Open Cross-Document Linking and Browsing based on a Visual Plug-in Architecture

Ahmed A.O. Tayeh and Beat Signer

Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
{atayeh,bsigner}@vub.ac.be

Abstract. Digital documents often do not exist in isolation but are implicitly or explicitly linked to parts of other documents. Nevertheless, most existing document formats only support links to web resources but not to parts of third-party documents. An open cross-document link service should address the multitude of existing document formats and be extensible to support emerging document formats and models. We present an architecture and prototype of an open cross-document link service and browser that is based on the RSL hypermedia metamodel. A main contribution is the specification and development of a visual plug-in solution that enables the integration of new document formats without requiring changes to the cross-document browser's main user interface component. The presented visual plug-in mechanism makes use of the Open Service Gateway initiative (OSGi) specification for modularisation and plug-in extensibility and has been validated by developing data as well as visual plug-ins for a number of existing document formats.

Keywords: Cross-document linking; hyperlinks; open link service

1 Introduction

As already mentioned by Vannevar Bush in 1945, documents do not exist in isolation but have relationships with other documents [5]. Rather than classifying documents in hierarchical structures, Bush proposed to mimic the working of the human brain by supporting associative links or so-called trails between documents. The trails proposed by Bush were seminal for succeeding hypermedia models and architectures such as Xanadu [18], the Dexter hypertext reference model [10] or the resource-selector-link (RSL) metamodel [22]. The concept of hyperlinks was furthermore instrumental in the success of the World Wide Web by enabling the referencing, annotation and augmentation of content. Nevertheless, existing hypermedia solutions and document formats often only support simple forms of linking. While many document formats offer the possibility to link to entire third-party documents, most of the time it is not possible to address parts of documents. In an HTML document, we can for example create hyperlinks targeting an entire PDF or Word document but it is impossible to link to parts of these documents.

Most of today's document formats offer a simple embedded unidirectional link model. Unidirectional linking implies that a target document is not aware of explicit relationships that have been defined from one or multiple source documents. Furthermore, the use of embedded links means that only the owner of a document can add new links to a document. The growth and acceptance of the Web led to an increasing number of document formats that can be rendered within a browser. More recently, documents of different formats can also be edited and stored in the cloud and we should investigate ways to link parts of documents regardless of the used storage platform.

The advent of the Extensible Markup Language (XML) in combination with its link model (XLink) has been a major step towards advanced linking on the Web. Similar to early hypermedia systems, the combination of XML and XLink can be used to separate the document content from its links. Thereby, XLink provides the typical linking functionality such as bidirectional, multi-directional, multi-source and multi-target hyperlinks. Unfortunately, XLink does not solve the problem of cross-document linking since it only deals with XML documents and does not support other non-XML document models and formats.

A flexible and extensible link model and architecture is not only required to integrate existing document types, but also to deal with new emerging document formats. It should also support more advanced linking features that are currently lacking in most document formats. In this paper, we outline a number of requirements for open cross-document linking solutions. We then present our prototype of a cross-document link service and browser which is based on the open cross-media link service architecture by Signer and Norrie [24]. We paid attention to the aspect of openness as defined by Signer and Norrie and provide a cross-document link solution that does not only provide extensibility on the model layer but also on the application layer by further investigating the concept of visual plug-ins introduced by Signer and Norrie [23].

We begin in Sect. 2 by providing an overview of different link models and mechanisms offered by existing document formats and describe a number of link services and standards. In Sect. 3, we outline the requirements for an open cross-document link service and present a cross-document link service and browser prototype. We discuss a number of data and visual plug-ins that have been developed in order to support text, PDF, HTML and XML documents. A critical discussion of the presented solution is followed by some concluding remarks.

2 Background

Despite the multitude of existing document formats and standards, most document formats adhere to conservative representations of information. As criticised by Nelson [19], the *"What You See Is What You Get"* (WYSIWYG) principle in document processing degraded the computer to a paper simulator, neglecting many features that digital document formats could offer in addition to printed paper. Moreover, many proprietary document formats prevent other documents and applications from accessing and linking to their content. The Extensible

Markup Language was an important step to open the structure of some document formats. The XML Pointer Language (XPointer) [8] and XML Path Language (XPath) [7] can be used to address parts of an XML structure, while the XLink language supports the creation of advanced hyperlinks. However, most XML-based document formats such as DocBook [25], OpenDocument [26] and OOXML [1] have sacrificed the rich linking features and adopted a simple unidirectional link model. Moreover, XML and Semantic Web technologies promote the concept of linked data [14] where data conforming to Semantic Web standards can be linked.

An overview of the support for hyperlinks in a number of popular document formats is provided in Tab. 1. Hyperlinks form a basic building block of the HTML language which offers simple typed and embedded unidirectional links to address arbitrary web resources and link to entire third-party documents (e.g. PDF or OOXML). An HTML link target is rendered via a specific web browser plug-in or opened in a third-party application based on the document's MIME type.

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

Table 1: Supported link models in existing document formats

The embedded hyperlinks in HTML web documents and other document formats prevent the management of hyperlinks separately from the underlying documents. A problem of this approach is that new hyperlinks can only be added by the author of a document. This limitation of embedded hyperlinks lead to research in open hypermedia systems where hyperlinks are managed externally in centralised databases or so-called linkbases. Intermedia [11] was an early system managing hyperlinks in an external linkbase. Another example is Microcosm [13] which offered a service for linking within arbitrary desktop applications such as AutoCAD or MS Word. Furthermore, the open hypermedia community tried to enrich the Web with external hyperlinks by considering the Web as another client for open hypermedia systems. Examples of this development include Chimera [2] or Arakne [4]. Open hypermedia systems had a significant impact in enhancing the management of hyperlinks. However, these types of systems had two major shortcomings. First of all, they did not investigate the possibilities for creating hyperlinks between snippets of information in different document formats. Second, it is not clear how to extend these systems to support cross-document linking on the model as well as on the application layer.

While the XML language does not provide a mechanism to create hyperlinks, XLink can be used to create links that go beyond simple embedded unidirectional hyperlinks. The XLink standard has been developed to improve the linking on the Web without relying on open hypermedia systems. Besides the simple unidirectional hyperlinks, XLink also supports so-called *extended hyperlinks*. With extended hyperlinks, bi- and multi-directional hyperlinks can be realised. XLink links can address web resources and parts of XML documents which means that the cross-document linking is limited to XML-based document formats.

Based on XLink and XPointer, various applications have been built to open web documents to third-party annotations and associations to external web documents. Annotea [15] is an RDF-based standard which enhances collaboration via shared web annotations and bookmarks. These annotations can be notes, explanations or comments that are externally attached to a webpage. Annotea uses XPointer to address specific parts of a webpage. A number of tools that implement the Annotea standard have been developed, including the W3C's Amaya[1] web browser or the Annozilla[2] Firefox extension. MADCOW [3] is another tool that enables the "opening" of webpages to arbitrary users. MADCOW offers richer media support than Annotea-based tools by enabling the annotation of parts of images or videos. All these annotation tools do not go beyond the features offered in XLink and hence the addressing of document parts for link sources and targets is still limited to XML-based document formats. Moreover, even if a number of these tools are extensible on the model layer, they lack extensibility on the application and visualisation layer. For example, if a new media type should be supported in MADCOW which is implemented as a monolithic component, a new version of the user interface has to be deployed.

Annotation tools for digital libraries are another trend to open different document formats for linking to external resources. An example of this family of tools is the Flexible Annotation Service Tool (FAST) [17], a standalone annotation system for digital libraries. An interesting feature of FAST is that it separates the core annotation model from the functionality offered by digital libraries. Any digital library information management system can benefit from the features of FAST by creating a new FAST interface (*gate*). Thereby, FAST does not make any assumptions about the structure of the annotated resource but defines a general *handle* concept for the resource to be annotated. Also FAST does not explicitly deal with extensibility on the application layer and is mainly targeting simple annotations rather than richer forms of hyperlinks.

Goate [16] represents another attempt to enhance the HTML link model. It is based on an HTTP proxy architecture for augmenting HTML documents with features of the XLink model such as bidirectional and n-ary hyperlinks. Similar to Goate, XLinkProxy [6] is using a web proxy to augment HTML documents with XLink features. However, like with other XLink-based solutions the linking in these systems in limited to XML-based documents.

[1] http://www.w3.org/Amaya
[2] http://annozilla.mozdev.org

The DocBook and OOXML standards only support simple unidirectional hyperlinks. DocBook hyperlinks allow us to address any web resource, entire external documents or parts of another DocBook document. Since the main goal of OOXML is to facilitate extensibility and interoperability between multiple office applications and on multiple platforms, it does not go beyond the simple unidirectional hyperlinks that were already supported in WYSIWYG office documents. With OOXML hyperlinks it is possible to address any web resource, parts of the document itself and parts of other OOXML documents.

While various approaches try to improve the linking between documents and web resources, there is still no single solution that supports arbitrary document formats (e.g. not only XML-based formats) in combination with easy extensibility. Some approaches can be extended for future emerging document formats but not without some major development efforts and the redeployment of the entire application. We present an architecture for open cross-document linking and browsing that offers rich hypermedia functionality and can be extended via data and visual plug-ins without having to redeploy the entire application.

3 Open Cross-Document Linking

We outline a number of issues that should be taken into account in order to achieve a cross-document linking solution. First of all, there exists a variety of document formats and standards including markup languages and WYSIWYG formats. These document formats have different logical representations such as linear document models, unconstrained tree-like document models or constrained heterogeneous tree-like document models as classified by Furuta [9]. Furthermore, document formats with similar document models can 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 individual characters. Moreover, many document formats suffer from the fact that they are adding new layers of complexity on top of the existing formats when new features have to be supported. The 5585 pages long specification of OOXML [1] is a testament of this never-ending growth of complexity of some document formats. The integration of cross-document linking functionality into existing document formats is a tedious and complex task since it requires some knowledge about other document formats and their logical structure. Therefore, a cross-document link service should not require any changes to the specification of existing document formats and standards. Further, a multitude of media types such as text, images, sound or video clips are supported in different document formats. These media types are informative and a selector within these media types can form the source or target of a hyperlink. One might, for instance, have a hyperlink from a selection of text in an PDF document to a specific time span of a video clip that is embedded in an HTML document. A cross-document linking solution should therefore not only specify how to address specific nodes of a document's logical structure but also define how to deal with a node's media type in order to select parts of it. All

theses issues indicate that it is impossible to make any prior assumptions about the types of linked document formats and their content. This is also one of the main reasons why we think that the XLink standard is not suitable for open cross-document linking since it assumes that the source and target documents have a tree-like document structure.

A promising approach for cross-documents linking is the use of an external link service which defines, stores and manages the general hyperlink concepts. One possible definition could be that hyperlinks can have one or more sources and one or multiple targets. The source might be a string representing an XPointer expression for tree-like documents or start and end indices for WYSIWYG formats. However, the definition of sources and targets should be kept abstract and each document format then has to 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 part can be achieved by using third-party document APIs, different programming language libraries for specific document formats or existing implementations for standards such as the Document Object Model (DOM). A link service should further be extensible to support existing as well as emerging new document formats.

An interesting idea that can be adopted to realise such a link service is the proposal of the general open cross-media annotation and link architecture by Signer and Norrie [24]. Similar to FAST, its basic idea is the separation of linking and annotation functionality from the annotated media. The annotation and link service knows how to deal with the core link and annotation model and is extensible to support new media types. This cross-media annotation and link architecture by Signer and Norrie shows a number of advantages. First of all, the annotation and link service is based on the RSL hypermedia metamodel [22] which is general and flexible enough to support evolving hypermedia systems. RSL overcomes some limitations of existing hypermedia models mixing technical and conceptual issues. The RSL model is based on the concept of linking arbitrary entities, whereby an entity can either be a resource, a selector or a link. A *resource* defines a media type such as a text, a video or a complete document. The *selector* is always attached to a resource and used to address parts of a resource. Finally, a *link* can be a one-to-one, one-to-many, many-to-one or many-to-many bidirectional association between any entities. In addition, the RSL model offers other features such as user management and support for overlapping hyperlinks via layering. To the best of our knowledge, the cross-media annotation and link architecture by Signer and Norrie [24] was the first to propose the concept of *"extensibility on the visualisation layer"* of a link service. Normally, when a link or annotation service is extended to support a new media type, also the user interface has to be extended to support the visualisation of the new media type. Signer and Norrie [24] recommend to use visual plug-ins in order to avoid a re-implementation and deployment of the entire user interface. In the remaining part of this paper, we elaborate on how we have developed a cross-document link service based on the ideas of the open cross-media archi-

tecture and discuss a number of plug-ins that have been realised so far for our cross-document link service and browser.

3.1 General Open Cross-Document Link Service Architecture

The general architecture of the link service is illustrated in Fig. 1. A central component is the core link service which is based on the RSL metamodel. The core is extensible to support arbitrary document formats by providing a *data plug-in* consisting of a media-specific implementation of the resource and selector concepts. The extensible visualisation component contains the user interface in the form of our link browser that visualises the supported document formats. For each document format to be rendered in the browser, a *visual plug-in* must be implemented. The visual plug-in has two main responsibilities. First, it has to render a specific document format and visualise any selectors that have been defined. Second, it provides a visual handle for the basic create, read, update and delete (CRUD) operations for a given document and its selectors. Taking into account that many document formats come along with their own proprietary third-party applications, visual plug-ins also have to be provided for these applications. These third-party visual plug-ins do not directly support the CRUD operations on the underlying documents but have to communicate with our link browser component in order to exchange information about selectors to be activated or created. Finally, our architecture consists of a data layer that is in charge of storing all the RSL metadata such as resources, selectors or hyperlinks.

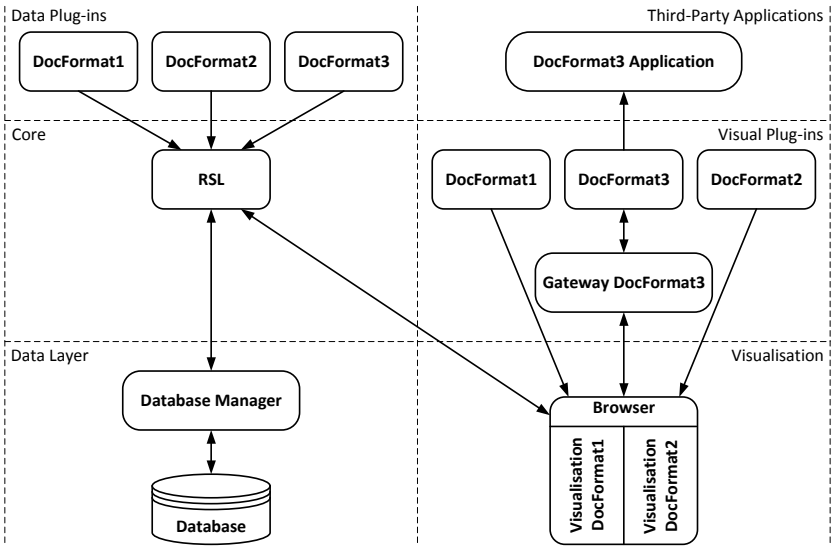

Fig. 1: General open cross-document link service architecture

We decided to make use of the Open Service Gateway initiative (OSGi) [12] for the development of the link service. The OSGi specifications defines a dynamic modular system for Java. Various server applications such as IBM Websphere apply the OSGi framework and also the Eclipse IDE uses OSGi to enable the modularisation of its components as well as to support dynamic extensions via plug-ins. We decided to use OSGi for several reasons. First of all, apart from reducing the application complexity, OSGi offers a decent mechanism for code sharing between different modules. In contrast to Java JAR files, OSGi modules do not share arbitrary code but explicitly define export packages to be shared and import packages to be used. Due to the clear definition of exported functionality, code cannot be "misused" by other installed modules. Second, the link service should be dynamically extensible to support new document formats without a need for redeployment. Dynamic extensibility also allows users to download plug-ins for new document formats on demand and thereby supports emerging document formats. The dynamic extensibility of the link service can be realised based on OSGi and its built-in support for the implementation of dynamic applications. Last but not least, our link service might provide different visual plug-ins (versions) for the same document format. Managing different versions of a module in a pure Java application often causes the so-called "JAR hell" problem, while the OSGi framework offers a specific mechanism for the versioning of modules and dependency resolution.

3.2 Communication with Plug-ins

After launching the link service and its link browser, the user can open any document in a format that is supported by the service. Thereby, documents can either be stored in the link service database or in the local file system. Figure 2 shows the main scenarios for interacting with the link service, including the opening of a document, the navigation of a link as well as the creation of a link. 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 selectors. The browser then checks the type of visual plug-in that is installed for the given document format via a registry that keeps track of all supported visual plug-ins. The browser forwards a request to the corresponding visual plug-in in order to visualise the document. If the intended visual plug-in is installed locally in the browser (`DocFormat1` in our example), it visualises the document and its selectors in a panel within the browser. On the other hand, if the plug-in is an external visual plug-in (`DocFormat3` in our example), it receives the request via a special gateway component that is responsible for launching the corresponding external third-party application and manages any communication between the visual plug-in and the browser. The external visual plug-in tells its associated third-party application to open the document and render existing selectors. The question is what should happen if a user opens a document with a supported third-party application rather than with the link browser? In this case, the external visual

plug-in notifies its gateway about the document that is currently being visualised. The gateway then communicates with the browser to retrieve potentially stored data about the currently visualised document. If any selectors have been defined for the currently opened document, the gateway returns the list of selectors to the external visual plug-in in order to visualise them.

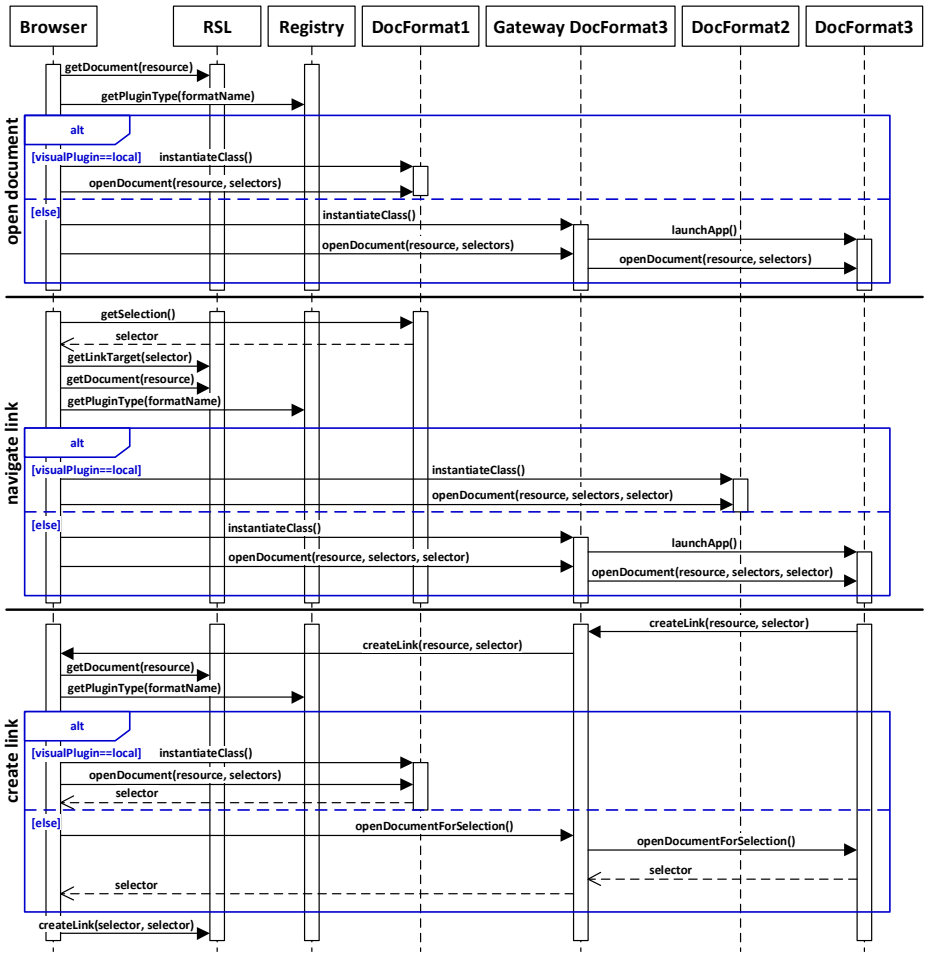

Fig. 2: Communication among different cross-document link service components

When a user clicks on a hyperlink in a document that is visualised in the browser, the browser communicates with the core RSL component to retrieve the target document and its selectors including the target selector of the selected link. The target document is then visualised with its selectors and the target selector of the followed link 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`) or one in the browser while the other document is rendered in a third-party application (`DocFormat1` and

DocFormat3). In the case that a link is selected in a document that is visualised in a third-party application, the visual plug-in sends a request to its gateway with the selected link source (selector) as a parameter. The gateway then forwards the request to the browser. From there on the browser handles the request in the same way as in the previous scenario to either visualise the link target in the browser or in a third-party application.

In the case of creating a hyperlink in a document that is visualised in a third-party application, the user selects the option of creating a hyperlink from the GUI actions supported by the visual-plug-in. The visual plug-in then sends a request to its gateway with the document and the source selector as a parameter. The gateway forwards the request to the browser which offers the possibility to open another document in order to define the link's target. The target document is then visualised in the browser or via an external visual plug-in and the browser or the external visual plug-in listens for any user selection. The user's selection is retrieved and the browser requests the creation of a hyperlink via the core RSL component. Note that the scenario shown in Fig. 2 assumes that the external documents have the same format. On the other hand, if the document is visualised in the browser, the user selects parts of the document and chooses the option to create a hyperlink from the supported browser actions. The browser handles the request similar as in the previous case by allowing the user to choose another document and listening for any target selector.

3.3 Open Cross-Document Link Service Components

A number of modules have been developed to realise the main components of the architecture. The core link service has been realised in a standalone module containing the necessary classes to implement the RSL metamodel. This module further provides a Java API for CRUD operations on resources, selectors or links. The core RSL package can be imported by any other module which is reflected by the `Export-Package: org.rsl.core` metadata in the manifest file. This allows data plug-ins to extend the RSL resource and selector concepts and the visualisation component to communicate with the core API.

The visualisation component has been realised as a standalone module. Besides the support for GUI actions for hyperlink CRUD operations and the navigation of hyperlinks between documents, this module is extensible to visualise arbitrary document formats via the plug-in mechanism mentioned earlier. The extensibility of the user interface is supported via a specific `DefaultDocument` class that has to be extended by the visual plug-ins for individual document formats. The `DefaultDocument` class extends the `JPanel` component and offers the abstract `getSelection()`, `setSelection()` and two different abstract `openDocument()` methods that are used to open a document with its selectors to either browse or highlight the target of a hyperlink. In addition to the core and visualisation module, we realised a separate module for storing documents and their hyperlinks. This module further offers the flexibility to use different database management systems for the persistent storage of documents and link

data. In our current implementation, we use the db4o[3] object database for storing system objects.

Last but not least, two plug-ins have to be provided for each document format as mentioned earlier. The data plug-ins are standalone OSGi modules that extend the core RSL module, whereas the visual plug-ins are either OSGi modules that are installed locally in the link service or extensions for third-party application user interfaces. Each external visual plug-in needs an extra module (a gateway plug-in) that needs to be installed in the OSGi platform. In order to manage the different data, gateway and visual plug-ins, we have implemented a *module tracker* which maintains a registry of supported document formats and the necessary metadata to instantiate the corresponding classes. Each plug-in has to specify its type via the `Extension-Type` which can be either `data`, `visual` or `gateway`. Furthermore, a plug-in has to specify the supported format via the `Extension-Format` key. Last but not least, local visual plug-ins use the `Extension-Class` to define the classpath of the class implementing the abstract methods of the `DefaultDocument` class. The same metadata is used in the gateway plug-in to define the classpath of the class handling the communication between the external visual plug-in and the browser.

The data plug-in for a specific document format must provide the definition of its logical structure by extending the RSL resource class and further define how to create selectors within this structure by extending the RSL selector class. The definition of the logical structure can vary even for a single document format. For example, if the link service communicates with an HTML visual plug-in that extends the FireFox web browser, a URI string is sufficient to define the HTML document resource. However, if the browser of our link service should support the HTML visualisation, sufficient information about the HTML tree syntax has to be provided.

Any local visual plug-ins has to implement the abstract methods of the `DefaultDocument` class which are used by the browser when visualising a document. Furthermore, the browser can easily retrieve or highlight a selector within a document by using the `getSelection()` and `setSelection()` methods. Thereby, third-party visualisation libraries might be used when implementing a visual plug-in. We have, for example, used the ICEpdf library[4] to implement a visual browser plug-in for PDF documents. In doing so, the visual plug-in provides a class extending the `DefaultDocument` class and acts as a proxy between the link browser and a third-party document visualisation library.

On the other hand, an external visual plug-in has to provide some methods to get and set selections in a document that is visualised in a third-party application. This can be achieved since most document applications provide their own SDKs or APIs such as the Foxit Reader Plug-in SDK[5] or Microsoft's Office Developer tools[6]. Furthermore, the external visual plug-in also has to provide

[3] http://www.db4o.com
[4] http://www.icesoft.org/java/projects/ICEpdf/overview.jsf
[5] http://www.foxitsoftware.com
[6] http://msdn.microsoft.com/en-us/library/jj620922.aspx

the GUI actions for the necessary hyperlink CRUD operation which are offered by the browser for local visual plug-ins.

Last but not least, the gateway plug-in class defined in the `Extension-Class` metadata needs to implement three methods: the `openDocumentForSelection()` method which listens for any selection in external documents as described in the link creation scenario and the two `openDocument()` methods. The gateway also has to implement methods for handling the communication between the browser and the visual plug-in.

4 HTML, PDF, Text and XML Plug-ins

Our cross-document link service currently supports the linking of HTML, PDF, text and XML documents. The browser is able to visualise XML, text and PDF documents via local visual plug-ins while the visualisation of HTML documents is delegated to the Google Chrome web browser. Figure 3 shows a bi-directional hyperlink between a PDF and XML document which are both visualised in our browser prototype as well as a bi-directional hyperlink between a PDF document and an HTML document visualised in the external web browser shown in the lower left part of Fig. 3. Note that we see the support for the XML document format as a first step towards the integration of different Open Office document formats.

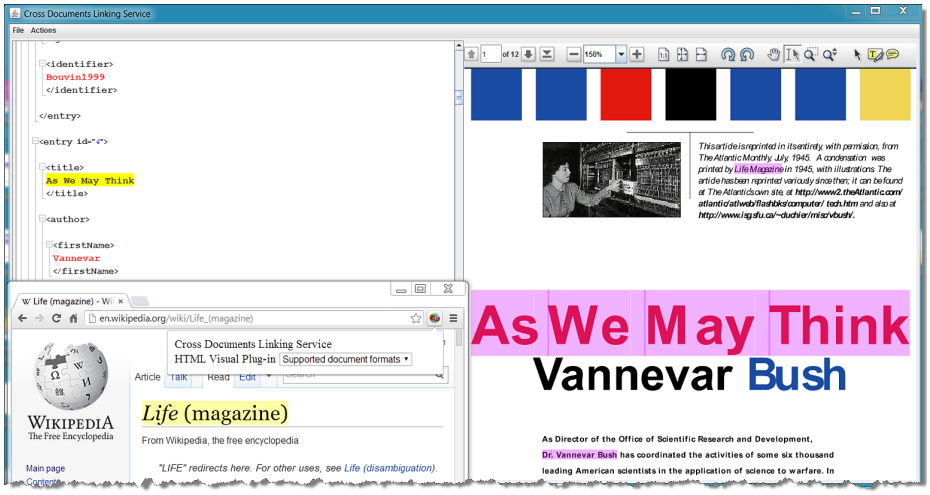

Fig. 3: Links between an XML and PDF as well as a PDF and HTML document

The data plug-in for the HTML document format defines an HTML resource by its URI, while the selector consists of an XPointer-like expression. The HTML visual plug-in has been implemented as Google Chrome extension which uses the Google Chrome API for accessing documents. Furthermore, the visual plug-in for

HTML offers a number of simple GUI actions within the Chrome browser in order to create and navigate hyperlinks. Rangy[7], a cross-browser JavaScript range and selection library, is used to retrieve (in an XPointer-like expression) and highlight selections in HTML documents. To distinguish between the hyperlinks coming from our link service and embedded HTML hyperlinks, the visual plug-in uses a different colour for the visualisation of hyperlinks. The communication between the visual plug-in and its gateway plug-in has been realised by using the WebSocket protocol.

The data plug-ins for PDF, text and XML documents specify their resources via the path and name of the documents in the user's local storage. The selector within a PDF document is defined through a page index and a rectangular area within a page. The local PDF visual plug-in uses the viewer of the ICEpdf library for the rendering of PDF documents. The visual plug-in uses the ICEpdf methods to get and create rectangular selections in a PDF document.

A selector within an XML document is defined via DOM ranges. Note that there are also some libraries that use XPointer such as XInclude, but these libraries are targeting XML inclusion. The local XML visual plug-in extends the `StyledEditorKit` component 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.

Finally, a selector within a text document is defined by the start and end indices of the selection. The local visual plug-in for text uses a `JTextPane` for the visualisation of arbitrary text documents.

5 Discussion and Future Work

The open cross-document link service and browser goes beyond the simple annotation concept offered by most annotation tools where only the reading, creation, saving, updating and retrieving of annotations is supported while support for bi-directional hyperlinks between document content is missing. Furthermore, different document formats can be integrated in our link service regardless of their document models. Two features were essential in achieving the presented prototype of an open cross-document link service. First, similar to early open hypermedia systems our cross-document link service uses an external link representation and storage. 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 and specialise the RSL selector with a definition of its selector. Moreover, the link service overcomes to some extent the issue of broken hyperlinks and consistency of hyperlinks when documents evolve. When a hyperlink source has been deleted

[7] https://code.google.com/p/rangy/

from a document, the link service automatically removes the link target from the other document. Furthermore, an archive of linked documents is a simple approach to ensure link consistency. To the best of our knowledge, the presented open cross-document link service is the first prototype to introduce flexibility and extensibility on the model as well as on the information visualisation layer as proposed by Signer and Norrie [24]. The extensibility of our open cross-document link service is further supported by the dynamic modular OSGi framework.

We also considered to realise our cross-document link service as an Open Web Platform-based solution. However, for a number of reasons we decided to go for a Java-based system rather than an Open Web Platform solution. There are currently only a limited number of Web-based open source libraries available for different document formats and most of them do not support the editing of documents. Furthermore, most web browsers offer only limited support (e.g. WebSockets) for communicating with third-party applications.

We are currently working on a dynamic plug-in extensibility where plug-ins will be automatically installed on demand. This dynamic extensibility is based on the well-known OSGi extender pattern which listens for the installation of new bundles in the OSGi platform by using a Secure Shell or Telnet protocol. Furthermore, we plan to investigate the extensibility of the link service in a study with developers and foresee to evaluate the usability of the presented solution in an end user study. Last but not least, we are planning to integrate some media plug-ins, for example for video and audio [21, 20], into our link service. These media plug-ins could then be used for addressing different media types forming part of specific documents.

6 Conclusion

We have presented a prototype of a cross-document link service and browser for integrating and linking different document formats. Based on ideas from the Open Hypermedia community and by using the RSL hypermedia metamodel, we have realised an extensible cross-media architecture. The presented cross-document link service prototype does not only support the extensibility on the data layer but more importantly also on the application and visualisation layer via visual plug-ins and a modular and extensible architecture that is based on the OSGi standard. While we have introduced various plug-ins for integrating HTML, PDF, XML and text documents with our browser as well as with external third-party applications, the presented cross-document link service and browser presents an ideal platform for investigating future innovative forms of cross-document linking.

References

1. Standard ECMA-376: Office Open XML File Formats, 3rd Edition, Jun. 2011.
2. K. M. Anderson, R. N. Taylor, and E. J. Whitehead Jr. Chimera: Hypermedia for Heterogeneous Software Development Environments. *ACM Transactions on Information Systems*, 18(2), Jul. 2000.

3. P. Bottoni, R. Civica, S. Levialdi, L. Orso, E. Panizzi, and R. Trinchese. MADCOW: a Multimedia Digital Annotation System. In *Proc. of AVI 2004*, Gallipoli, Italy, May 2004.
4. N. O. Bouvin. Unifying Strategies for Web Augmentation. In *Proc. of Hypertext 1999*, Darmstadt, Germany, Feb. 1999.
5. V. Bush. As We May Think. *Atlantic Monthly*, 176(1), 1945.
6. P. Ciancarini, F. Folli, D. Rossi, and F. Vitali. XLinkProxy: External Linkbases with XLink. In *Proc. of DocEng 2002*, McLean, USA, Nov. 2002.
7. J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0, Nov. 1999.
8. S. DeRose, E. Maler, and R. Daniel Jr. XML Pointer Language (XPointer) Version 1.0, Jan. 2001.
9. R. Furuta. Concepts and Models for Structured Documents. In *Structured Documents*. Cambridge University Press, 1989.
10. K. Grønbæk, J. A. Hem, O. L. Madsen, and L. Sloth. Designing Dexter-based Cooperative Hypermedia Systems. In *Proc. of Hypertext 1993*, Seatle, USA, Nov. 1993.
11. B. J. Haan, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz. IRIS Hypermedia Services. *Comunication of the ACM*, 35(1), 1992.
12. R. Hall, K. Pauls, S. McCulloch, and D. Savage. *OSGi in Action*. Manning Publications, 2011.
13. W. Hall, H. Davis, and G. Hutchings. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers, 1996.
14. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan and Claypool Publishers, 2011.
15. M.-R. Koivunen. Semantic Authoring by Tagging with Annotea Social Bookmarks and Topics. In *Proc. of SAAW 2006*, Athens, Greece, Nov. 2006.
16. D. Martin and H. Ashman. Goate: An Infrastructure for New Web Linking. In *Proc. of the International Workshop on Open Hypermedia Systems at HT 2002*, Maryland, USA, Jun. 2002.
17. M. A. Model, Architecturesti, and N. Ferro. A System Architecture as a Support to a Flexible Annotation Service. In *Proc. of the 6th Thematic Workshop of the EU Network of Excellence DELOS*, Cagliari, Italy, Jun. 2004.
18. T. H. Nelson. *Literary Machines*. Mindful Press, 1982.
19. T. H. Nelson. *Geeks Bearing Gifts: How the Computer World Got This Way*. Mindful Press, 2009.
20. B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. Books on Demand GmbH, May 2008.
21. B. Signer and M. C. Norrie. A Framework for Cross-Media Information Mangement. In *Proc. of EuroIMSA 2005*, Grindelwald, Switzerland, Feb. 2005.
22. B. Signer and M. C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In *Proc. of ER 2007*, Auckland, New Zealand, Nov. 2007.
23. B. Signer and M. C. Norrie. An Architecture for Open Cross-Media Annotation Services. In *Proc. of WISE 2009*, Poznan, Poland, Oct. 2009.
24. B. Signer and M. C. Norrie. A Model and Architecture for Open Cross-Media Annotation and Link Services. *Information Systems*, 6(36), May 2011.
25. N. Walsb. *DocBook 5 The Definitive Guide*. O'Reilly, 2010.
26. R. Weir, M. Brauer, and P. Durusau. Open Document Format for Office Applications (OpenDocument) Version 1.2, Mar. 2011.