A Dynamically Extensible Open Cross-Document Link Service

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. Since the introduction of the term hypertext in the early 1960s, the goal has been to link, annotate as well as transclude parts of documents. However, most existing document linking approaches show some shortcomings in terms of the offered link granularity and cannot easily be extended to support new document formats. More recently, we see new document formats such as the Office Open XML (OOXML) standard which facilitate the linking to parts of certain document formats. We present a dynamically extensible open cross-document link service enabling the linking and integration of arbitrary documents and multimedia content. In our link browser, emerging document formats are supported via visual plug-ins or by integrating third-party applications via gateways. The presented concepts and architecture for dynamic extensibility improve the document life cycle in so-called cross-media information spaces and enable future-proof cross-document linking.

Keywords: Cross-document linking, dynamic link service extensibility

1 Introduction

Most existing document formats only provide a simple embedded unidirectional link model for defining associations between documents [16, 19]. While many document formats support the linking to third-party documents, it is normally not possible to address and link to parts of these documents. For example, in a PDF document we can create hyperlinks to an entire Word document via a URI but we cannot easily link to specific parts within this document. There is no doubt that the advent of the Extensible Markup Language (XML) in combination with the link model defined by XLink has been a major step towards advanced linking on the Web. However, XLink only deals with XML-based documents and does not support other document formats or models.

We recently witnessed enhancements in various document formats for the linking of documents. For example, in the current PDF specification [1] links can be created between pieces of information stored in a PDF document via so-called `GoToE` actions. Furthermore, OOXML [2] hyperlinks enable the addressing of parts of other OOXML documents. Nevertheless, as argued by Tayeh

and Signer [19], the support of cross-document linking for arbitrary document formats is a challenging task which requires concrete information about each document format to be linked. Furthermore, it asks for a revision of a given document format specification in order to support a new document type. We aim for a link service that is flexible enough to support existing as well as emerging document formats.

The extensibility of a link service is an essential feature and should form an integral part of any document linking service. However, nowadays we can see this extensibility feature only implemented for link resolution in web browsers. When an HTML link points to a third-party document (e.g. a PDF document), based on the target's MIME type the web browser calls a specific plug-in to visualise the document. Nevertheless, the link service extensibility should not be limited to resolving links to entire documents but also allow users to create links between snippets of information in arbitrary document formats. Furthermore, a link service should take into account that users rely on proprietary applications (e.g. Microsoft Word) to author and visualise specific types of documents and address the challenge of seamlessly integrating these third-party applications. Last but not least, a linking service has to be dynamically extensible. Imagine, that each time we navigate to a new document type on the Web, we would have to install a new version of the Web browser. This would definitely be a big burden for any user. We can outline various reasons why a link service should be dynamically extensible. First of all, it is not feasible to extend or redeploy the existing service and user interface every time a new document format has to be supported. Further, each user might only make use of a small subset from the multitude of existing document formats. Rather than having a monolithic link service that supports all document formats, users should be able to dynamically extend the link service on demand in order to support their preferred document formats. Finally, offering cross-document linking features to proprietary third-party applications should not ask for changes to the core of these systems since this might not be accepted by their creators.

In this paper, we present recent extensions and adaptations of our open cross-document link service in order to deal with dynamic extensibility. We begin in Sect. 2 by providing a brief overview of our link service and highlight some of its earlier shortcomings with respect to dynamic extensibility. The essential requirements for an extensible link service are outlined in Sect. 3. The concepts and architecture to support dynamic extensibility are discussed in Sect. 4. In Sect. 5 we compare our link service solution with the existing body of work. After a critical discussion of the presented solution and an outline of future research directions, we provide some concluding remarks.

2 Cross-Document Link Service

Most document formats offer an embedded unidirectional link model, implying that only the owner of a document can add new links to the document. Furthermore, a target document is not aware of any links that have been defined

pointing to it from one or multiple source documents. Thereby, in most document formats the offered link models are far away from the "non-sequential writing" definition of hypertext [14] where at any given time pointers can be added to documents to direct a reader to a different section, paragraph or another (part of a) document. In order to overcome the shortcomings of existing document link models, in some of our earlier work we have presented an open cross-document link service [19]. The link service offers a plug-in architecture for integrating different document formats. The link model of our cross-document link service is based on the Resource Selector Link (RSL) hypermedia metamodel [18]. RSL is based on the principle 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 specific resource and is used to address parts of the resource. Finally, a link is a bidirectional association between one or multiple source entities and one or multiple target entities. In our cross-document link service, new document formats are supported by implementing *data plug-ins* that extend the RSL resource and selector concepts and contain information on how to address resources (documents) as well as selectors (anchors) attached to documents in a given document format. For example, in a data plug-in for the HTML document format, an HTML resource (document) can be represented as a URI pointing to a web resource and its selector might be represented via an XPointer-like expression.

The visualisation component of our link service consists of a link browser to visualise the supported document formats. The user interface further offers the necessary GUI actions for performing the basic CRUD (create, read, update and delete) operations on a link. For each document format to be visualised in the link browser, a *visual plug-in* has to be implemented. A visual plug-in for a given document format needs to visualise documents as well as their anchors and has to provide the necessary functionality to create, update and delete anchors. Our cross-document link service currently supports the XML, text and PDF document formats as well as general multimedia content such as images via the corresponding data and visual plug-ins. It is worth mentioning that our link service can not only deal with various document formats but also general multimedia content such as images or movies. A selector of a movie resource can, for example, be defined as a timespan. The underlying RSL metamodel is general enough to support resources and selectors of different media types and has been used before to support links across movies and web pages [17]. The visual plug-ins for various document formats visualise the documents in a Java Swing `JPanel` which can also be used to visualise arbitrary multimedia content.

We further provide a proof of concept implementation integrating a third-party application in the form of the Google Chrome browser which acts as a client for our link service. The Google Chrome plug-in has been implemented by using the Google Chrome API[1]. The communication between the link service and the Google Chrome application plug-in has been realised via a special HTML

[1] https://developer.chrome.com/extensions

gateway component. The HTML gateway implements the WebSocket communication protocol and translates any messages between the cross-document link service and the Google Chrome plug-in. Thereby, users are able to create bidirectional hyperlinks between supported document formats. Figure 1 shows a bidirectional hyperlink between a PDF document and a JPEG image which are visualised on the left and right hand side of the link browser component as well as bidirectional link between a PDF document and an HTML document visualised in the third-party Google Chrome web browser.

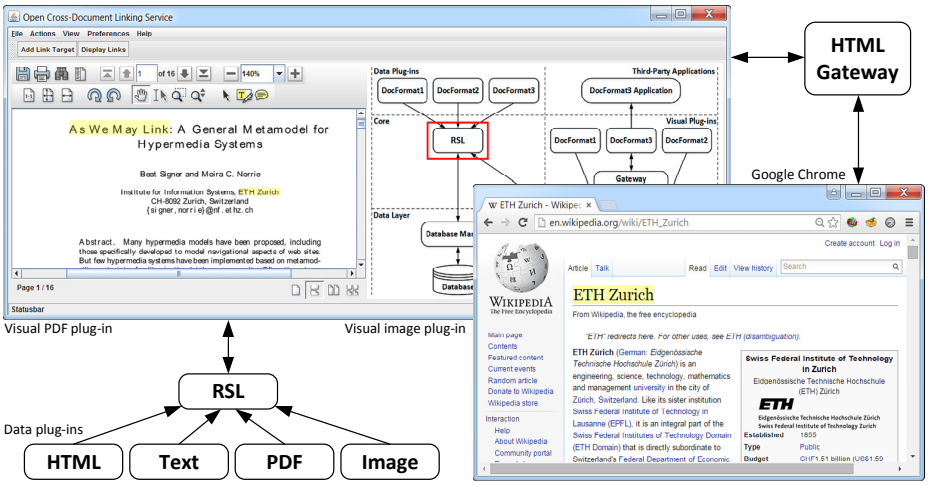

Fig. 1: Cross-document link browser with third-party application

The existing cross-document link service showed a number of shortcomings in terms of extensibility. In order to illustrate these shortcomings, we use the scenario of a user who installed the link service with support for the text, PDF, XML and HTML formats. Our user is a researcher often reading Word and PDF documents and would therefore like to add support for cross-document links between PDF and Word documents. Unfortunately, there is no online repository offering plug-ins for different document formats and our user cannot find and download the necessary plug-ins for Word. However, the researcher is lucky because they found some information on how to implement data and visual plug-ins. Moreover, they have access to the code of the HTML gateway. Our user decided to ask a developer who is familiar with Microsoft's Office Developer Tools to implement the necessary plug-ins for supporting the Word document format. When checking the code of the gateway, the developer realised that they cannot implement a new plug-in since the coupling between the communication protocol as well as the message handling in the gateway asks for a good understanding of the communication among the link service components. Furthermore, some classes and packages used in the gateway are not available online and are not accessible via public modifiers to be used outside of the HTML plug-in. Therefore, they

decided that supporting a visualisation of Word documents and their anchors within the link browser side by side with PDF documents is enough. Unfortunately, even if the developer can provide an implementation of the necessary data and visual plug-ins for Word, we still need a way that the link service can discover these new plug-ins. This scenario highlights that even if a link service offers a decent plug-in architecture, we still have to address the issue of dynamic extensibility. Users should be able to extend the link service without the need for a new release of the link service supporting the new document format. In the remaining part of this paper we describe the new essential concepts necessary for realising a dynamically extensible link service.

3 Requirements

From our previous experience with the cross-document link service and through an analysis of the shortcomings of existing link services, we derived the following fundamental requirements for an extensible cross-document link service.

Flexible and Extensible Link Service Architecture Most annotation and link services are not based on explicit link models. They often contain a mixture of conceptual and technical hard-coded link models. The fact that document formats have different document models means that anchors and selections can be defined and addressed in different ways. For example, an anchor in a tree-document model can be defined via an XPointer-like expression while an anchor in a linear text-only document model might be defined by its start and end indices. A link model should be flexible enough to deal with different document formats whereas the extensibility deals with emerging document formats to be supported at a later stage. Most annotation and link services have to be redeployed whenever a new media type should be supported. We strongly believe that the link service user interface should be extensible and support new document formats without redeploying the entire link service.

Support Multiple Document Formats This requirement can be divided into two sub-requirements: *the link service should be able to support existing document formats* and *should be able to deal with emerging document formats.* A link service should not be restricted to a fixed set of predefined document formats. However, a link service that adheres to the first requirement of a flexible and extensible link service architecture does not necessarily have to satisfy this requirement. For example, the previous version of our link service addressed the first requirement but the support for a new document format still required some intervention of the link service provider. An extensible link service should offer a simple mechanism to allow third-party developers or users to integrate additional document formats without a redeployment of the core link service.

Easy Integration of Third-Party Applications Supporting third-party applications should be taken into account in any successful link service. Otherwise, the link service would have to provide the authoring/editing of third-party documents and be appealing enough to convince users to abandon their preferred

third-party applications. Indeed, this is not practical as users might want to continue using the applications they are familiar with. Nowadays, most proprietary third-party document processing applications come along with their own Software Development Kit (SDK) in order to be extended on demand with some extra functionality. An extensible link service should benefit from a third-party application's extensibility rather than forcing third-party application vendors to rewrite their applications. Plug-ins or add-ins can be implemented for these third-party applications in order to provide visual handles for creating and editing anchors in the supported document format.

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

Customisable Link Service The previous requirements imply that the different link service components are extensible via data plug-ins, visual plug-ins or third-party application plug-ins. It is not practical to push users to install all plug-ins at once given that they might not use most of the supported document formats or third-party applications. Therefore, end users should be able to customise their link service by installing only the plug-ins for the document formats that are really needed. Customisability of the link service means that the link service is extensible on demand. The LaTeX environment which has been used to write this article is a good example for on-demand extensibility via various packages to support extra functionality. This on-demand extensibility not only saves storage space but also increases the overall performance. In order to successfully support on-demand customisation, the availability of the plug-ins should be ensured via a central online plug-in repository.

Plug-in Versioning Different document format specifications are often updated to support new features. Moreover, third-party applications are normally updated with new features to either support the new document format specification or to enable new features in the application itself. Therefore, new versions of plug-ins for some document formats are expected to be published and the link service should offer some plug-in versioning mechanism.

4 Dynamically Extensible Link Service

The overall architecture of our link service is outlined in Fig. 2. Components depicted with solid lines have been presented in earlier work [19], while some of the dashed *gateways*, *plug-in tracking*, *online repository* and *communication* components are playing a central role for the dynamic link service extensibility described in this paper.

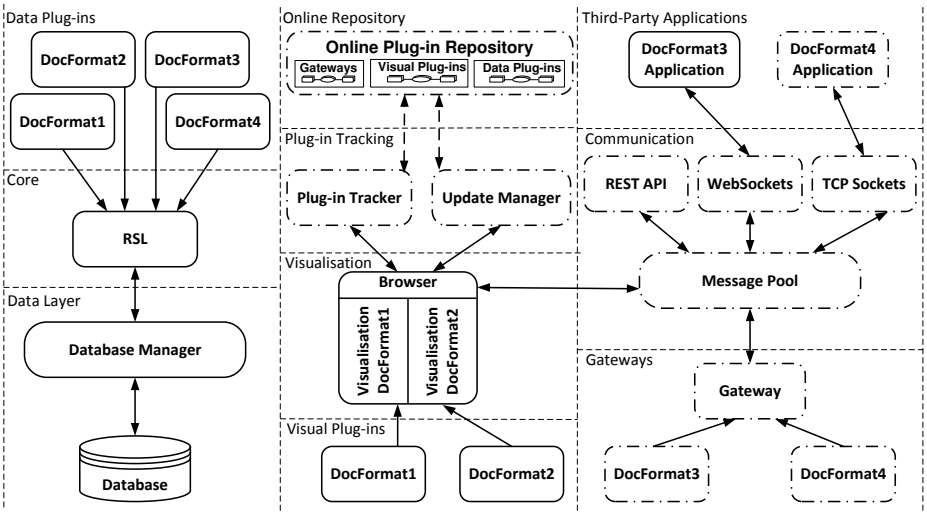

Fig. 2: Cross-document link service architecture

In our link service, the gateway and communication components play an important role for integrating third-party applications. For each document format to be integrated via a third-party application, a gateway handling the messages with the third-party application plug-in has to be provided. The third-party application plug-in can use any communication protocol supported by the link service. The plug-in tracking component consists of a plug-in tracker and an update manager. These two components are responsible for keeping track of the installed plug-ins as well as for installing new plug-ins on demand by communicating with the online plug-in repository.

The OSGi framework [10] plays an important role in achieving our link service's dynamic extensibility. The OSGi specification defines a dynamic modular system for the Java programming language but the deployment of an OSGi-based application does not differ from regular Java applications. More details about the motivation for using the OSGi framework in our link service can be found in [19]. Conceptually, the OSGi framework consists of three layers, including the *module layer*, the *life cycle layer* and a *service layer*. The module layer is responsible for packaging and sharing the application code. Each module of an application is called a *bundle* and corresponds to a Java JAR file with some extra metadata in the form of a manifest file. The life cycle layer controls specific modules at execution time. The interaction and communication between installed modules is managed by the service layer. Each component depicted by a rectangle in the link service is a bundle. All data, visual and gateway plug-ins are bundles with different metadata. The life cycle layer is extensively used by the plug-in tracker for dynamic extensibility.

In order to get a better understanding of the roles played by the different components, we come back to the scenario presented earlier, but this time we

assume that the user installed the extensible version of the link service. When the user wants to add support for the Word document format, they can either open the online repository web page to search for Word document plug-ins or search via the link service's update manager interface. The user learns that there are two options for the Word document format. The first extension visualises documents and their anchors in the link browser. This means that the extension consists of a data plug-in and a visual plug-in. A second extension supports the visualisation directly within Microsoft Word. This extension consists of a data plug-in, a gateway as well as the necessary Microsoft Word application add-in. In both cases, the user must install the data plug-in which has some metadata stored in its manifest file in order to be correctly identified and used by the link service components. The update manager reads the metadata, the plug-in is downloaded via a secure shell protocol and installed by the plug-in tracker. The plug-in tracker does the necessary work to inject the plug-in into the running link service. If the user wishes to visualise their documents within the generic link browser, the visual plug-in for the Word document type has to be installed.

In the case that the user wants to visualise their documents in Microsoft Word, they have to install the gateway plug-in in the link service and follow the instructions provided for the Microsoft Word add-in in order to extend Microsoft Word. The gateway plug-in is installed based on the same mechanism used for data and visual plug-ins. After installing the plug-ins, the user should be able to create hyperlinks in Word documents and link them to any of the already supported document formats. Let us assume that after three months the developer of the third-party Word plug-in has fixed some bugs and implemented some new nice visualisation as well as other features. They upload a new version of the plug-in to the online repository. The update manager will inform the user that a new version of a Word plug-in is available and wait for a confirmation to install the new version. In the following subsections we elaborate on the different components necessary for the dynamic extensibility described in the scenario.

4.1 Gateways

The gateway component is introduced in the link service in order to overcome the limitation of third-party application integration introduced earlier by the link service. The gateway component is the most essential component to integrate existing third-party applications and it is flexible enough to integrate any new third-party application. A third-party application extension (plug-in) acts as a client to the link service and can be implemented in any programming language supported by the third-party application SDK. The component contains an interface that provides the abstract methods needed to translate any message exchanged with a third-party application plug-in. Message translation simply involves the marshalling and unmarshalling of Java objects. JSON messages sent by an external third-party plug-in are unmarshalled into Java objects by the corresponding gateway and Java objects are marshalled to JSON objects by the gateway to be sent to its corresponding external third-party application plug-in.

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

Listing 1.1: Gateway interface

```
abstract long getResourceId(JSONObject msg);
abstract JSONObject openDocument(Resource res,
    HashSet <Anchor> anchors, Anchor entityHighlight);
abstract long getTargetEntityID(JSONObject msg);
```

In a gateway plug-in, the developer has to provide a class implementing the gateway interface. Some of the methods of the gateway interface are shown in Listing 1.1. The `getResourceId()` method is called by the link service in order to return the ID of the resource (document) included in the JSON message. The second method is used to serialise a Java object to a JSON message which can be sent to the external third-party application plug-in to open a document. The method receives the resource (document), the set of anchors contained within the document and optionally a specific anchor to be highlighted in the case that the document has to be visualised as a result of a link that has been followed. It is worth mentioning that the anchor object in the link service contains information about a link source which is either a selector or a complete document. It further contains information about the targets which can either be other documents or selectors of documents. The anchor object contains enough information about each target such as its MIME type, its ID as well as the contained document in the case of a selector, enabling the development of rich visual plug-ins. For example, multi-target links can be represented via a pop-up menu in order to give the user the flexibility to navigate to any target document.

4.2 Plug-in Metadata and Repository

The different types of plug-ins (e.g. data, visual and gateway plug-ins) require a mechanism to differentiate between them in order to correctly use them in

the link service and to correctly inject them when extending the link service. We have exploited the OSGi manifest file to correctly identify each document format plug-in. Aside from the specific OSGi metadata required by any OSGi bundle, different document format plug-ins must contain specific metadata to be a valid extension for the link service and to be correctly identified by the link service. Based on the type of the plug-in (i.e. data, visual or gateway), different metadata keys and values should be included in the plug-in. The `Extension-Name`, `Extension-Mime` and `Extension-Type` metadata is required for all types of plug-ins. This metadata provides information about the MIME type (e.g. `text/html` or `application/pdf`) supported by the plug-in, its name and its type. A plug-in developer should maintain the consistency of the MIME type provided in a document format plug-in. In other words, a data plug-in and a visual plug-in for a given document format must have the same MIME type. The same holds for data and gateway plug-ins in the case of a third-party application integration.

The link service's user interface contains an abstract class defining the necessary functions to visualise a document as well as to perform the CRUD operations for links in a given document format. A visual plug-in must extend the abstract class of the user interface component. Furthermore, each gateway has to implement the gateway interface. The link service can communicate with a visual plug-in to visualise documents or for CRUD operations on a link by instantiating the class that extends the abstract user interface class. Furthermore, the link service can communicate with a gateway plug-in in order to marshal or unmarshal objects by instantiating the class that implements the gateway interface. In order to instantiate these classes, the classpath is stored in the `Extension-Class` metadata included in the corresponding visual or gateway plug-in. Finally, the online repository provides simple interfaces to search for plug-ins and for uploading new plug-ins.

4.3 Plug-in Tracking

The plug-in tracking consists of the plug-in tracker and the update manager. The update manager is responsible for keeping track of the available document format plug-ins in the online repository by reading the metadata for every plug-in. Users can interact with the update manager by using its GUI in order to search for plug-ins of different document formats. Moreover, the update manager notifies the plug-in tracker about the availability of any document format. Last but not least, the update manager is in charge of downloading the different plug-ins.

The plug-in tracker is responsible for installing, keeping track of and managing the different plug-ins in the link service. The plug-in tracker's extender pattern listens for any OSGi bundles (plug-ins) being started or stopped in the link service. When a plug-in is installed, the tracker checks whether it is an extension based on the predefined extension metadata and performs the necessary operations to integrate it in the link service. In the case of a data plug-in, the tracker adds the MIME type to the list of supported document formats. If a visual plug-in is installed, the plug-in tracker checks whether the data plug-in for the same document format (MIME) has already been installed. In the case

that the data plug-in is missing or the user does not confirm the installation of the data plug-in, the visual plug-in will not be installed. If the data plug-in is already installed or installed after a user's confirmation, the plug-in tracker notifies the user interface component that a new visual plug-in exists which in turn injects the new plug-in into the link browser user interface. As a result, the user can see that the new document format has successfully been integrated in the link service and can start using it. For gateway plug-ins, the plug-in tracker will maintain the availability of its data plug-in with the same mechanism used when installing a visual plug-in and add it to the list of supported gateways.

4.4 Communication Protocols

In order to support as many third-party applications as possible, our link service communication component supports three different communication protocols. TCP sockets and WebSockets are used for full duplex communication channels and a REST API can be used as a fall-back solution for third-party application SDKs not offering full duplex communication. JSON messages coming from third-party applications through different communication protocols are centrally managed via the message pool component. The message pool also keeps track of all active third-party application plug-ins and their sessions in order to forward the JSON messages produced by different gateways.

Note that each JSON message exchanged with a third-party application plug-in contains a `command` key with one value for the predefined request values. For example, the `create` value is used in some messages sent by the external plug-ins and informs the message pool that the user wants to create a link in a document visualised in its third-party application. The `showTarget` value is contained in some messages sent by the external plug-ins and tells the message pool that the user clicked on a link in a document visualised in the third-party application. In this case, the JSON message should contain information about the specific link target. The message pool then asks the corresponding gateway to return the ID of the intended target by passing the JSON message to the `getTargetEntityId()` gateway method shown earlier in Listing 1.1. When the link service requires an external third-party application plug-in to open a document with its selectors, it sends the request to the message pool. The message pool then asks the corresponding gateway to marshal the object to a JSON message containing the `command` key with the `openDocument` value and the third-party application plug-in will know how to process the message.

Messages coming from third-party application plug-ins are forwarded to the user interface component via the message pool after they have been unmarshalled to Java objects by the corresponding gateway. The message pool can identify the correct gateway by using the MIME type defined by the communication session in the handshake process. Moreover, when a message has to be sent from the link service to the external third-party application plug-in, the message pool requests the gateway to marshal the message to a JSON message before forwarding it to the correct communication protocol with the active session.

5 Related Work

Based on the six requirements introduced earlier in Sect. 3, we present a comparison of some existing link services and annotation tools in Tab. 1. Each requirement is mapped to one dimension (column) in the comparison table, expect for the second requirement which is mapped to the two 'Cross-Document Linking' and 'Emerging Document Formats' dimensions. The former evaluates whether a link service supports cross-document linking between multiple existing document formats, while the latter evaluates whether a link service is extensible and might support emerging document formats. Further, the last row in the table presents our dynamic link service. We use the ✔ symbol to illustrate that a feature is supported whereas the (✔) symbol means that there is only limited support for a given feature or the feature is supported but with some major drawbacks.

Link Service	Extensible Architecture	Cross-Document Linking	Emerging Document Formats	External Applications	Flexible Channels	Customisability	Plug-in Versioning
Intermedia	(✔)	✔	✗	✗	✗	✗	✗
Sun's Link Service	(✔)	✔	(✔)	(✔)	✗	✗	✗
Microcosm	✗	✔	✗	(✔)	✗	✗	✗
Annotea Solutions	✗	✗	✗	✗	✗	✗	✗
MADCOW	✗	✗	✗	✗	✗	✗	✗
FAST	(✔)	✗	✗	(✔)	✗	✗	✗
Dynamic Link Service	✔	✔	✔	✔	✔	✔	✔

Table 1: Comparison of existing link services and annotation tools

Open hypermedia systems such as Intermedia [9], Sun's link service [15] and Microcosm [11] addressed the limitations of embedded links by managing links separately from the linked documents in so-called linkbases. Intermedia supports the linking across five different document formats but shows a number of shortcomings. Even though Intermedia is based on a layered architecture, it is not evident how it can be extended to support additional document formats. Moreover, Intermedia was intended to be used as a complete authoring tool and not purely as a link service. This implies that any document format that would like to profit from Intermedia's linking features has to be visualised and authored with Intermedia's viewers. Furthermore, features such as the integration of third-party applications, customisation and versioning have not been considered in Intermedia.

Sun's link service—a pure link service providing a protocol to communicate with external applications—shows two major shortcomings. First, the link service forms a monolithic component with a core link model that is not extensible. In other words, more advanced forms of links cannot be supported without a

redeployment of the link service. Second, the protocol comes in the form of a program library that has to be included in any external application in order to communicate with the link service. This implies that third-party applications have to be rewritten to benefit from the features offered by the link service.

Microcosm supports linking for Microsoft applications and further offers some nice features including generic links or the dynamic linking of documents. Nevertheless, it is not evident how Microcosm could be extended in order to support some of the other important features. It is worth mentioning that a number of open hypermedia systems have been used to enrich the Web with external links by considering the Web as a client for these open hypermedia solutions [4, 6].

The XLink standard supports so-called extended links which can be stored in linkbases and be used to realise bi- and multi-directional links. A link service making use of the XLink model and XPointer expressions can only support the linking across four types (MIME) of XML documents [8]. Nevertheless, most web link services and applications that make use of XLink, such as XLinkProxy [7], solely support HTML documents. On the other hand, RSL is flexible and provides a number of features that XLink lacks such as user rights management, context resolvers and overlapping links. Note that recent Semantic Web technologies and XML promote the concept of linked data [12] on the Web.

A number of systems such as the W3C's Amaya[2] web browser implement the Annotea standard [13]. MADCOW [5] is another web annotation plug-in for Microsoft Internet Explorer that uses a client-server architecture allowing users to store their annotations on dedicated remote servers. MADCOW goes beyond the functionality provided by Annotea-based tools and offers the possibility to annotate richer media types such as images and videos. Nevertheless, most of these tools and standards adopt the simple annotation concepts (e.g. notes or comments) and do not support the creation of hyperlinks between existing content. Even if the linking of existing content is supported and the extensibility is addressed, these solutions are limited to the features offered by XLink.

More recently, various digital libraries management systems (DLMSs) have incorporated interactive annotation features that facilitate discussions among researchers. Whereas in most of the DLMSs the annotation features are offered by built-in components, the Flexible Annotation Service Tool (FAST) [3] has been developed to be a stand-alone annotation tool in order to offer its services to multiple DLMSs. Even though FAST offers simple annotation features rather than cross-document linking and also lacks some of the other features, it is based on an interesting extensible architecture. FAST consists of two main components, the core annotation service and a number of gateways (interfaces). Each gateway is connected to a different DLMS and ensures that the DLMS gets access to the core annotation service offered by FAST. Hence, any DLMS can benefit from FAST's features by developing a new FAST gateway. The flexible integration of different DLMSs with FAST can be considered as third-party integration.

[2] http://www.w3.org/Amaya/

6 Discussion and Future Work

To the best of our knowledge, the presented link service is the first dynamically extensible and customisable link service. The extensible open cross-document link service is a future-proof linking solution for arbitrary document formats and multimedia content types. Its dynamic extensibility further allows third-party developers and end users to support any document format and multimedia content type without the intervention of the link service provider. To address the preferences of the end users, the integration of different document formats and multimedia content types can either happen within the link browser or in their preferred third-party applications. The presented dynamically extensible link service further deals with updates for specific document formats or third-party applications by providing a mechanism for maintaining different versions of the same document format plug-in. Furthermore, the link service represents an ideal platform for investigating innovative forms of cross-media linking.

The manageability and maintainability of links has always been an issue in hypermedia systems. This includes broken links, the consistency of links when the linked documents evolve or the management of link metadata in collaborative environments such as Google Docs. The RSL links used by our link service are bidirectional and therefore the link service is able to solve the issue of broken links by removing any link target from a document when the link source has been deleted from the source document. The management of links in evolving documents is still a hot topic. Even though we are planning to apply more than one mechanism for addressing this problem, we have currently adopted a simple document archiving solution of linked documents which also helps to address the broken link problem in the case of missing documents.

We are currently working on the integration of third-party applications such as Microsoft Word, PowerPoint or Acrobat Reader. Moreover, we are investigating a model for cross-media document formats where content can easily be transcluded from various document formats and multimedia types. In the near future, we plan to evaluate the usability of the presented link service in a user study. Last but not least, we are working on an enhanced desktop environment that exploits the links defined via the link service in combination with some data mining algorithms to enhance the search and retrieval of desktop documents.

7 Conclusion

We have presented a dynamically extensible link service enabling the linking across arbitrary document formats and media types. In contrast to existing link services and link models, our solution supports the integration of new document formats without having to apply any changes to the core of the link service or graphical link browsers and without a redeployment of the link service. Based on the concepts of data plug-ins, visual plug-ins, gateways as well as third-party application add-ins, emerging document formats can either be supported within our link browser or via any third-party application that has been extended to

communicate with a document format-specific gateway. The presented dynamically extensible cross-document link service acts as a research platform for investigating document and link management as well as maintainability in so-called cross-media information spaces. Our solution might further inspire other link service providers to reconsider the dynamic extensibility of their approaches.

References

1. Adobe Portable Document Format Reference, 6th Edition, Version 1.7, Adobe Systems Incorporated, Nov 2006.
2. Standard ECMA-376: Office Open XML File Formats, 3rd Edition, ECMA International, Jun 2011.
3. M. Agosti and N. Ferro. A System Architecture as a Support to a Flexible Annotation Service. In *Proc. of 6th Thematic Workshop of the EU Network of Excellence DELOS*, Cagliari, Italy, Jun 2004.
4. K. M. Anderson, R. N. Taylor, and E. J. Whitehead Jr. Chimera: Hypermedia for Heterogeneous Software Development Environments. *ACM Transactions on Information Systems*, 18:211–245, 2000.
5. 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.
6. N. O. Bouvin. Unifying Strategies for Web Augmentation. In *Proc. of Hypertext 1999*, Darmstadt, Germany, Feb 1999.
7. P. Ciancarini, F. Folli, D. Rossi, and F. Vitali. XLinkProxy: External Linkbases with XLink. In *Proc. of DocEng 2002*, McLean, USA, Nov 2002.
8. S. DeRose, E. Maler, and R. Daniel Jr. XML Pointer Language (XPointer) Version 1.0, Jan 2001.
9. B. J. Haan, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz. IRIS Hypermedia Services. *Comunication of the ACM*, 35(1):36–51, 1992.
10. R. Hall, K. Pauls, S. McCulloch, and D. Savage. *OSGi in Action*. Manning Publications, 2011.
11. W. Hall, H. Davis, and G. Hutchings. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers, 1996.
12. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan and Claypool Publishers, 2011.
13. M.-R. Koivunen. Semantic Authoring by Tagging with Annotea Social Bookmarks and Topics. In *Proc. of SAAW 2006*, Athens, Greece, Nov 2006.
14. T. H. Nelson. *Literary Machines*. Mindful Press, 1982.
15. A. Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proc. of Hypertext 1989*, Pittsburgh, USA, Nov 1989.
16. B. Signer. What is Wrong with Digital Documents? A Conceptual Model for Structural Cross Content Composition and Resuse. In *Proc. of ER 2010*, Vancouver, Canada, Nov 2010.
17. B. Signer and M. C. Norrie. A Framework for Cross-Media Information Mangement. In *Proc. of EuroIMSA 2005*, Grindelwald, Switzerland, Feb 2005.
18. 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.
19. A. A.O. Tayeh and B. Signer. Open Cross-Document Linking and Browsing based on a Visual Plug-in Architecture. In *Proc. of WISE 2014*, Thessaloniki, Greece, Oct 2014.