Cross-Media Document Linking and Navigation

Ahmed A.O. Tayeh, Payam Ebrahimi and Beat Signer
Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
atayeh,pebrahim,bsigner@vub.be

ABSTRACT

Documents do often not exist in isolation but are implicitly or explicitly linked to parts of other documents. However, due to a multitude of proprietary document formats with rather simple link models, today's possibilities for creating hyperlinks between snippets of information in different document formats are limited. In previous work, we have presented a dynamically extensible cross-document link service overcoming the limitations of the simple link models supported by most existing document formats. Based on a plug-in mechanism, our link service enables the linking across different document types. In this paper, we assess the extensibility of our link service by integrating some document formats as well as third-party document viewers. We illustrate the flexibility of creating advanced hyperlinks across these document formats and viewers that cannot be realised with existing linking solutions or link models of existing document formats. A user study further investigates the user experience when creating and navigating cross-document hyperlinks.

CCS CONCEPTS

• **Human-centered computing** → **User studies**; **User centered design**; • **Applied computing** → **Hypertext / hypermedia creation**; *Document management*;

KEYWORDS

Cross-document linking; information linking; link navigation; user linking behaviour

ACM Reference Format:
Ahmed A.O. Tayeh, Payam Ebrahimi and Beat Signer. 2018. Cross-Media Document Linking and Navigation. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209280.3209529

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DocEng '18, August 28–31, 2018, Halifax, NS, Canada
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5769-2/18/08. . . $15.00
https://doi.org/10.1145/3209280.3209529

1 INTRODUCTION

The fact that documents are often not used in isolation has been confirmed by HCI research investigating the user behaviour while reading and writing physical and digital documents [21]. According to Adler et al. [1], cross-document referencing tasks form a significant part of the entire reading and writing activity. Furthermore, Marshal [13] stated that users use highlights or underlines as anchors while reading, in order to explicitly associate (link) information within a document. Other research in the domain of Personal Information Management (PIM) identified that users make use of digital and physical folders for organising and relating (associating) information across documents [10].

Associative hyperlinks are considered the basis for creating and managing relations and associations between documents as well as other information objects [9]. They were originally introduced by Bush [3] and according to Nelson [15] they enable non-sequential reading and writing. Hyperlinks were seminal for succeeding digital hypermedia models and systems such as the oN-Line System (NLS) [5] and further instrumental in the success of the World Wide Web by enabling the referencing and annotation of content.

Most recent digital document formats support simple forms of linking (i.e. embedded and unidirectional associative hyperlinks) allowing users to associate information across different documents. While many document formats offer the possibility to link to third-party documents, it is normally not possible to address parts of these documents [17]. For example, a developer might create hyperlinks in an HTML document targeting entire PDF or text documents but it is impossible to address parts of these documents. Many solutions including open hypermedia systems (e.g. Sun's Link Service [16] or Microcosm [8]), annotation systems (e.g. MADCOW [2]) or the XLink standard[1] have been proposed in order to address the shortcomings of the link models of existing document formats. Nevertheless, as explained in [22, 24], most existing linking solutions have two major shortcomings. First of all, they only support the linking across a predefined set of document formats that have to be visualised and authored within the link service itself. Users therefore have to leave their preferred third-party document viewers and editors (e.g. Microsoft Word) in order to profit from the features offered by a link service. Second, it is not evident how the architectures of existing linking solutions can be extended in order to support other document formats.

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

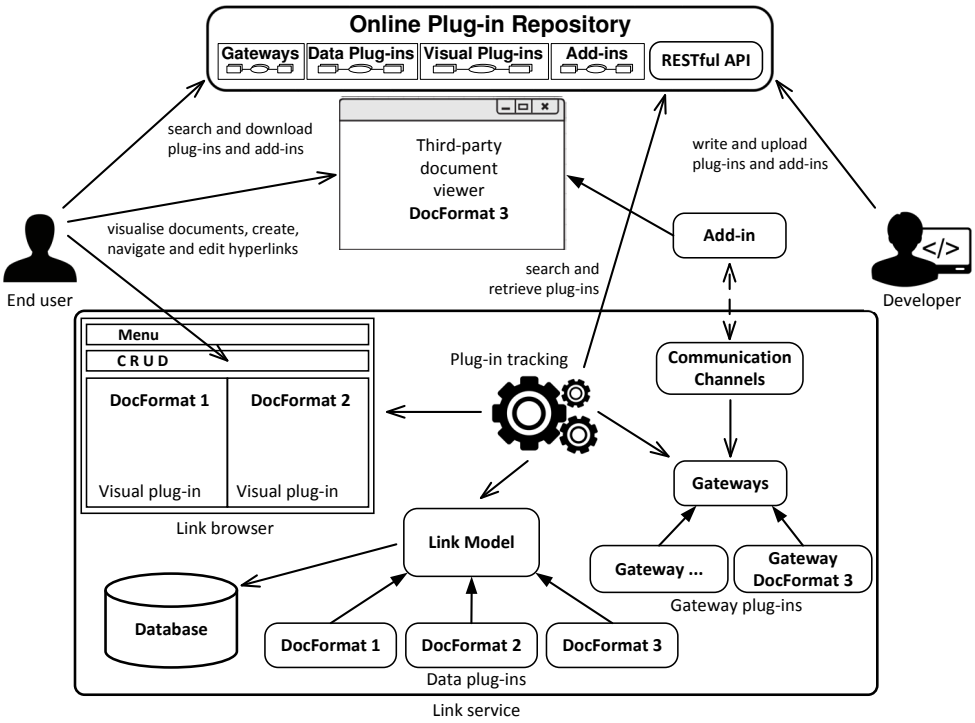

Figure 1: Conceptual schema of the dynamically extensible cross-document link service

In order to overcome the shortcomings of link models in existing document formats as well as to enable users in linking pieces of information across different document types, we have presented the dynamically extensible cross-document link service [23, 24]. The link service enables users to create external bi- and multidirectional hyperlinks within and across documents. It is further dynamically extensible via plug-ins to support existing as well as future document formats. In this paper, we assess the extensibility of our link service by integrating various document formats and viewers, including plain text, PDF, XML, Microsoft Office and Google Chrome. Moreover, we discuss the possibility of integrating other document formats and viewers. We present a user study investigating the user experience in creating and navigating bi- and multidirectional hyperlinks across some document formats and viewers as well as in extending our link service to support other document formats or viewers.

We begin in Section 2 by giving a brief overview of our link service and provide details about the advanced form of hyperlinks supported by our link service in Section 3. In Section 4 we discuss the integration of a number of document formats and viewers into our link service which is followed by an end-user evaluation in Section 5. After a critical discussion of the presented approach and study results, we provide some concluding remarks.

2 CROSS-DOCUMENT LINK SERVICE

The simple linking features offered by most existing document formats only allow users with editing permissions to the source document (e.g. the owner of a document) to create new hyperlinks. For example, the hyperlinks in an HTML web page 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. As mentioned earlier, the linking solutions that have been proposed to overcome the shortcomings of the linking features of different document formats exposed some shortcomings in terms of integrating other document formats or viewers. For example, in order to support linking across documents that are visualised in their own third-party document viewers by using Sun's Link Service [16], the third-party document viewers have to be rewritten to take into account Sun's Link Service library that facilitates the communication across different document viewers.

In earlier work [23, 24], we have presented a dynamically extensible cross-document link service addressing the shortcomings of existing document link models and linking solutions. Figure 1 illustrates the conceptual schema of our link service offering a plug-in architecture to integrate different document formats as well as third-party document viewers. As discussed in [22], in most design decisions we took into account that our link service should be extensible by third-party developers to support existing as well as emerging document formats.

The link model of our link service is based on the RSL hypermedia metamodel [18] and its idea of linking arbitrary entities, whereby an entity can either be a *resource*, a *selector* or a *link*. A resource represents the base unit for a given media type, such as an image, a video or a complete document. A selector is always related to a resource and used to address parts of the resource. Finally, a link can either be a one-to-one, a one-to-many or a many-to-many bidirectional association between arbitrary entities. In our link service, new document formats are supported by implementing *data plug-ins* extending the RSL resources (documents) as well as selectors attached to documents in a given document format. For example, in a data plug-in for the plain text document format, a text resource (document) can be represented as a URI pointing to the document and its selector might be represented via a start and end index.

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 create, read, update and delete (CRUD) operations on hyperlinks. In this visualisation component, we have defined a list of necessary functionality required to visualise and interact with any document in a link service such as the opening of a document, the highlighting of document selectors, the creation of a selector or the navigation to a hyperlink target. For each document format to be visualised in the link browser, *a visual plug-in* has to be implemented [19, 20]. A visual plug-in for a given document format needs to visualise documents as well as their selectors. Furthermore, it has to provide the necessary functionality to create, delete and update selectors.

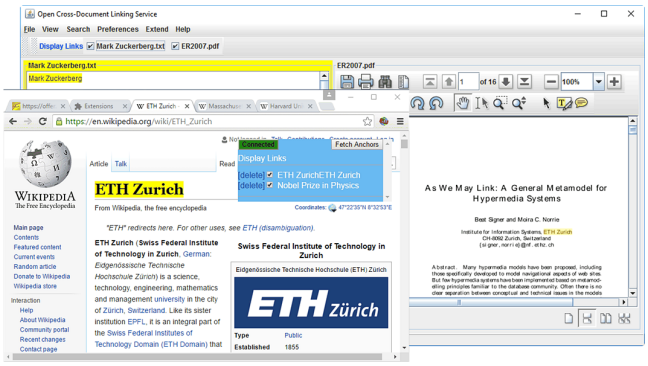

Figure 2: A bidirectional hyperlink between a PDF document visualised in the link browser and an HTML document visualised in the Google Chrome web browser

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 different documents. For example, Figure 2 shows a bidirectional hyperlink created between a PDF document visualised within our link browser and an HTML document visualised in an external web browser. We exploited extensibility features offered by third-party document viewers (i.e. their SDKs or APIs) in order to integrate them into our link service. For every third-party document viewer to be integrated with our link service, *an add-in* for the third-party document viewer is required. The add-in should provide a user interface that enables users to create, update as well as delete selectors. Furthermore, an add-in needs to communicate with our link service through any communication channel (e.g. TCP sockets, WebSockets or RESTful API) provided by the extensible communication module of our link service. Messages communicated between add-ins and our link service are represented in JSON format. The fact that document formats of third-party document viewers have different logical structures as well as selectors implies that messages exchanged between the link service and an add-in are different from messages exchanged between the link service and another add-in.

In order to cope with this challenge, we decided 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 necessary task. The gateway component has been proposed in order to facilitate the integration of third-party document viewers. All the functionality necessary to translate messages exchanged with a third-party document viewer add-in has been 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. It is worth mentioning that each message communicated between the link service and an add-in must contain a `command` key with one of the predefined request values that convey different types of interaction (e.g. creating a selector, navigating to a hyperlink target or opening a document). For more details about the message structure and request values please refer to [22].

Our link service exploits the Dynamic Module System for Java by OSGi [7] due to its dynamic extensibility as well as the management and tracking of different plug-ins. Each data, visual or gateway plug-in must contain specific key/value metadata in order to be a valid extension for our link service. The tracking component exploits the plug-in metadata in order to identify them and correctly inject a plug-in in the link service at run-time. Furthermore, the plug-in tracking component avoids plug-ins with invalid metadata. Last but not least, we proposed the online repository to store the different plug-ins and add-ins of the link service. End users should be able to support the different document formats or third-party document viewers on demand by installing the required plug-ins from the online repository. The link service enables users to do so by communicating with the online repository and shows the available plug-ins to be downloaded.

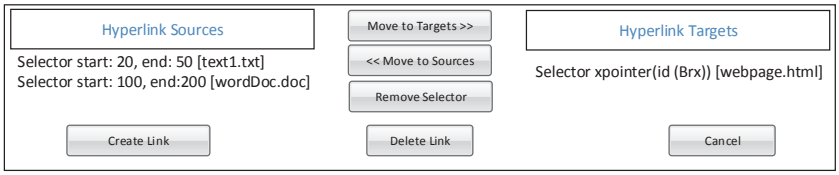

Figure 3: The hyperlink overview menu for editing a hyperlink's sources and targets

Upon a user request, the tracking module installs the different plug-ins by downloading them from the online repository. This feature is similar to the extensibility of the Eclipse[2] IDE. In the next section, we elaborate on how end users can use our link service to create advanced hyperlinks across documents to be visualised in the link browser or in third-party document viewers.

3 CREATING ADVANCED HYPERLINKS

We present three different scenarios for creating bi- and multidirectional hyperlinks by using our link service. In the first scenario, we explain the creation of a hyperlink across documents that are visualised in the link browser such as the hyperlink illustrated in Figure 4. In the second scenario, we explain the creation of the hyperlink depicted in Figure 2 where one document is visualised in our link browser and the other document is visualised in its third-party document viewer. In the last scenario, we explain the creation of a hyperlink between two different documents visualised in different third-party document viewers.

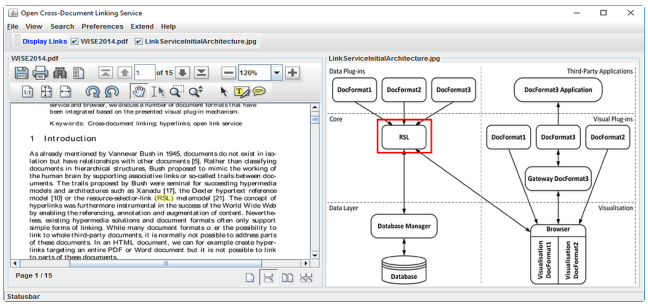

Figure 4: Bidirectional hyperlink between a PDF document on the left and a JPEG image on the right

A user can easily create bi- and multidirectional hyperlinks between documents visualised in our link browser. After opening a document in the link browser, they can select parts of the document and choose the option `Add Selector` to create a selection via the supported CRUD operations. The link browser then allows the user to choose (open) another document in order to create a target selector. The user might either confirm the creation of a new bidirectional hyperlink by pressing the `Create Link` button (i.e. provided in the link browser) or open other documents to create additional selectors resulting in 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, offering them flexibility in editing the hyperlink sources and targets. For instance, Figure 3 shows three different selectors defined 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 a multidirectional hyperlink with one source and two targets.

Similar as in the previous scenario, the user can confirm the creation of hyperlinks by using the `Create Link` button provided by the link browser or use the hyperlink overview menu shown in Figure 3 for editing and confirming the creation of hyperlinks. The hyperlink illustrated in Figure 2 can be created in a few simple steps. Let us assume that the user is working simultaneously with the PDF document visualised in the link browser and the HTML document shown in the Google Chrome web browser. The user selects the text fragment "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. As illustrated in Figure 5, the user then selects `ETH Zurich` from

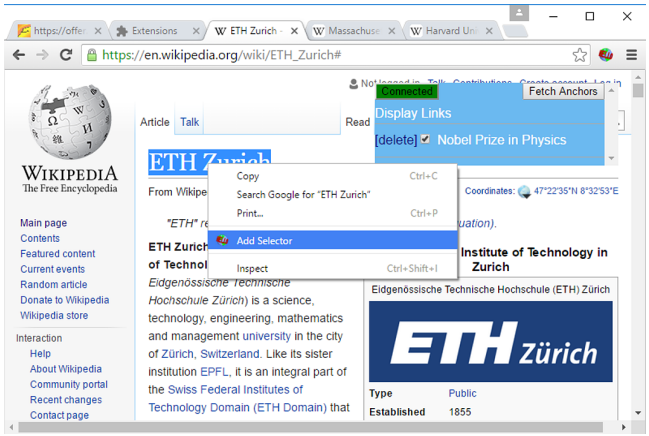

Figure 5: Creating a selector in an HTML document using a Google Chrome add-in

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

the HTML document and chooses the option to create a selector from the GUI actions supported by Google Chrome add-in. The add-in sends a JSON request 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 representing the HTML document. Further, the HTML gateway also has to transform the information about the HTML selector into an RSL selector representing the HTML selector. This means that the HTML gateway communicates with the HTML data plug-in in order to transform parts of the received message into an HTML resource or selector. The link service then adds the HTML selector to the active hyperlink under construction. The HTML selector is automatically listed under the `Hyperlink Targets` in the hyperlink overview menu. When the hyperlink is saved, the link service asks both the PDF visual plug-in and the Google Chrome add-in to update their documents in order to visualise the new hyperlink.

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 apart from Google Chrome also Adobe's Acrobat Reader is supported by the link service. The user wants to create a hyperlink between a PDF document visualised in Acrobat Reader and an HTML document rendered in Google Chrome. They select parts of the PDF document and choose the option to create a selector from the GUI actions supported by the 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, and the PDF gateway is asked to return the PDF resource and selector from the received message. As a result, the link service 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 the 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. Finally a hyperlink is created between the two documents.

4 SUPPORTED DOCUMENT FORMATS AND VIEWERS

As discussed, in order to integrate a document format with our link browser, a data as well as a visual plug-in need to be provided for the corresponding document format. The implementation of a data plug-in for a given document format is a relatively simple task since it only defines how to identify a resource (e.g. URI) and how to address its selectors (e.g. XPointer expression). Note that a data plug-in can use any existing resource identifier schema to address its resources. The implementation of a visual plug-in for a given document format requires a Java-based visualisation library for a given document format in order to correctly visualise the document format within the link browser. In the case that there is no available Java-based visualisation library for a given document format, it is still possible to implement the necessary functionality to visualise the document format from scratch. 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 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 developers to extend and customise the third-party document viewer's user interface. This feature should enable 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). Unfortunately, some existing document viewers such as iBooks Author [14] and Sumatra PDF[3] do not meet those four conditions and can therefore not be integrated with our link service.

The flexible and extensible plug-in architecture of our link service allowed us to integrate a number of document formats and document viewers as well as two multimedia content types. Our link service currently supports the linking of PDF, XML, plain text, HTML, Word and PowerPoint documents as well as images and YouTube videos. The link browser is further able to visualise PDF, XML, plain text documents or images via visual plug-ins. The HTML document format as well as YouTube videos are supported by 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.

4.1 Plain Text

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.

The visual plug-in for text documents 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

[3] http://www.sumatrapdfreader.org

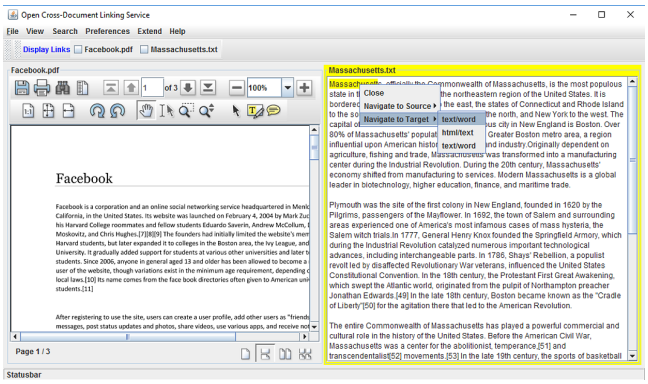

Figure 6: A rich visualisation of a multidirectional hyperlink by the plain text visual plug-in

uses the `DefaultHighlighter` component of the `swing.text` package to highlight the selectors of text documents. Last but not least, the visual plug-in provides the rich visualisation enabling users to navigate to hyperlink sources and targets as illustrated in Figure 6.

4.2 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 ICEpdf[4] open source Java library for visualising PDF documents in the link browser. The visual plug-in further uses the ICEpdf library API to get and create selections of rectangular areas in PDF documents. It is 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.

4.3 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 XInclude[5] that use XPointer, but these libraries are targeting XML inclusions (XInclude). The XML visual plug-in 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.

4.4 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 or YouTube videos. We enable the visualisation of images within our link browser by using a visual plug-in. Figure 4 illustrates a bidirectional hyperlink between a PDF document and a JPEG image. In order to render images, the visual plug-in makes use of the Java Swing library that supports a number of image formats. 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 a user's local storage and the image selector is defined as rectangular shape.

4.5 Microsoft Word

Microsoft office applications (e.g. Microsoft Word) are extensible via add-ins. Add-ins can be developed by either using a 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, at the time of our implementation it showed a number of limitations (e.g. cannot customise Microsoft Word context menu) due to restrictions of the JavaScript-based API. We therefore developed another add-in based on the C# API which is described in the following. For details about the JavaScript-based add-in please refer to [22].

4.5.1 C# Add-in. The C# API enables add-ins to customise most aspects 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. By using the API, we were able to develop a rich add-in that allows users to create and navigate advanced hyperlinks. As illustrated in Figure 7, the add-in provides a user interface on the right-hand side of a visualised document. 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 by adding the `Add Selector` command to it. In addition, users can navigate to hyperlink sources or targets via the customised context menu.

4.5.2 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 and creates JSON messages to be sent to the corresponding add-in. It also unmarshals the JSON messages

[4] http://www.icesoft.org/java/projects/ICEpdf/overview.jsf
[5] https://www.w3.org/TR/xinclude

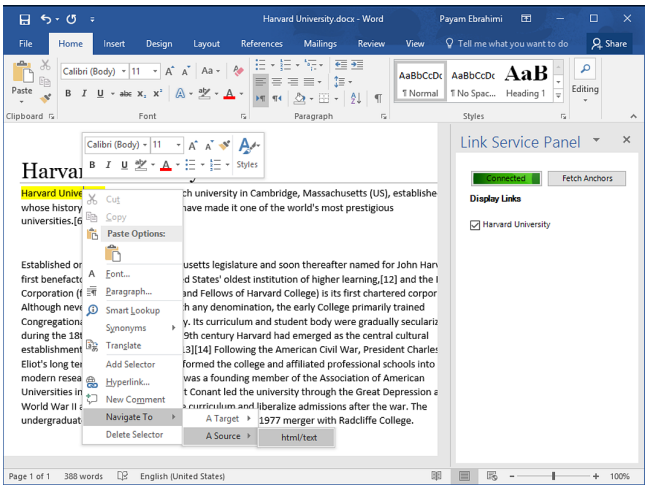

Figure 7: Microsoft Word user interface of the C# add-in

coming from the corresponding add-in to Word resources and selectors (i.e. Java objects). Note that we will not discuss the gateways for the other document viewers since they all implement the same gateway interface with similar tasks.

4.6 Microsoft PowerPoint

4.6.1 Microsoft PowerPoint Add-in. We have integrated Microsoft PowerPoint only via a JavaScript-based 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, enabling the user to update, navigate and delete them. The selector of a PowerPoint document is defined as slide ranges, meaning that a user can link to a specific slide or a range of slides (e.g. from slide 1 to slide 4).

4.6.2 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.

4.7 Google Chrome

Google Chrome is extensible via add-ins. It provides a powerful API for developing add-ins by using HTML5, CSS and JavaScript. 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 providing 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.

4.7.1 Google Chrome Add-in: HTML. The add-in user interface is visualised on the right hand-side of the Google Chrome web browser previously illustrated in Figure 5. The add-in makes use of the `WebSocket` JavaScript object in order to connect to our link service. It further uses Rangy[6], a cross-browser JavaScript library, to create, get and 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.

4.7.2 HTML Document Format Plug-ins. Two different plug-ins, a data and a gateway plug-in, have been developed for the HTML document format. The HTML data plug-in defines its resources via URIs. The selector within an HTML document is defined via an XPointer-like expression as offered by the Rangy library.

4.7.3 Google Chrome Add-in: YouTube. 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 can access the HTML5 `<video>` object offering several functions and attributes to create and play timespans within YouTube videos.

4.7.4 YouTube Video Plug-ins. As mentioned earlier, 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.

5 END-USER EVALUATION

5.1 Methodology

We evaluated the usability of our link service in terms of the ease of use, satisfaction and the quality of interactions by means of both qualitative and quantitative evaluations in order to get a better understanding of the end-users experience and gain some insights [4]. According to previous research [6, 12], using a mixed-method approach is more effective than using a single-method approach. Our 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) [11] which measures the end-user satisfaction with the usability of computer systems. CSUQ 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. Eight questions evaluate the ease of use (SYSUSE). Seven other questions evaluate the information quality (INFOQUAL) such as error messages or the documentation on how to use the system. Three other questions evaluate the interface quality (INTERQUAL), while the last question evaluates the overall satisfaction (OVERALL). We have chosen the CSUQ questionnaire for our usability evaluation due of its accepted reliability. An alpha coefficient exceeding 0.89 has been proven for all four different parts [11].

[6] https://github.com/timdown/rangy/

5.2 Population

We were mainly interested in knowledge workers' point of view and their experience when using our link service. Therefore, we choose to recruit participants among 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 (e.g. via annotations or by using bibliography reference managers such as Zotero[7]). Fourteen researchers working either on their Master's or PhD theses participated in our usability evaluation. Their age ranged from 23 to 37 years (M=27) and we recruited seven male and seven female participants for 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 Brussel (VUB) and were recruited through the "VUB experiment participant pool"[8] Facebook page which allows students to participate in research experiments and evaluations conducted at the VUB.

5.3 Study Setup

Before conducting our study, we have prepared the link service and only added support for one document format (plain text) and one third-party document viewer (Microsoft Word). We further populated the online repository with 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 Store[9] since it requires special validation from Google. We have further prepared some documents (two plain texts, two PDFs and two Word documents) to be used by the participants for creating and editing hyperlinks. Each of these documents contained at least two pages. Moreover, the set of available documents was supplemented by 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 these explanations, participants 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 plain text and Word documents. Further, they had to enable and disable the highlighting of document selectors. The participants also needed 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 questionnaire. This was followed by a semi-structured interview focusing on their perception and subjective satisfaction.

[7]https://www.zotero.org
[8]https://www.facebook.com/groups/VUB.participant.pool
[9]https://chrome.google.com/webstore/

Subscale	Statistical Indices		
	Mean	Median	SD
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 1: Summary of overall sample CSUQ

5.4 Results

In general, we received promising feedback about the usability of our link service as illustrated in Table 1 summarising the overall sample CSUQ. It further shows that the overall user satisfaction (OVERALL) was assessed very positively with a high mean and a small standard deviation (SD). 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 questionnaires 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 3 when editing their hyperlinks. All 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 (parts of) the selected paragraph or statement.

One participant complained about installing the Google Chrome add-in. Given that this participant was not a Google Chrome user, they were not aware that Google Chrome can be extended via add-ins. Moreover, for this participant it was not easy to see the main menu in Google Chrome and to perform the necessary steps for installing the add-in. The remaining thirteen participants 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 8. 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.

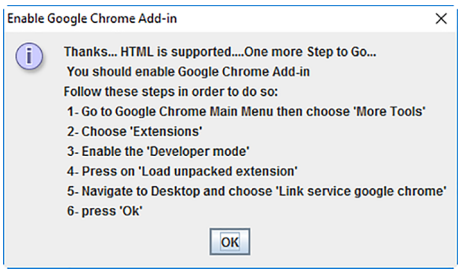

Figure 8: Guidelines for Google Chrome add-in extension

Out of the fourteen participants, ten participants have confirmed that the link service functionality is very intuitive and that they managed to become experts in using it after the creation of the first few 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 including the link browser, the Google Chrome add-in and the 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 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 tried again to create the selector by searching for the correct `Add Selector` command.

Most participants (eleven participants) confirmed that the information provided by the system—explanations for buttons on mouseover events or the detailed examples provided in the `Help` menu—was clear and helpful. Three participants who prefer explanations via videos suggested to provide a short demo video 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 to the Microsoft Word add-in, guiding users when creating their first Word selectors.

6 DISCUSSION

The integration of a number of document formats and third-party document viewers served as a technical 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 Java libraries, we could 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 has proven its flexibility and extensibility. 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 these third-party document viewers. For instance, we managed to develop two different add-ins for Microsoft Word by using two different available APIs.

Besides the aforementioned document formats and viewers, we are confident that various other document formats and viewers can also be integrated into our link service. The OOXML document format [25] has many available Java-based libraries such as Apache POI[10] and doxc4j[11]. Even though both libraries currently do not support the creation and highlighting of selectors, they can be extended to support this feature. Epublib[12] 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 the library can be extended to support the missing features and potentially be used to create a visual plug-in for the EPUB document format. Similar to Google Chrome, most web browsers such as Opera and Firefox are extensible via add-ins and can be integrated into our link service. Furthermore, the Adobe Acrobat Reader is extensible via plug-ins using the available Acrobat SDK[13]. The Acrobat SDK allows 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 using JavaScript. Finally, Apache OpenOffice can easily be

[10] https://poi.apache.org
[11] https://www.docx4java.org/trac/docx4j/
[12] https://github.com/psiegman/epublib/
[13] https://www.adobe.com/devnet/acrobat/

extended via an add-in that communicates with our link service using TCP sockets or WebSockets.

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 a major 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 is the scalability of any hypermedia solution. The use of a document archiving solution might affect the scalability of our link service.

The presented user study revealed that end users are satisfied with our link service. However, we should take into account their valuable remarks regarding the sometimes too technical naming used in our link service. In a future revision of the link service, we consider listing the hyperlink sources under the hyperlink targets as suggested by some of the participants. Moreover, we plan to create a demo to help beginners in using our link service. We conducted the presented 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. Furthermore, we are planning to integrate other document formats and viewers into our link service.

7 CONCLUSION

We have presented a technical evaluation for verifying the extensibility of our cross-document link service by integrating a number of document formats and viewers. Furthermore, we have presented an initial end-user evaluation of the presented link service revealing that participants are satisfied with the usability of our link service. The various constructive remarks might help us in enhancing the usability of our dynamically extensible cross-document link service.

ACKNOWLEDGEMENTS

We would like to thank all the study participants for their valuable feedback.

REFERENCES

[1] Annette Adler, Anuj Gujar, Beverly L. Harrison, Kenton O'Hara, and Abigail Sellen. 1998. A Diary Study of Work-related Reading: Design Implications for Digital Reading Devices. In *Proceedings of CHI 1998*. Los Angeles, USA. https://doi.org/10.1145/274644.274679

[2] Paolo Bottoni, Roberta Civica, Stefano Levialdi, Laura Orso, Emanuele Panizzi, and Rosa Trinchese. 2004. MADCOW: A Multimedia Digital Annotation System. In *Proceedings of AVI 2004*. Gallipoli, Italy. https://doi.org/10.1145/989863.989870

[3] Vannevar Bush. 1945. As We May Think. *Atlantic Monthly* 176, 1 (1945).

[4] Valerie J. Caracelli and Jennifer Greene. 1997. Crafting Mixed-Method Evaluation Design. *New Directions for Evaluation* 74, 19 (1997). https://doi.org/10.1002/ev.1069

[5] Douglas C. Engelbart and William K. English. 1968. A Research Center for Augmenting Human Intellect. In *Proceedings of AFIPS 1968*. San Francisco, USA. https://doi.org/10.1145/1476589.1476645

[6] Jennifer Greene, Valerie J. Caracelli, and Wendy F. Graham. 1989. Toward a Conceptual Framework Mixed-Method Evaluation Design. *Educational Evaluation and Policy Analysis* 11, 3 (1989). https://doi.org/10.2307/1163620

[7] Richard Hall, Karl Pauls, Sturat McCulloch, and David Savage. 2011. *OSGi in Action*. Manning Publications.

[8] Wendy Hall, Hugh Davis, and Gerard Hutchings. 1996. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers.

[9] Tomás Isakowitz, Edward A. Stohr, and P. Balasubramanian. 1995. RMM: A Methodology for Structured Hypermedia Design. *Communication of the ACM* 38, 8 (1995). https://doi.org/10.1145/208344.208346

[10] William Jones, Ammy Jiranida Phuwanartnurak, Rajdeep Gill, and Harry Bruce. 2005. Don't Take My Folders Away! Organizing Personal Information to Get Things Done. In *Proceedings of CHI 2005*. Portland, USA. https://doi.org/10.1145/1056808.1056952

[11] James R. Lewis. 1995. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction* 7, 1 (1995). https://doi.org/10.1080/10447319509526110

[12] Ellen B Mandinach. 2005. The Development of Effective Evaluation Methods for E-Learning: A Concept Paper and Action Plan. *The Teachers College Record* 107, 8 (2005). https://doi.org/10.1111/j.1467-9620.2005.00543.x

[13] Catherine C. Marshall. 1998. Toward an Ecology of Hypertext Annotation. In *Proceedings of Hypertext 1998*. Pittsburgh, USA. https://doi.org/10.1145/276627.276632

[14] Nellie McKesson and Adam Witwer. 2012. *Publishing with iBook Author: An Introduction to Creating Ebooks for the iPad*. O'Reilly.

[15] Ted H. Nelson. 1982. *Literary Machines*. Mindful Press.

[16] Amy Pearl. 1989. Sun's Link Service: A Protocol for Open Linking. In *Proceedings of Hypertext 1989*. Pittsburgh, USA. https://doi.org/10.1145/74224.74236

[17] Beat Signer. 2010. What is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse. In *Proceedings of ER 2010*. Vancouver, Canada. https://doi.org/10.1007/978-3-642-16373-9_28

[18] Beat Signer and Moira C. Norrie. 2007. As We May Link: A General Metamodel for Hypermedia Systems. In *Proceedings of ER 2007*. Auckland, New Zealand. https://doi.org/10.1007/978-3-540-75563-0_25

[19] Beat Signer and Moira C. Norrie. 2009. An Architecture for Open Cross-Media Annotation Services. In *Proceedings of WISE 2009*. Poznan, Poland. https://doi.org/10.1007/978-3-642-04409-0

[20] Beat Signer and Moira C. Norrie. 2011. A Model and Architecture for Open Cross-Media Annotation and Link Services. *Information Systems* 36, 6 (May 2011). https://doi.org/10.1016/j.is.2010.08.002

[21] Ahmed A.O. Tayeh and Beat Signer. 2018. An Analysis of Cross-Document Linking Mechanisms. In *Proceedings of JCDL 2018*. Fort Wort, USA. https://doi.org/10.1145/3197026.3197053

[22] Ahmed A.O Tayeh. 2016. *A Dynamically Extensible Cross-Document Link Service*. Ph.D. Dissertation. Vrije Universiteit Brussel.

[23] Ahmed A.O Tayeh and Beat Signer. 2014. Open Cross-Document Linking and Browsing based on A Visual Plug-in Architecture. In *Proceedings of WISE 2014*. Thessaloniki, Greece. https://doi.org/10.1007/978-3-319-11746-1_17

[24] Ahmed A.O Tayeh and Beat Signer. 2015. A Dynamically Extensible Open Cross-Document Link Service. In *Proceeding of WISE 2015*. Miami, USA. https://doi.org/10.1007/978-3-319-26190-4_5

[25] R. Weir, M. Brauer, and P. Durusau. 2011. *Open Document Format for Office Applications (OpenDocument) Version 1.2*. Technical Report. Organization for the Advancement of Structured Information Standards (OASIS).