some characteristics of the different types of associations created by users using the different association mechanisms.

Our study further showed that most users are not satisfied with their currently used mechanisms for associating information across documents. We have critically analysed and discussed the participants' criticism and complaints regarding their currently used linking mechanisms. Most of the participants who are engaged in information association activities have indicated the need for a cross-document linking solution to facilitate the creation and management of associations. Based on the participants' criticism and their suggestions, we formulated a number of design implications for an extensible information and cross-document linking solution.

7.1.2 A Dynamically Extensible Cross-Document Link Service

The presented link service meets the user needs and requirements for an efficient cross-document linking solution. It overcomes the limitations of existing linking solutions (e.g. support for only a predefined set of document formats). Our link service provides end users with a link browser enabling them to visualise documents and create as well as edit hyperlinks. It allows users to create advanced hyperlinks in the form of bi- and multidirectional hyperlinks between snippets of information in the supported document formats. Our link service takes into account that users use third-party document viewers for editing and reading different document formats. Therefore, it addresses the challenge of seamlessly integrating third-party document viewers. For any document format that

should be supported by the link service —either visualised in the link browser or externally in its third-party document viewer— a data plug-in extending the link service link model (RSL metamodel) has to be provided. The data plug-in for a document format must contain information on how to address its resources (documents) as well as selectors (anchors) attached to its documents. For every document format to be integrated and rendered in the link browser, a visual plug-in extending the link browser has to be provided. A visual plug-in for a given document format has to render its documents and visualise any selectors that have been defined. For each document format to be integrated via a third-party document viewer, a specific document format gateway as well as an add-in for the third-party document viewer have to be provided. A gateway for a specific document format extends the link service and is responsible for launching the corresponding third-party document viewer and handling messages with the third-party document viewer add-in. A third-party document viewer add-in should enable users to create and edit selectors within a document and communicate with the link service via its corresponding gateway about selectors to be selected, deleted or updated.

Most of the presented link service components are flexible and extensible to support the general goal of integrating existing as well as emerging document formats. For instance, the link service supports multiple communication channels (i.e. TCP sockets, WebSockets or a RESTful API) and is flexible to support other communication channels in order to support a wide range of third-party document viewers. The link service's dynamic extensibility further allows third-party developers and end users to support any document format and multimedia content type (e.g. images and videos) without the intervention of the link service provider.

The link service currently supports the linking across six different document formats including XML, plain text, PDF, HTML, Microsoft Word and Microsoft PowerPoint. The XML, PDF and plain text document formats have been integrated using visual plug-ins for the link browser while HTML, Microsoft Word and Microsoft PowerPoint have been integrated via Google Chrome, Microsoft Word and Microsoft PowerPoint third-party document viewers. Furthermore, the link service supports the linking to general multimedia types (i.e. images and YouTube videos). Advanced hyperlinks can be created between (parts of) documents and (parts of) multimedia types.

Our link service has been evaluated in three different evaluations. In a first evaluation, we assessed the extensibility of the link service by integrating different document formats, general multimedia types and third-party document viewers. In a second evaluation, we investigated whether a number of existing document formats can be integrated in the link browser by analysing their corresponding programming libraries. In the same evaluation, we analysed whether a number of existing third-party document viewers can be integrated with our link service by investigating their SDKs. In a last evaluation, the usability of the link service has been evaluated in an end-user study. In this user study, 14 participants had the chance to use the link service and perform a number of tasks. After performing their tasks, each participant has been asked to fill in a questionnaire which was followed by a semi-structured interview.

We have further illustrated the usefulness of the hyperlinks created by our link service by presenting a framework for enhanced desktop document retrieval and discovery which makes use of these hyperlinks. Besides our link service's hyperlinks, the presented framework applies a clustering algorithm in order to discover implicit relationships between desktop documents. The framework then exploits the link service's hyperlinks as well as the discovered implicit relationships to deliver different visualisations of document search queries.

7.2 Discussion

Throughout this dissertation we have presented answers to the research questions formulated in Section 1.4. The presented user study provides answers to the first research question (RQ1) and is the first user study that mainly investigates the user behaviour in associating information within and across digital and physical documents. The user study also provides some answers to RQ2.1 by outlining a number of design implications for a suitable information and cross-document linking solution. Our review and critical analysis of existing linking approaches provides the remaining answers to RQ2.1 by outlining six fundamental requirements for an ideal cross-document link service. The presented link service architecture, prototype and technical evaluations offer answers to research question RQ2.2. While developing our link service we have taken into account the answers for RQ2.1, the design implications for a suitable in-

formation and cross-document linking solution as well as the fundamental requirements for an ideal cross-document link service. The end-user usability evaluation of the presented link service goes beyond answering RQ2.2 by presenting the end user feedback regarding the usability of the presented link service.

Our user study has made a number of contributions and revealed a number of important findings such as the discovery of different association mechanisms that are used by users, their characteristics or the user need for an efficient linking tool. Due to the exploratory nature of our user study, we believe that its findings open new horizons for in-depth exploration of user behaviour in associating information within and across documents. Our user study has shown the need for efficient linking solutions in order to enable knowledge workers to associate information during their reading and writing activities. Unfortunately, most existing linking solutions as well as bibliography reference managers are not sufficient to help users in creating and managing their associations. The proposed design implications should help developers and researchers in realising efficient and usable cross-document linking solutions.

We have taken the initiative by presenting a first cross-document link service that takes into account end-user needs and requirements. The presented cross-document link service also addresses other important requirements in order to overcome the limitations of existing linking solutions (e.g. extensibility or support for third-party document viewers). The conducted end-user evaluation revealed that our approach is accepted by end users, whereas the two technical evaluations confirmed that our link service excels existing link services in terms of extensibility and its support for third-party document viewers. We think that one main reason for the success of our link service is the use of a powerful link metamodel (i.e. RSL) and the adoption of the general architecture for open cross-media annotation and link services originally proposed by Signer and Norrie [126, 125].

Our answers to the formulated research questions make a number of important contributions. Furthermore, as any research effort, the presented research also shows some shortcomings. In the following we elaborate on our contributions and highlight the limitations of the presented work.

7.2.1 Contributions

1. Our user study makes a number of important contributions:
 - (a) Our user study is the first study that is mainly investigating the user behaviour in associating information within and across physical and digital documents. The study has confirmed previous research findings [2, 109, 81] by showing that knowledge workers are associating information across documents while reading and writing. Our user study further showed that most users are either occasionally or frequently associating information across documents. The study also showed that knowledge workers are associating information within a single document more frequently than across different documents. Our study further revealed 12 different association mechanisms that are adopted by participants when associating information in digital as well as physical documents. The study also analysed how frequently these association mechanisms are used in the different information association scenarios (i.e. SP1, SP2, SD1, SD2, SD3 and SDP) described in Section 3.3. Furthermore, our study revealed some general characteristics of the different types of association mechanisms. For every association mechanism, we describe the directionality of the resulting association (e.g. uni- or bidirectional) as well the granularity of the associated information.
 - (b) Our user study revealed some general findings about the users' appreciation and criticism about their currently used association mechanisms. Furthermore, it has demonstrated that there is a need for an efficient and suitable information and cross-document linking solution in order to help users in creating and finding their associations.
 - (c) Based on the users' criticism about their used mechanisms for associating information, their suggestions for a better solution and our interpretations of the collected data, we have formulated a set of design implications for information and cross-document linking solutions.
2. We have critically analysed and compared existing linking solutions in Chapter 2. Based on this critical analysis and our own expert-

ise, we have outlined six fundamental requirements for an ideal cross-document link service. These requirements have further been disseminated in our publication entitled “A Dynamically Extensible Open Cross-Document Link Service” [135] which was presented at the WISE 2015 conference.

3. Our link service which has been described in two publications [134, 135] also makes a number of important contributions:
 - (a) We have proposed an architecture for a dynamically extensible link service. In contrast to most existing link services that have been built as monolithic components, our link service has proven its flexibility and extensibility in two different technical evaluations. As a result and in contrast to most existing link services, our link service can be extended by third-party developers and end users in order to support any existing as well as emerging document formats without the need for any intervention by the link service provider.
 - (b) By using a visual plug-in mechanism, our link browser can support the visualisation of any document format. Therefore in contrast to existing link services that support the linking across a predefined set of document formats, our link service is able to support the linking across existing as well as emerging document formats.
 - (c) Our link service addresses the challenge of seamlessly integrating existing third-party document viewers by providing flexible communication channels as well as a gateway component that mediates the communication between the link service and third-party document viewers. In contrast to existing link services, third-party document viewers do not need to be rewritten in order to benefit from our link service.
 - (d) We support advanced linking in the form of bi- and multidirectional hyperlinks across six different document formats. Three of the supported document formats are visualised within their third-party document viewers. We further illustrated the usefulness of the hyperlinks created between different documents by presenting the use case of a novel document discovery and retrieval framework that overcomes some of the limitations of existing document retrieval frameworks.

7.2.2 Limitations

Our sample of totally 238 researchers can be considered as a representative sample of the research community. However, our subsamples of 23 Master's students, 169 PhD students and 46 researchers holding an PhD degree are not large enough to be representatives of their corresponding populations. We believe that the Master's community (people who are doing some Master's studies) is much larger than the PhD community (people who are doing PhD). Therefore, we believe that if the number of Master's students would be a representative of their corresponding population, we could analyse and compare the information association behaviour between the different subsamples. Moreover, we think that the sample of female participants (82) is not representative for their corresponding population (female researchers) and therefore, we were not able not analyse and compare the information association behaviour between female and male researchers.

The manageability and maintainability of hyperlinks has always been an issue in any hypermedia system. This includes broken hyperlinks, the consistency of hyperlinks when the linked documents evolve or the management of hyperlink metadata in collaborative environments such as Google Docs. Since these issues were not part of the goal of our work, we have adopted a simple document archiving solution of linked documents which also helps to address the broken hyperlink problem in the case of missing documents. Another important issue that should always be taken into account, which also was not part of the goal of our work, is the scalability of any hypermedia solution. Using a document archiving solution might affect the scalability as well as the performance of our link service. In fact, the performance of our link service has been affected while using the archiving solution. Therefore, we believe that the scalability as well as the performance of our link service should be more investigated. It is worth mentioning that we have already used our link service for creating hyperlinks between more than sixty documents without the archiving solution and we did not notice any problem of its performance. Furthermore, during the end-user study we did not receive any negative comments about the performance of our link service.

The current version of the link service only works on a single computer and does not support the portability of hyperlinks. In order to illustrate what we mean with hyperlink portability, let us assume that a document A resides on a computer C1. A user has created a number of

hyperlinks in document A by using our link service that is installed on computer C1. Later, the user transferred their document A from computer C1 to another computer C2. Unfortunately, even if the user has our link service installed on his computer C2, document A will be transferred without its hyperlinks. There are different ways how this issue can be addressed. First of all, our link service can be used in a shared repository where hyperlinks are stored in a shared database and are visible to all users of the shared repository. However, as we discussed earlier in Section 4.1, using our link service in a shared repository might trigger many challenging issues such as privacy and the collaborative editing of hyperlinks that were out of the scope of this thesis. Another solution might be to track documents that have to be transferred to other computers. A user should then be given the flexibility to transfer a document along with its metadata (i.e. hyperlinks) as well as other documents sharing hyperlinks with the document to avoid the problem of broken hyperlinks. For example, when the user wants to transfer document A from computer C1 to computer C2, the tracking component of computer C1 will extract the metadata from the link service and give the user the possibility to transfer the document along with its metadata as well as all documents sharing hyperlinks with document A. The tracking component on the other computer (i.e. computer C2), should notify the link service that is installed on the same machine about the new document and its metadata.

We got encouraging feedback from most of the participants of the end-user usability evaluation. Nevertheless, we believe that this evaluation is not sufficient enough to evaluate the usefulness of our link service. In order to get more accurate and precise results, the link service would have to be evaluated in a long-term end-user evaluation where users use the link service in their daily reading and writing activities over a long time period. Users should keep daily diaries of their reading and writing activities as well as their interaction with our link service.

7.3 Future Work

The work presented in this dissertation offers multiple opportunities for future work. Some of these opportunities are related to the previously discussed limitations. Other opportunities are improvements to our link service or extensions to our work into other research domains. In the

following, we discuss some future work regarding the presented research.

We are planning to integrate more document formats in the link browser and to support additional third-party document viewers such as Adobe Acrobat Reader, Foxit Reader and Mozilla Firefox. Furthermore, we are planning to evaluate the ease of extending the link service in a user study with third-party developers. Moreover, we would like to investigate some mechanisms for enhancing the manageability and maintainability of the link service's hyperlinks.

We are further interested in conducting a long-term end-user study in order to evaluate the presented link service's efficiency, effectiveness and usefulness. We are planning to give a number of end users the opportunity to use the link service for a period of four months starting from February 2017. End users will be asked to keep diary notes on their interactions with the link service. A number of interviews would also be planned during and at the end of the study.

Besides the aforementioned opportunities, the presented dynamically extensible cross-document linking solution serves as a research platform for investigating document and link management as well as maintainability in so-called cross-media information spaces. Our solution might inspire other link service providers to reconsider the dynamic extensibility of their approaches. It further presents an ideal platform for investigating innovative forms of cross-document linking and transclusion using the bidirectional RSL hyperlinks.

Appendices

A

Survey on Cross-Document Associations

Survey on Cross-Document Associations

This survey takes place in the context of research on an extensible and scalable authoring approach for open cross-media information spaces. The survey investigates users behaviours in associating information across digital and physical documents. It should take about 10 to 15 minutes to answer the questions of this survey.

We would like to thank you for your participation!

Ahmed A. O. Tayeh, PhD Candidate
Web Information System Engineering (WISE) Lab
Computer Science Department
Vrije Universiteit Brussel

For additional information, please feel free to contact Ahmed A. O. Tayeh (Ahmed.Tayeh@vub.ac.be)

There are 34 questions in this survey

General Information

[]What is your age? *

Please choose **only one** of the following:

- ☐ Younger than 18 years
- ☐ 18-24 years old
- ☐ 25-34 years old
- ☐ 35-44 years old
- ☐ Older than 45 years

[]What is your gender? *

Please choose **only one** of the following:

- ☐ Female
- ☐ Male

[]What is your current country of residence? *

Please write your answer here:

[]What is the highest level of education that you have achieved? *

Please choose **only one** of the following:

- ☐ Secondary school
- ☐ High school or equivalent
- ☐ Bachelor's degree
- ☐ Master's degree
- ☐ Doctoral degree (PhD)
- ☐ Other

Associating Information in Physical Documents

The following questions investigate a user's behaviour in **associating information in physical documents** (e.g. printed books or papers). We differentiate between two scenarios; 1) associating information in a single physical document and 2) associating information in different physical documents.

[]

Have you ever felt the need to make an association

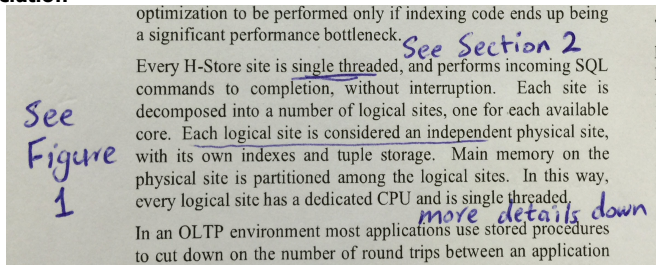
between different *parts of a physical document*

(e.g. between different paragraphs or sections)?

*

Please choose **only one** of the following:

- ☐ Yes
☐ No



The association between different pieces of information can, for example, be indicated with a comment, a note or an arrow between the different parts as shown in the figure on the right. **The association should contain references to one or all the associated (parts of) documents.**

[] How often do you feel the need to associate different *parts of a physical document*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '5 [1]' (Have you ever felt the need to make an association between different parts of a physical document (e.g. between different paragraphs or sections)?)

Please choose **only one** of the following:

- ☐ Very frequently
☐ Frequently
☐ Occasionally
☐ Rarely
☐ Very rarely

[] How do you create associations between the corresponding *parts of a physical document*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '5 [1]' (Have you ever felt the need to make an association between different parts of a physical document (e.g. between different paragraphs or sections)?)

Please choose **all** that apply:

- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each part*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each part*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Other:

An annotation should contain a reference to the other associated part.

[]Have you ever felt the need to make an association between different *parts of different physical documents* (e.g. between paragraphs of different documents)? *

Please choose **only one** of the following:

- ☐ Yes
☐ No

The association between different pieces of information can, for example, be indicated with a comment or a note. **The association should contain references to one or all the associated (parts of) documents.**

[]How often do you feel the need to associate *parts of different physical documents* (e.g. between paragraphs of different documents)? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '8 [2]' (Have you ever felt the need to make an association between different parts of different physical documents (e.g. between paragraphs of different documents)?)

Please choose **only one** of the following:

- ☐ Very frequently
☐ Frequently
☐ Occasionally
☐ Rarely
☐ Very rarely

[]How do you create associations between the corresponding *parts of different physical documents*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '8 [2]' (Have you ever felt the need to make an association between different parts of different physical documents (e.g. between paragraphs of different documents)?)

Please choose **all** that apply:

- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each part*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to one of the parts*, explicitly indicating the association between the different parts.
- ☐ Other:

An annotation in a document should contain a reference to the other associated (part of) document.

Associating Information in Digital Documents

The following questions investigate a user's behaviour in **associating information in digital documents**. We differentiate between the **three different scenarios** shown in Figure 1.

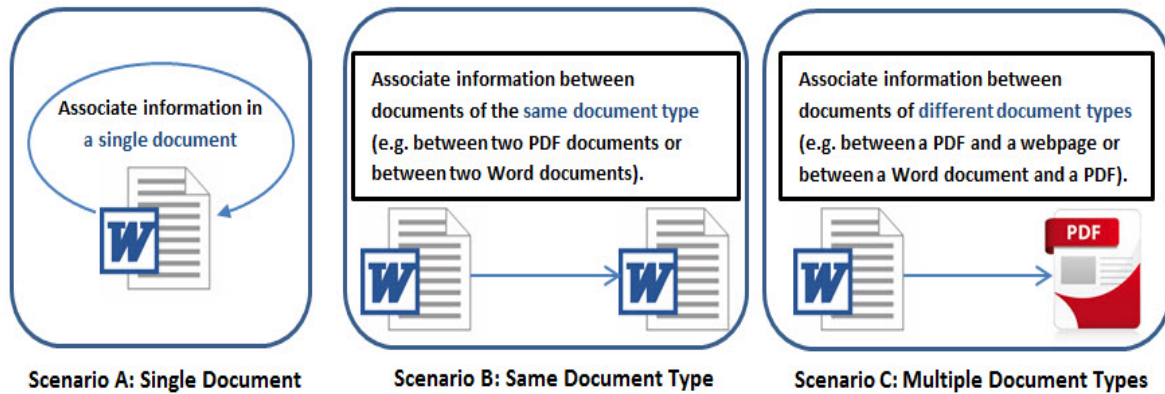


Figure 1: Three different scenarios of associating information in digital documents

Examples of associating information in digital documents are shown in Figure 2 (screenshot of a PDF document). 1 and 2 are examples of associating information in "Scenario A: Single Document". 3 is an example of associating information in "Scenario B: Same Document Type". 4 and 5 are examples of associating information in "Scenario C: Multiple Document Types".

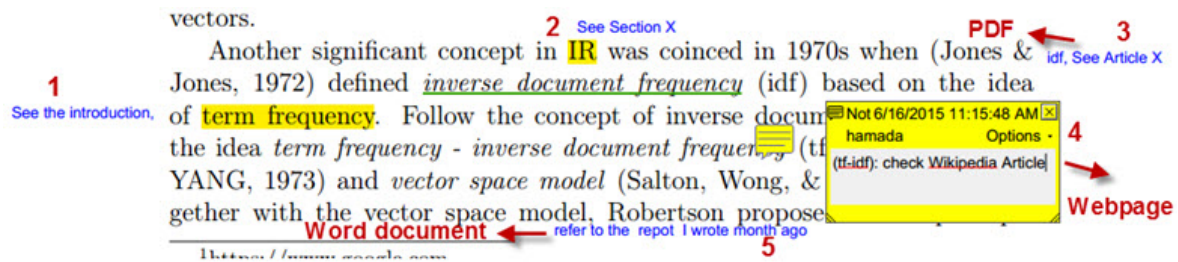


Figure 2: Screenshot of an annotated PDF document

[] Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? *

Please choose the appropriate response for each item:

	Yes	No
Scenario A [Single Document]	<input type="radio"/>	<input type="radio"/>
Scenario B [Same Document Type]	<input type="radio"/>	<input type="radio"/>
Scenario C [Multiple Document Types]	<input type="radio"/>	<input type="radio"/>

Help: The association between different pieces of information can, for example, be indicated with a comment or a note. **The association should contain references to one or all the associated (parts of) documents.**

[]How often do you feel the need to associate information in the different scenarios? *

Please choose the appropriate response for each item:

	Very frequently	Frequently	Occasionally	Rarely	Very rarely	Never
Scenario A [Single Document]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scenario B [Same Document Type]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scenario C [Multiple Document Types]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

[]In Scenario A [Single Document], how do you create associations between the corresponding *parts of a single document*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario A [Single Document]))

Please choose **only one** of the following:

- ☐ I cannot create associations.
- ☐ I create associations by using some of the document viewer's annotation features (e.g. highlighting or notes) to explicitly associate different parts of a digital document.
- ☐ I create associations in other ways.

[]In Scenario A [Single Document], I cannot create associations because

Only answer this question if the following conditions are met:

Answer was 'I cannot create associations.' at question '13 [71]' (In Scenario A [Single Document], how do you create associations between the corresponding parts of a single document?)

Please write your answer here:

[]In Scenario A [Single Document], I create associations by using some of the document viewer's annotation features (e.g. highlighting or notes) to explicitly associate different parts of a digital document

Only answer this question if the following conditions are met:

Answer was 'I create associations by using some of the document viewer's annotation features (e.g. highlighting or notes) to explicitly associate different parts of a digital document.' at question '13 [71]' (In Scenario A [Single Document], how do you create associations between the corresponding parts of a single document?)

Please choose **all** that apply:

- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to one of the parts*, explicitly indicating the association between the different parts.
- ☐ Other:

An annotation should contain a reference to the other associated part.

[]In Scenario B [Same Document Type], how do you create associations between documents of the same document type? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario B [Same Document Type]))

Please choose **all** that apply:

- ☐ Sometimes, I *store the documents in the same folder* in order to reflect the existence of the association between them.
- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I create the association by using another mechanism.
- ☐ Unfortunately, I cannot create the association between them. Please specify why?:

An annotation in a document should contain a reference to the other associated (part of) document.

[]In Scenario B [Same Document Type] , what mechanism do you use to create associations between documents of the same document type?

Only answer this question if the following conditions are met:

Answer was at question '16 [81]' (In Scenario B [Same Document Type], how do you create associations between documents of the same document type?)

Please write your answer here:

[] In Scenario B [Same Document Type], I want to create associations between different documents of the same document type because *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario B [Same Document Type]))

Please choose **all** that apply:

- ☐ Sometimes, *parts of these documents are related to each other* and therefore should be associated with each other.
- ☐ Sometimes, *parts of a document are related to an entire other document(s)* and therefore should be associated with each other.
- ☐ Sometimes, the *entire documents are related to each other* and should be associated with each other.
- ☐ Other:

You can choose multiple answers

[] In Scenario C [Multiple Document Types], how do you create associations between documents of different document types? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario C [Multiple Document Types]))

Please choose **all** that apply:

- ☐ Sometimes, I *store the documents in the same folder* in order to reflect the existence of the association between them.
- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I create the association by using another mechanism.
- ☐ Unfortunately, I cannot create the association between them. Please specify why?:

An annotation in a document should contain a reference to the other associated (part of) document.

[] In Scenario C [Multiple Document Types], what mechanism do you use to create associations between documents of different document types?

Only answer this question if the following conditions are met:

Answer was at question '19 [91]' (In Scenario C [Multiple Document Types], how do you create associations between documents of different document types?)

Please write your answer here:

[] In Scenario C [Multiple Document Types], I want to create associations between different documents of different document types because *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario C [Multiple Document Types]))

Please choose **all** that apply:

- ☐ Sometimes, *parts of these documents are related to each other* and therefore should be associated with each other.
- ☐ Sometimes, *parts of a document are related to an entire other document(s)* and therefore should be associated with each other.
- ☐ Sometimes, the *entire documents are related to each other* and should be associated with each other.
- ☐ Other:

You can choose multiple answers

[] Are you satisfied with the way you create associations between different documents in the different scenarios? *

Please choose the appropriate response for each item:

	Yes	Uncertain	No
Scenario A [Single Document]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scenario B [Same Document Type]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scenario C [Multiple Document Types]	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

[] For any of the scenarios, could you please mention some reasons why you are not satisfied with the way you create associations?

Please write your answer here:

[] In any of the scenarios, do you see the need for a linking tool that easily supports the easy creation of associations between different document parts as well as the *easy navigation between these document parts*? *

Please choose **only one** of the following:

- ☐ Yes
- ☐ No

[]Do you have any suggestions for a suitable mechanism for creating associations in digital documents?

Please write your answer here:

Associating Information Across the Digital and Physical Space

[] Have you ever felt the need to associate *parts of a physical document* with *parts of a digital document* (e.g. paragraph in a printed document with a paragraph in a digital document)? *

Please choose **only one** of the following:

- ☐ Yes
☐ No

The association should contain references to one or all the associated (parts of) documents.

[] How often do you feel the need to associate *parts of physical documents* with *parts of digital documents*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '26 [c]' (Have you ever felt the need to associate parts of a physical document with parts of a digital document (e.g. paragraph in a printed document with a paragraph in a digital document)?)

Please choose **only one** of the following:

- ☐ Very frequently
☐ Frequently
☐ Occasionally
☐ Rarely
☐ Very rarely

[] How do you create associations between the corresponding parts of *the physical document* and *the digital document*? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '26 [c]' (Have you ever felt the need to associate parts of a physical document with parts of a digital document (e.g. paragraph in a printed document with a paragraph in a digital document)?)

Please choose **all** that apply:

- ☐ Sometimes, I *highlight the different parts* and *write annotations next to each part*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *highlight the different parts* and *write an annotation next to only one of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write annotations next to each of the parts*, explicitly indicating the association between the different parts.
- ☐ Sometimes, I *write an annotation next to one of the parts*, explicitly indicating the association between the different parts.
- ☐ Other:

An annotation in a document should contain a reference to the other associated (part of) document.

[] Are you satisfied with the way you create associations between *physical* and *digital* documents? *

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '26 [c]' (Have you ever felt the need to associate parts of a physical document with parts of a digital document (e.g. paragraph in a printed document with a paragraph in a digital document)?)

Please choose **only one** of the following:

- ☐ Yes
☐ No

[] Do you have any suggestions for a mechanism to associate digital and physical documents?

Only answer this question if the following conditions are met:

Answer was 'No' at question '29 [c3]' (Are you satisfied with the way you create associations between physical and digital documents?)

Please write your answer here:

Examples of Associated Information

Attach some screenshots or photos of some examples of information association in your documents or send them later if you wish to Ahmed A. O. Tayeh (Ahmed.Tayeh@vub.ac.be).

[](Optional) Attach a screenshot or a photo of an example of information association between two documents of different types (e.g. between a PDF and a Word document or between a PDF and a web page)

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario C [Multiple Document Types]))

Kindly attach the aforementioned documents along with the survey

[](Optional) Attach a screenshot or a photo of an example of information association between two documents of the same type (e.g. between a PDF and another PDF or between a Word document and another Word document)

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario B [Same Document Type]))

Kindly attach the aforementioned documents along with the survey

[](Optional) Attach a screenshot or a photo of an example of information association in a single document (physical or digital)

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '11 [1]' (Have you ever felt the need to associate information (e.g. different paragraphs, sections or arbitrary parts) in the different scenarios? (Scenario A [Single Document]))

Kindly attach the aforementioned documents along with the survey

Participating in an Interview Later on

[] If you would not mind participating in an interview later on, please provide us your email address and [participate in a lottery for five 30 EUR Amazon vouchers](#) (of course your email will not be given to any third party).

Please write your answer here:

Thank you for completing our survey.

For additional information, please feel free to contact Ahmed A. O. Tayeh (Ahmed.Tayeh@vub.ac.be)

01-27-2016 – 16:20

Submit your survey.

Thank you for completing this survey.

B

Abstract DefaultDocument Class for Visual Plug-ins

```
1 package org.userinterface.localvisualplugins;
2
3 import java.awt.GridLayout;
4 import java.util.HashMap;
5 import java.util.HashSet;
6 import javax.swing.JPanel;
7 import org.rsl.core.Entity;
8 import org.rsl.core.Resource;
9 import org.rsl.userInterface.util.Anchor;
10
11
12 /**
13  * Abstract DefaultDocument class for visual plug-ins
14  * @author Ahmed A. O. Tayeh, ahmed.tayeh@vub.ac.be
15  */
16
17 public abstract class DefaultDocument extends JPanel{
18     static HashSet <DefaultDocumentListener> listeners;
19
20     public DefaultDocument (){
21         listeners = new HashSet <DefaultDocumentListener>();
22         this.setLayout(new GridLayout(0,1));
23     }
24
25     /**
```

```

26  * @return: Returns the visualised document
27  */
28  public abstract Resource getResource();
29
30  /**
31  * @param resource: The document to be visualised
32  */
33  public abstract void setResource(Resource resource);
34
35  /**
36  * Update a selector or a resource (document)
37  * @param en: The entity to be updated
38  */
39  public void updateEntity(Entity en){
40      this.fireUpdateEntity(new DefaultDocumentEvent(this), en);
41  }
42
43  /**
44  * Delete a selector or a resource
45  * @param en: The entity to be deleted
46  */
47  public void deleteEntity(Entity en){
48      this.fireDeleteEntity(new DefaultDocumentEvent(this), en);
49  }
50
51  /**
52  * Set the document's anchors
53  * @param anchors: The document's anchors
54  */
55  public abstract void setAnchors(HashSet <Anchor> anchors);
56
57  /**
58  * Get the document's anchors
59  */
60  public abstract HashSet <Anchor> getAnchors();
61
62  /**
63  * Highlight the document's anchors
64  */
65  public abstract void highlightAnchors();
66
67  /**
68  * Don't highlight the document's anchors
69  */
70  public abstract void removeHighlights();
71
72  /**
73  * @param o: The document to be visualised
74  * @param anchors: The document's anchors
75  * @param anchor: An anchor that should be visualised in a different colour than the other
76  *                anchors
77  * @return true if the document is visualised properly otherwise return false
78  */
79  public abstract boolean openDocument(Object o, HashSet <Anchor> anchors, Anchor
    anchor);

```

```

80 /**
81  * The link browser calls this method informing the visual plug-in to update a document
82  *
83  * @param resource: The document to be updated
84  * @param anchors: The document's anchors
85  * @param operation: The operation (e.g creation of a new hyperlink or deleting a selector)
86  *                   that requires the update
87  * @param status: Is the operation was successful? If yes, then update the visualisation
88  *                   otherwise you can neglect the command.
89  * @return true if the document is updated properly otherwise return false
90  */
91 /**
92  * @return: Return the entities (e.g. selectors or the entire document) that should be added to a
93  *           new hyperlink. The HashMap should contain every entity that will be added to the new
94  *           hyperlink as well as one of three predefined values SOURCE, TARGET or
95  *           UNIDENTIFIED. These values informs the link browser to list a given entity under the
96  *           hyperlink sources or targets.
97  */
98 /**
99  * public abstract HashMap <Entity, String> getSelections();
100  */
101 /**
102  * This method should be called by a visual plug-in after loading (visualising) a document
103  * properly.
104  */
105 /**
106  * public void setLoaded(){
107  *   this.fireDocumentLoaded( new DefaultDocumentEvent(this));
108  * }
109  */
110 /**
111  * A plug-in should call this method after a user closes a document.
112  */
113 /**
114  * public void close(){
115  *   this.fireDocumentClosed(new DefaultDocumentEvent(this));
116  * }
117  */
118 /**
119  * A visual plug-in should call this method when a user starts creating selectors.
120  */
121 /**
122  * public void setActiveSelection(){
123  *   this.fireActiveSelection(new DefaultDocumentEvent(this));
124  * }
125  */
126 /**
127  * When a user navigates to a hyperlink source or target, this method has to be called by the
128  * visual plug-in
129  * @param targetEntity: The target entity to be visualised
130  */
131 /**
132  * public void getEntityTarget(Entity targetEntity){
133  *   this.fireSelectionClicked(new DefaultDocumentEvent(this), targetEntity);
134  * }
135  */
136 /**
137  * Add listeners (e.g. the link browser) to the visualised document

```

```

127  * @param listener: The listener to be added
128  */
129  public synchronized void addDefaultDocumentListener(DefaultDocumentListener listener){
130      if(listeners != null){
131          listeners.add(listener);
132      }
133  }
134
135  /**
136   * Informs listeners (e.g. the link browser) that a user wants to update an entity
137   * @param e: The event
138   * @param en: The entity to be updated
139   */
140  protected synchronized void fireUpdateEntity(DefaultDocumentEvent e, Entity en){
141      e.setEntityToUpdate(en);
142      for(DefaultDocumentListener listener: listeners){
143          listener.updateRequested(e);
144      }
145  }
146
147  /**
148   * Informs listeners (e.g. the link browser) that a user wants to delete an entity
149   * @param e: The event
150   * @param en: The entity to be deleted
151   */
152  protected synchronized void fireDeleteEntity(DefaultDocumentEvent e, Entity en){
153      e.setEntityToDelete(en);
154      for(DefaultDocumentListener listener: listeners){
155          listener.deleteRequested(e);
156      }
157  }
158
159  /**
160   * Informs listeners (e.g. the link browser) that a user closed a document
161   * @param e: The event
162   */
163  protected synchronized void fireDocumentClosed(DefaultDocumentEvent e){
164      for(DefaultDocumentListener listener: listeners){
165          listener.documentClosed(e);
166      }
167  }
168
169  /**
170   * Informs listeners (e.g. the link browser) that a document is loaded (visualised) properly
171   * @param e: The event
172   */
173  protected synchronized void fireDocumentLoaded(DefaultDocumentEvent e){
174      for(DefaultDocumentListener listener: listeners){
175          listener.documentIsLoaded(e);
176      }
177  }
178
179  /**
180   * Informs listeners (the link browser) that a user is creating a selector
181   * @param e: The event
182   */

```



```
183 protected synchronized void fireActiveSelection(DefaultDocumentEvent e){  
184     for(DefaultDocumentListener listener: listeners){  
185         listener.activeSelectionInProcess(e);  
186     }  
187 }  
188  
189 }
```

Listing B.1: The abstract DefaultDocument class for visual plug-ins

C

Gateway Interface

```
1 package org.userinterface.externalvisualplugins;
2
3 import java.util.HashMap;
4 import java.util.HashSet;
5 import org.rsl.core.Entity;
6 import org.rsl.core.Resource;
7 import org.rsl.core.Selector;
8 import org.rsl.userInterface.util.Anchor;
9 import org.json.*;
10
11 /**
12  * Gateway interface
13  * @author Ahmed Tayeh, ahmed.tayeh@vub.ac.be
14  *
15  */
16
17 public interface Gateway {
18
19     /**
20      * launch the external third-party document viewer
21      */
22     public abstract void launchApp();
23
24     /**
25      * This function should return the resource contained in any JSON message coming from the
26      * corresponding add-in.
27      * @param command: The JSON message received from the corresponding add-in
28      * @return: The resource contained in a JSON message
29      */
30     public abstract Resource getResource(JSONObject command);
```

```

31 /**
32  * When a corresponding add-in asks the link service about anchors for a given document that
   is visualised in a third-party document viewer, the received JSON message will not
   contain enough information about the document (resource). The JSON message contains
   the URI of the opened document. This function should return the URI of the opened do
   cument.
33  * @param command: The JSON message received from the corresponding add-in
34  * @return: The URI of a resource that is contained in a JSON message
35  */
36 public abstract String getURIofOpenedResource(JSONObject command);
37
38 /**
39  * This function should return the resource id.
40  * @param command: The JSON message received from the corresponding add-in
41  * @return: The Id of a resource that is contained in a JSON message
42  */
43 public abstract long getResourceId(JSONObject command);
44
45 /**
46  * This function should form a JSON message (with a request value for opening a document) in
   order to be sent to the corresponding add-in.
47  * @param res: The resource (document) to be opened
48  * @param anchors: The document's anchors
49  * @param entityToHighLight: An anchor to be highlighted with a different colour than the
   other anchors
50  * @return: The JSON message to be sent to the corresponding add-in via one of the
   communication channels
51  */
52 public abstract JSONObject openDocument (Resource res, HashSet <Anchor> anchors,
   Anchor entityToHighLight);
53
54 /**
55  * When navigating a hyperlink in a document that is visualised in a third-party document
   viewer, the JSON message contains the ID of the target. This function should return the
   ID of the target in order to visualise it.
56  * @param command: The JSON message received from the corresponding add-in
57  * @return: The ID of the target document or selector
58  */
59 public abstract long getTargetEntityID(JSONObject command);
60
61 /**
62  * Update the visualisation of a document after executing any request coming from the
   corresponding add-in (e.g. deleting a selector or creating a selector).
63  * @param res: The document to be updated
64  * @param anchors: The document's anchors
65  * @param operation: The operation (e.g creation of a new hyperlink or deleting a selector)
   that requires the update
66  * @param status: Is the operation was successful? If yes, then update the visualisation
   otherwise you can neglect the command.
67  * @return: The JSON message to be sent to the corresponding add-in via one of the
   communication channels
68  */
69 public abstract JSONObject updateView(Resource res, HashSet <Anchor> anchors, String
   operation, boolean status);
70
71 /**

```

```

72  *Return the entities (e.g. selectors or the entire document) that should be added to a new
    *hyperlink. The HashMap should contain every entity that will be added to the new
    *hyperlink as well as one of three predefined values SOURCE, TARGET or
    *UNIDENTIFIED. These values informs the link browser to list a given entity under the
    *hyperlink sources or targets.
73  * @param command: The JSON message received from the corresponding add-in
74  * @return: a HashMap of entities and their roles in a hyperlink
75  */
76  public abstract HashMap <Entity, String> deserialiseSelections(JSONObject command);
77
78  /**
79  * This function should return the id of the entity to be updated or deleted
80  * @param command: The JSON message received from the corresponding add-in
81  * @return: The ID of the entity to be updated to deleted
82  */
83  public abstract long getIdOfEntityToUpdateOrDelete(JSONObject command);
84
85  /**
86  * This function should update an entity. When the previous function is called by the link
    *service, the link service will use the id in order to retrieve the entity. The entity will be
    *passed to this function along with the JSON command received from the corresponding
    *add-in. This function should use the JSON message to update the entity
87  * @param command: The JSON message received from the corresponding add-in
88  * @param en: The entity that has to be updated
89  * @return true if the entity is updated properly otherwise return false
90  */
91  public abstract boolean updateEntity(JSONObject command, Entity en);
92
93  }

```

Listing C.1: The Gateway interface for integrating document formats that are visualised via their third-party document viewers

D

Link Service Evaluation Questionnaire

The Link Service Evaluation Questionnaire

General Information:

What is your age?

What is your gender?

What is the highest level of education you have achieved?

What is your field of study?

Usability Evaluation

		(NA=not appropriate)							
		1	2	3	4	5	6	7	NA
1. Overall, I am satisfied with how easy it is to use this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
2. It was simple to use this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
3. I can effectively complete my work using this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
4. I am able to complete my work quickly using this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
5. I am able to efficiently complete my work using this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
6. I feel comfortable using this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
7. It was easy to learn to use this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
8. I believe I became productive quickly using this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
9. The system gives error messages that clearly tell me how to fix problems	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
10. Whenever I make a mistake using the system, I recover easily and quickly	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
11. The information (such as labels, icons, messages, and other documentation) provided with this system is clear	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
12. It is easy to find the information I needed	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
13. The information provided for the system is easy to understand	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
14. The information is effective in helping me complete the tasks and scenarios	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
15. The organization of information on the system screens is clear	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>
16. The interface of this system is pleasant	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree <input type="radio"/>

17. I like using the interface of this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree	<input type="radio"/>
18. This system has all the functions and capabilities I expect it to have	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree	<input type="radio"/>
19. Overall, I am satisfied with this system	strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	strongly agree	<input type="radio"/>
		1	2	3	4	5	6	7		NA

Bibliography

- [1] Open eBook Publication Structure 1.2, August 2002.
- [2] A. Adler, A. Gujar, B. L. Harrison, K. O'Hara, and A. Sellen. A Diary Study of Work-Related Reading: Design Implications for Digital Reading Devices. In *Proceedings of CHI 1998, ACM SIGCHI Conference on Human Factors in Computing Systems*, Los Angeles, USA, April 1998.
- [3] Adobe Systems Incorporated. Adobe Portable Document Format Reference, 6th Edition, Version 1.7, November 2006.
- [4] M. Agosti, L. Benfante, and N. Orio. IPSA: A Digital Archive of Herbals to Support Scientific Research. In *Proceedings of ICADL 2003, 16th International Conference on Asian Digital Libraries*, Kuala Lumpur, Malaysia, December 2003.
- [5] M. Agosti and N. Ferro. Annotations: Enriching a Digital Library. In *Proceedings of ECDL 2003, 7th European Conference on Research and Advanced Technology for Digital Libraries*, Trondheim, Norway, August 2003.
- [6] M. Agosti and N. Ferro. A System Architecture as a Support to a Flexible Annotation Service. In *Proceedings of the 6th Thematic Workshop of the EU Network of Excellence DELOS*, Cagliari, Italy, June 2004.
- [7] M. Agosti and N. Ferro. An Information Service Architecture for Annotations. In *Proceedings of the 6th Thematic Workshop of the EU Network of Excellence DELOS*, Cagliari, Italy, June 2004.
- [8] M. Agosti and N. Ferro. A Formal Model of Annotation of Digital Content. *ACM Transactions on Information Systems*, 26(1):3:1–3:57, 2007.

- [9] M. Agosti, N. Ferro, I. Fommoholz, and U. Thiel. Annotations in Digital Libraries and Collaboratives: Facets, Model and Usage. In *Proceedings of ECDL 2004, 8th European Conference on Research and Advanced Technology for Digital Libraries*, Bath, UK, September 2004.
- [10] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modelling in RDFS and OWL*. Morgan Kaufmann, 2011.
- [11] 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.
- [12] J. Andre, R. Furuta, and V. Quint. *Structured Documents*, chapter By Way of an Introduction. Structured Documents: What and Why?, pages 1–7. Cambridge University Press, 1989.
- [13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.
- [14] D. K. Barreau. Context as a Factor in Personal Information Management. *Journal of the American Society for Information Science*, 46(5):327–339, 1995.
- [15] D. Batory, C. Johanson, B. Macdonald, and D. V. Heeder. Achieving Extensibility Through Product-Lines and Domain-Specific Language: A Case Study. *ACM Transactions on Software Engineering and Methodology*, 11(2):191–214, 2002.
- [16] S. Björk. Hierarchical Flip Zooming: Enabling Parallel Exploration of Hierarchical Visualizations. In *Proceedings of AVI 2000, Working Conference on Advanced Visual Interfaces*, Palermo, Italy, May 2000.
- [17] J.-L. Bloechle. *Physical and Logical Structure Recognition of PDF Documents*. PhD thesis, University of Fribourg, Switzerland, 2010.
- [18] R. Boardmann, R. Spence, and A. M. Sasse. Too Many Hierarchies? The Daily Struggle for Control of the Workspace. In *Proceedings*

- of HCI 2003, 10th International Conference on Human-Computer Interaction*, Crete, Greece, June 2003.
- [19] P. Bottoni, R. Civia, L. Orso, E. Panizzi, and R. Trinchese. Digital Library Content Annotation with the MADCOW System. In *Proceedings of AVIVDiLib 2005, 7th International Workshop of the EU Network of Excellence DELOS on Audio-Visual Content and Information Visualisation in Digital Libraries*, Cortona, Italy, May 2005.
- [20] P. Bottoni, R. Civica, S. Levialdi, L. Orso, E. Panizzi, and R. Trinchese. MADCOW: A Multimedia Digital Annotation System. In *Proceedings of AVI 2004, Working Conference on Advanced Visual Interfaces*, Gallipoli, Italy, May 2004.
- [21] N. O. Bouvin. Unifying Strategies for Web Augmentation. In *Proceedings of Hypertext 1999, 10th ACM Conference on Hypertext and Hypermedia*, Darmstadt, Germany, February 1999.
- [22] P. D. Bra, G.-J. Houben, and H. Wu. AHAM: a Dexter-based Reference Model for Adaptive Hypermedia. In *Proceeding of Hypertext 1999, 10th ACM Conference on Hypertext and Hypermedia*, Darmstadt, Germany, February 1999.
- [23] P. J. Brown. Interactive Document. *Software Practice & Experience*, 16(3):292–299, 1986.
- [24] A. J. B. Brush, D. Bargerion, J. Grudin, A. Borning, and A. Gupta. Supporting Interaction Outside of Class: Anchored Discussion vs. Discussion Boards. In *Proceedings of CSCCL 2002, 8th International Conference on Computer Support for Collaborative*, Boulder, USA, June 2002.
- [25] A. J. B. Brush, D. Bargerion, A. Gupta, and J. J. Cadiz. Robust Annotation Positioning in Digital Documents. In *Proceedings of CHI 2001, ACM SIGCHI Conference on Human Factors in Computing Systems*, Seattle, USA, March 2001.
- [26] V. Bush. As We May Think. *Atlantic Monthly*, 176(1):101–108, 1945.
- [27] Y. Cai, X. L. Dong, A. Halevy, J. M. Liu, and J. Madhavan. Personal Information Management with SEMEX. In *Proceedings of*

- SIGMOD 2005, ACM SIGMOD International Conference on Management of Data*, Baltimore, USA, June 2005.
- [28] V. J. Caracelli and J. Greene. Crafting Mixed-Method Evaluation Design. *New Directions for Evaluation*, 74(19):19–32, 1997.
- [29] G. Cardone. An RSL-based Associative Filesystem. Master’s thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2010.
- [30] S. Carmody, W. Gross, T. H. Nelson, D. Rice, and A. van Dam. A Hypertext Editing System for the /360. In *Proceedings of the Second University of Illinois Conference on Computer Graphics*, Urbana, USA, January 1969.
- [31] L. Carr, D. D. Roure, W. Hall, and G. Hill. The Distributed Link Service: A Tool for Publishers, Authors and Reader. In *Proceedings of WWW 1995, 4th International World Wide Web Conference*, Beijing, China, September 1995.
- [32] R. Chatley, S. Eisenbach, J. Kramer, J. Magee, and S. Uchitel. Predictable Dynamic Plugin Systems. In *Proceedings of FASE 2004, 7th International Conference on Fundamental Approaches to Software Engineering*, Barcelona, Spain, March 2004.
- [33] R. Chatley, S. Eisenbach, and J. Magee. Modelling a Framework for Plugin. In *Proceedings of SAVCBS 2003, Workshop on Specification and Verification of Component-Based Systems*, Helsinki, Finland, September 2003.
- [34] N. Chen, F. Guimbretiere, and A. J. Sellen. Designing a Multi-Slate Reading Environment to Support Active Reading Activities. *ACM Transactions on Computer-Human Interaction*, 19(3):18:1–18:35, 2012.
- [35] N. Chen, F. Guimbretiere, and A. J. Sellen. Graduate Student Use of a Multi-Slate Reading System. In *Proceedings of CHI 2013, ACM SIGCHI Conference on Human Factors in Computing Systems*, Paris, France, April 2013.
- [36] B. G. Christensen, F. A. Hansen, and N. O. Bouvin. Xspect: Bridging Open Hypermedia and XLink. In *Proceedings of WWW 2003, 12th International World Wide Web Conference*, Budapest, Hungary, May 2003.

- [37] 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.
- [38] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. [online: <https://www.w3.org/TR/xpath>], November 1999.
- [39] A. Cockburn and B. Mckenzie. 3D or Not 3D? Evaluating the Effect of the Third Dimension in a Document Management System. In *Proceedings of CHI 2001, ACM SIGCHI Conference on Human Factors in Computing Systems*, Seattle, USA, April 2001.
- [40] A. Cockburn and B. Mckenzie. Evaluating the Effectiveness of Spatial Memory in 2D and 3D Physical and Virtual Environments. In *Proceedings of CHI 2002, ACM SIGCHI Conference on Human Factors in Computing Systems*, Minneapolis, USA, April 2002.
- [41] G. Conboy, M. Garrish, M. Gylling, W. McCoy, M. Makoto, and D. Weck. EPUB 3 Overview, Recommended Specification. [online: <http://www.idpf.org/epub/301/spec/epub-overview.html>], June 2014.
- [42] P. Constantopoulos, M. Doerr, M. Theodoridou, and M. Tzobanakis. On Information Organization in Annotation Systems. In *Proceedings of the International Workshop on Intuitive Human Interfaces for Organizing and Accessing Intellectual Assest*, Saarland, Germany, March 2004.
- [43] O. Cure and G. Blin. *RDF Database Systems: Tripels Storage and SPARGL Query Processing*. Morgan Kaufmann, 2014.
- [44] J. R. Davis and D. P. Huttenlocher. Shared Annotation for Cooperative Learning. In *Proceedings of CSCCL 1995, 1st International Conference on Computer Support for Collaborative Learning*, Bloomington, USA, October 1995.
- [45] 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.

- [46] 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.
- [47] S. DeRose and A. van Dam. Document Structure and Markup in the FRESS Hypertext System. *Markup Languages*, 1(1):7–32, 1999.
- [48] P. Dourish, W. K. Edwards, A. Lamarca, J. Lamping, K. Petersen, M. Salisbury, D. B. Terry, and J. Thornton. Extending Document Management Systems with User-Specific Active Properties. *ACM Transactions on Information Systems*, 18(2):140–170, 2000.
- [49] M. J. Durst. Internationalized Resource Identifiers: From Specification to Testing. In *Proceedings IUC19, 19th International Unicode Conference*, San Jose, USA, September 2001.
- [50] ECMA. Standard ECMA-376: Office Open XML File Formats, 3rd Edition, ECMA International, June 2011.
- [51] D. C. Engelbart and W. K. English. A Research Center for Augmenting Human Intellect. In *Proceedings of the AFIPS Joint Computer Conferences*, San Francisco, USA, December 1968.
- [52] I. Fommholz, U. Thiel, and T. Kamps. Annotation-based Document Retrieval with Four-Valued Probabilistic Datalog. In *Proceedings of WIRD 2004, 1st Workshop on the Integration of Information Retrieval and Databases*, Sheffield, UK, July 2004.
- [53] E. A. Fox, M. A. Goncalves, and R. Shen. *Theoretical Foundations for Digital Libraries: The 5S (Societies, Scenarios, Spaces, Structures, Streams) Approach*. Morgan & Claypoop, 2012.
- [54] E. Freeman and D. Gelernter. *Beyond the Desktop: Integrated Digital Work Environments*, chapter Beyond Lifestreams: The Inevitable Demise of the Desktop Metaphor, pages 19–48. MIT Press, 2007.
- [55] I. Frommholz, H. Brocks, U. Thiel, E. Neuhold, L. Innone, G. Semeraro, M. Berardi, and M. Ceci. Document-Centered Collaboration for Scholars in the Humanities: The COLLATE System. In *Proceedings of ECDL 2003, 7th European Conference on Research*

and Advanced Technology for Digital Libraries, Trondheim, Norway, August 2003.

- [56] R. Furuta. *An Integrated, but not Exact-Representation, Editor/-Formatter*. PhD thesis, University of Washington, Department of Computer Science, Seattle, USA, 1986.
- [57] R. Furuta. *Structured Documents*, chapter Concepts and Models for Structured Documents, pages 7–39. Cambridge University Press, 1989.
- [58] C. F. Goldfarb. A Generalized Approach to Document Markup. In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, Oregon, June 1981.
- [59] C. F. Goldfarb. *The SGML HandBook*. Clarendon Press, 1990.
- [60] C. F. Goldfarb and P. Paul. *The XML Handbook*. Prentice Hall PTR, 1998.
- [61] M. Golemati, A. Katifori, E. G. Giannopoulou, I. Daradimos, and C. Vassilakis. Evaluating the Significance of the Windows Explorer Visualization in Personal Information Management Browsing Tasks. In *Proceedings of IV 2007, 11th International Conference on Information Visualisation*, Zurich, Switzerland, July 2007.
- [62] J. Greene, V. J. Caracelli, and W. F. Graham. Toward a Conceptual Framework Mixed-Method Evaluation Design. *Educational Evaluation and Policy Analysis*, 11(3):255–274, 1989.
- [63] M. Greiler, H.-G. Gross, and A. V. Deursen. Understanding Plugin Test Suites from an Extensibility Perspective. Technical report, Delft University of Technology, 2010.
- [64] K. Grønbaek, N. O. Bouvin, and L. Sloth. Designing Dexter-based Hypermedia Services for the World Wide Web. In *Proceedings of the Hypertext 1997, 8th ACM Conference on Hypertext and Hypermedia*, Southampton, UK, April 1997.
- [65] K. Grønbaek, J. A. Hem, O. L. Madsen, and L. Sloth. Cooperative Hypermedia Systems: A Dexter-based Architecture. *Communications of the ACM*, 37(2):64–74, 1994.

- [66] K. Grønbaek, L. Sloth, and P. Ørbæk. Webvise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the World Wide Web. In *Proceedings of WWW 1999, 8th International World Wide Web Conference*, Toronto, Canada, May 1999.
- [67] K. Grønbaek and R. H. Trigg. Design Issues for a Dexter-based Hypermedia System. *Communication of the ACM*, 37(2):40–49, 1994.
- [68] B. J. Haan, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz. IRIS Hypermedia Services. *Communications of the ACM*, 35(1):36–51, 1992.
- [69] F. G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, 2001.
- [70] F. G. Halasz, T. P. Moran, and R. H. Trigg. Notecards in a Nutshell. In *Proceedings of CHI 1987, ACM SIGCHI Conference on Human Factors in Computing Systems*, Toronto, Canada, April 1987.
- [71] F. G. Halasz, M. Schwartz, K. Grønbaek, and R. H. Trigg. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.
- [72] R. Hall, K. Pauls, S. McCulloch, and D. Savage. *OSGi in Action*. Manning Publications, 2011.
- [73] W. Hall, H. Davis, and G. Hutchings. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers, 1996.
- [74] H. Haller and A. Abecker. iMapping: A Zooming User Interface Approach for Personal and Semantic Knowledge Management. In *Proceedings of Hypertext 2010, 21st ACM Conference on Hypertext and Hypermedia*, Toronto, Canada, June 2010.
- [75] L. Hardman, D. C. A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, 37(2):50–62, 1994.
- [76] T. Heath and C. Bizer. *Linked Data: Evolving the Web Into a Global Data Space*. Morgan and Claypool Publishers, 2011.

- [77] R. E. Herriott and W. A. Firestone. Multisite Qualitative Policy Research: Optimizing Description and Generalizability. *Educational Researcher*, 12(2):14–19, 1983.
- [78] 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.
- [79] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, 1995.
- [80] B. Johnson and B. Shneiderman. Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. In *Proceedings of VIS 1991, 2nd Conference on Visualization*, San Diego, USA, October 1991.
- [81] W. Jones, A. J. Phuwanartnurak, R. Gill, and H. Bruce. Don't Take My Folders Away! Organizing Personal Information to Get Things Done. In *Proceeding of CHI 2005, ACM SIGCHI Conference on Human Factors in Computing Systems*, Portland, USA, April 2005.
- [82] B. W. Kernighan, M. E. Lesk, and J. F. Ossanna. UNIX Time-Sharing System: Document Preparation. *The Bell System Technical Journal*, 57(6):2115–2135, July 1978.
- [83] M.-R. Koivunen. Semantic Authoring by Tagging with Annotea Social Bookmarks and Topics. In *Proceddings of SAAW 2006, 1st Semantic Authoring and Annotation Workshop*, Athens, Greece, November 2006.
- [84] H. Krottmaier and H. Maurer. Transclusions in the 21st Century. *Universal Computer Science*, 7(12):1125–1136, 2001.
- [85] J. Lamping, R. Rao, and P. Pirolli. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. In *Proceedings of CHI 1995, ACM SIGCHI Conference on Human Factors in Computing Systems*, Denver, USA, May 1995.
- [86] L. Lamport. *L^AT_EX: A Document Preparation System , User's Guide and Reference Manual*. Addison-Wesley, 2004.

- [87] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.
- [88] J. R. Lewis. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.
- [89] A. Likas, N. Vlassis, and J. J. Verbeek. The Global K-means Clustering Algorithm. *Pattern Recognition*, 36(2):451–461, 2003.
- [90] T. W. Malone. How People Organize Their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems*, 1(1):99–112, 1983.
- [91] E. B. Mandinach. The Development of Effective Evaluation Methods for E-Learning: A Concept Paper and Action Plan. *The Teachers College Record*, 107(8):1814–1836, 2005.
- [92] K. Marquardt. Patterns for Plug-Ins. In *Proceedings of EuroPloP 1999, 4th European Conference on Pattern Languages of Programs*, Irsee, Germany, July 1999.
- [93] C. C. Marshall. Toward an Ecology of Hypertext Annotation. In *Proceedings of Hypertext 1998, 9th ACM Conference on Hypertext and Hypermedia*, Pittsburgh, USA, June 1998.
- [94] 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.
- [95] H. Maurer. *Hyper-Gnow Hyperwave: The Next Generation Web Solution*. Longman Group United Kingdom, 1996.
- [96] H. Maurer. *Hyperwave: The Next Generation Web Solution*. Addison-Wesley, 1996.
- [97] J. Mayer, I. Melzer, and F. Schweiggert. Lightweight Plug-In-based Application Development. In *Proceedings of NODE 2002, International Conference NetObjectDays*, Erfurt, Germany, October 2002.
- [98] N. McKesson and A. Witwer. *Publishing with iBook Author: An Introduction to Creating Ebooks for the iPad*. OREILLY, 2012.

- [99] A. McVeigh. *A Rigorous, Architectural Approach to Extensible Applications*. PhD thesis, Imperial College London, London, UK, 2009.
- [100] D. E. Millard, L. H. N. Davis, and S. Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypert Domains. In *Proceedings of Hypertext 2000, 11th ACM Conference on Hypertext and Hypermedia*, San Antonio, USA, May 2000.
- [101] G. A. Miller. Wordnet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- [102] G. Mosweunyane, L. Carr, and N. Gibbins. *Digital Information and Communication Technology and Its Applications*, chapter A Tag-like, Linked Navigation Approach for Retrieval and Discovery of Desktop Documents, pages 692–706. Springer, 2011.
- [103] T. Nelson. *Geeks Bearing Gifts: How the Computer World Got This Way*. Mindful Press, 2009.
- [104] 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.
- [105] T. H. Nelson. *Literary Machines*. Mindful Press, 1982.
- [106] T. H. Nelson. Xanalogical Structure, Needed Now More Than Ever: Parallel Documents, Deep Links to Content, Deep Versioning, and Deep Re-use. *ACM Computing Surveys (CSUR)*, 31(4):1–32, 1999.
- [107] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, Banf, Canada, November 2006.
- [108] H. Obendorf and H. Weinreich. Comparing Link Marker Visualization Techniques: Changes in Reading Behaviour. In *Proceedings of WWW 2003, 12th International World Wide Web Conference*, Budapest, Hungary, May 2003.

- [109] K. O'Hara and A. Sellen. A Comparison of Reading Paper and On-Line Documents. In *Proceedings of CHI 1997, ACM SIGCHI Conference on Human Factors in Computing Systems*, Atlanta, USA, March 1997.
- [110] K. O'Hara, F. Smith, W. Newman, and A. Sellen. Student Readers' Use of Library Documents: Implications for Library Technologies. In *Proceedings of CHI 1998, ACM SIGCHI Conference on Human Factors in Computing Systems*, Los Angeles, USA, April 1998.
- [111] J. K. Ousterhout. Scripting: Higher-Level Programming for the 21st Century. *Comupter*, 31(3):23–30, 1998.
- [112] 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.
- [113] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A Design Science Reseach Methodology for Information Systems Research. *Journal of Management Inforamtion systems*, 24(3):45–78, 2007.
- [114] S. Pemberton, D. Austin, J. Axelsson, T. Celik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navarro, S. Peruvemba, R. Relyea, S. Schnitzenbaumer, and P. Stark. XHTMLTM 1.0: The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0. [online: <https://www.w3.org/TR/xhtml1>], January 2000.
- [115] B. K. Reid. *Scribe: A Document Specification Language and Its Compiler*. PhD thesis, Carnegie-Mellon University Computer Science Department, Pittsburgh, USA, 1980.
- [116] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. Van Dantzich. Data Mountain: Using Spatial Memory for Document Management. In *Proceedings of UIST 1998, 11th Annual ACM Symposium on User Interface Software and Technology*, San Francisco, USA, November 1998.
- [117] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *Proceedings of CHI 1991, ACM SIGCHI Conference on Human Factors in Computing Systems*, New Orleans, USA, April 1991.

- [118] G. Salton. Search Strategy and the Optimization of Retrieval Effectiveness. In *Proceedings of the FID-IFIP Conference on Mechanized Information Storage, Retrieval and Dissemination*, Amsterdam, Netherlands, June 1967.
- [119] J. Saltzer. TYPSET and RUNOFF, Memorandum Editor and Type-Out Commands. [online: <http://web.mit.edu/saltzer/www/publications/ctss/CC-244.html>], February 2003.
- [120] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Mangement*. PhD thesis, ETH Zurich, Zurich, Switzerland, 2005.
- [121] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. Books on Demand GmbH, 2008.
- [122] B. Signer. What is Wrong with Digital Documents? A Conceptual Model for Structural Cross Content Composition and Resuse. In *Proceedings of ER 2010, 29th International Conference on Conceptual Modeling*, Vancouver, Canada, November 2010.
- [123] B. Signer and M. C. Norrie. A Framework for Cross-Media Information Mangement. In *Proceesings of EuroIMSA 2005, International Conference on Internet and Multimedia Systems and Applications*, Grindelwald, Switzerland, February 2005.
- [124] 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.
- [125] B. Signer and M. C. Norrie. An Architecture for Open Cross-Media Annotation Services. In *Proceedings of WISE 2009, 10th International Conference on Web Information Systems Engineering*, Poznan, Poland, October 2009.
- [126] B. Signer and M. C. Norrie. A Model and Architecture for Open Cross-Media Annotation and Link Services. *Information Systems*, 6(36):538–550, 2011.
- [127] L. Sikos. *Mastering Structured Data on the Semantic Web: From HTML5 Microdata to Linked Open Data*. Apress, October 2015.

- [128] P. Sorotokin, G. Conboy, B. Duga, J. Rivlin, D. Beaver, K. Ballard, A. Fettes, and D. Weck. EPUB Canonical Fragment Identifier (epubcfi) Specification, Recommended Specification. [online: <http://www.idpf.org/epub/linking/cfi/epub-cfi-20140628.html>], October 2014.
- [129] M. Sporny, S. McCarron, B. Adida, M. Birbeck, and S. Pemberton. HTML+RDFa 1.1: Support for RDFa in HTML4 and HTML5. [online: <https://www.w3.org/TR/2014/PER-html-rdfa-20141216/>], March 2014.
- [130] J. Stasko and E. Zhang. Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. In *Proceedings of InfoVis 2000, IEEE Symposium on Information Visualization*, Salt Lake, USA, October 2000.
- [131] R. Strniša, P. Sewell, and M. Parkinson. The Java Module System: Core Design and Semantic Definition. In *Proceedings of OOPSLA 2007, 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, Amsterdam, Netherlands, October 2007.
- [132] M. T. Support. Rich Text Format (RTF) Specification, version 1.6. [online: <http://latex2rtf.sourceforge.net/rtfspec.html>], May 1999.
- [133] A. A.O. Tayeh. A Metamodel and Prototype for Fluid Cross-Media Document Formats. Master's thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2012.
- [134] 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.
- [135] 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.
- [136] T. N. Tran. Enhanced Retrieval and Discovery of Desktop Documents. Master's thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2015.

- [137] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.
- [138] N. Walsb. *DocBook 5 The Definitive Guide*. O'REILLY, 2010.
- [139] R. Want and S. Clara. An Introduction to RFID Technology. *Pervasive Computing*, 5(1):25–33, 2006.
- [140] R. Weir, M. Brauer, and P. Durusau. Open Document Format for Office Applications (OpenDocument) Version 1.2. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), March 2011.
- [141] S. Whittaker, L. Terveen, and B. A. Nardi. Let's Stop Pushing the Envelope and Start Addressing it: A Reference Task Agenda for HCI. *Human Computer Interaction*, 15(2):75–106, 2000.
- [142] R. Wieringa. Design Science as Nested Problem Solving. In *Proceedings of DESRIST 2009, 4th International Conference on Design Science Research in Information Systems and Technology*, Philadelphia, USA, May 2009.
- [143] R. Wolfinger and J. Kepler. Plug-in Architecture and Design Guidelines for Customizable Enterprise Applications. In *Proceedings of OOPSLA 2008, 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, Nashville, USA, October 2008.
- [144] R. K. Yin. *Case Study Research: Desing and Methods*. Sage Publications, 2009.