RDFa as a lightweight metadata interoperability layer between repository software and LOCKSS

Felix Ostrowski

Humboldt-Universität zu Berlin
Institut für Bibliotheks- und Informationswissenschaft
Unter den Linden 6, 10099 Berlin, Germany
felix.ostrowski@hu-berlin.de
http://www.ibi.hu-berlin.de/

Abstract. Semantic Web and Linked Data standards have recently been gaining momentum in the library domain. It seems more likely than not that future systems used in library environments will make increasing use of these standards. This paper outlines possible usage scenarios of semantic metadata in the LOCKSS digital preservation software generally, and the possibilities for metadata interoperability between repository software and the LOCKSS system based on the RDFa standard specifically.

Keywords: digital preservation, repository software, LOCKSS, Semantic Web, metadata, RDFa, interoperability

1 Introduction

Digital libraries and digital archives are closely related. In fact, it is a common misunderstanding that digital library software such as repositories *are* digital archives. If a document is ingested into a repository, it is often thought to be archived. That is not true; it might even be considered a dangerous misconception. At the very bottom of an archival system lies the long-term preservation of bitstreams. Repository software on the other hand are designed for ingest and medium-term access. Of course repositories store bitstreams, too, but this storage usually lives in a typical web server environment. While that environment is hopefully integrated into some sort of backup routine, the storage layer of any common repository software can not be considered to fulfil long term archiving needs in any way.

Commercial long-term preservation systems are built as a whole from the ground up[1] and thus consist of an ingest mechanism that closely resembles that of a repository system and a management layer to control the archive and perform tasks such as format migration. Most interestingly, bitstream preservation is considered a solved problem[12] and is not addressed with the appropriate

[1] Ex Libris' Rosetta[2] and kopal[1] for example focus heavily on ingest, access and management tasks such as format migration.

attention in these systems. Considering the facts that (1) an open source solution for bitstream preservation exists in form of LOCKSS[9], (2) high quality open source repository software to publish documents to the web is available and (3) bitstreams get into a LOCKSS-Network by means of web harvesting, the only missing part to build an OAIS compliant archival system out of these components is a management component[2].

This paper will first briefly discuss the advantages of using semantic metadata in shape of RDF in a LOCKSS environment and then sketch out a possible interoperability of repository software and LOCKSS based on RDFa. It should thus be considered a rather theoretical outline of an archival solution that is based on the loose coupling of existing software solutions by means of semantic metadata. The availability of this data can smooth the way how to add a management component to the environment by facilitating data exchange and integration.

2 LOCKSS and semantic metadata

There are at least three different aspects of how semantic metadata can be of advantage in a LOCKSS environment. Among them are the data management within individual LOCKSS nodes, the integration of metadata present in a LOCKSS network and the interaction of publication platforms such as repositories and the LOCKSS crawler. These three perspectives are briefly discussed next, followed by a more in-depth view on the latter.

Firstly, a generic data model such as RDF allows for a generic database in LOCKSS nodes that can store arbitrary metadata. LOCKSS is at its core a web crawling system that consists of several independent gathering nodes that communicate in order to ensure the integrity of the harvested content. With regards to metadata of any kind, LOCKSS natively used to consider only information at HTTP level, such as content-type and length. It has been extended to support additional metadata by making use of a metadata extraction framework, though. This framework is capable of extracting metadata by analyzing the crawled content and extracting metadata from there. The data is then available within the system and can be further processed. The current version of LOCKSS only includes a relational database with a fixed schema to store descriptive metadata. In context of the LuKII project[5], the need to add a database for technical metadata arose. While the modular architecture of LOCKSS allowed for a relatively easy implementation of an additional metadata manager for technical metadata, it is still tedious to implement such a component for each future data model.

Second off, the integration of metadata accross an entire LOCKSS network or even between LOCKSS nodes and additional services is facilitated. The metadata that is currently being dealt with in the LuKII project[14, p. 4-7], for

[2] The dispute regarding sense and nonsense of prophylactic format migration[13], and other tasks a management component should perform, implies that such a component should not be tied to the other layers too tightly. Discussion of this layer is beyond the scope of this paper.

example, is XML-based, because of which an embedded eXist XML-database has been prototypically added to the LOCKSS daemon and is currently being evaluated. This solves the problem of hard-coded database schemas. Since one goal of the project is testing prophylactic format migration in a LOCKSS network, the metadata from all nodes in the network will have to be integrated at some point to be evaluated by a central preservation management tool. While out of scope of the LuKII project, evaluating the use of a data model that makes it easier to integrate data from multiple sources seems promising for the future. In a distributed environment such as a LOCKSS network, a distributed data model such as RDF appears to be a natural match. On top of that, the management component could apply preservation policies formally expressed in OWL ontologies to identify objects for which actions such as format migration should be taken. Figure 1 outlines this scenario.

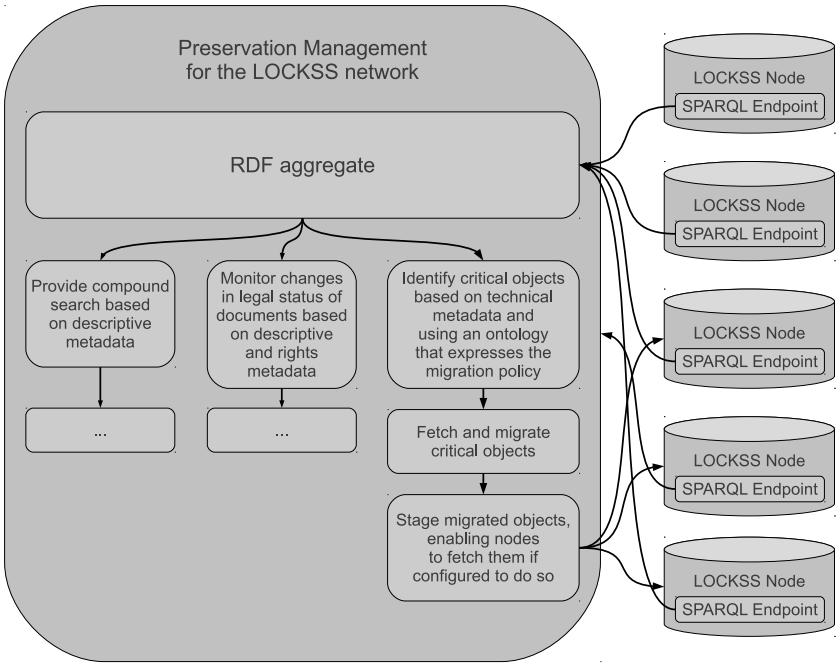

Fig. 1. SPARQL-based aggregation of metadata from LOCKSS nodes

In the third place, the possibilities of semantic metadata are also interesting from the crawling component's point of view. There is an obvious need for a crawler to know what comprises a complex object that should be archived in order to know which links to follow in the potentially infinte web environment. LOCKSS plugins allow to define these rules for different publication platforms. Currently these specific crawling rules of the LOCKSS crawler are defined on

the server side. Vocabularies such as OAI-ORE[6] can be used to more explicitly express these bounds of a complex object (an "article" in the LOCKSS terminology) on the client side, and thus allow for more generic crawling rules in the server side plugin. This would reduce the need for new plugins on the one hand, and allow the specification of an object's bounds at the authorative place - the publication platform - on the other hand. The remainder of this paper will investigate this third usage-scenario for semantic metadata in a LOCKSS environment. Of course, the usage of semantic metadata for interoperability of LOCKSS and repositories is not limited to a single vocabulary. Ontologies such as [4] can be used, for example, to pass legal metadata from the repository to LOCKSS, thereby enabling a more fine-grained notion of rights-management compared to the generic permission statement[10] mechanism currently used by LOCKSS.

3 Exposing semantic metadata from repositories using RDFa

Besides providing the means to ingest and access documents, common repository software at its core also includes possibilities to expose metadata. The metadata capabilities usually consist of a description page that focuses primarily on human readability on the one hand, and machine-readable interfaces such as OAI-PMH on the other hand. With regards to passing content from a repository to a long term preservation system, two problems arise: The lowest common denominator of expressiveness in OAI-PMH is Dublin Core. This does not include structural information of any kind and thus is not detailed enough with regards to long-term archiving needs. Unfortunately, adding support for additional metadata schemes usually entails non-trivial extensions to the software that need to query the database and implement an additional query interface. Besides that, the machine-readable version of the metadata is exposed at a different URL than the human-readable version. This makes the configuration of a web crawler such as the one in LOCKSS more complicated than necessary.

A lightweight solution for both of those problems can be found in RDF in attributes (RDFa)[11]. The essentials in brief are that this Semantic Web standard enables human-readable versions of websites to be enriched in such a way that they can also be interpreted by machines. This means that it is possible to add machine-readable metadata at the template level of repository software. Boulal et al. conclude that the OAI-ORE vocabulary already mentioned above is qualified "as an interoperability layer because it allows describing scholarly work items in a way that is compatible to the web architecture"[3, p. 9]. Providing the necessary resource maps using RDFa is most likely the easiest way to enable existing repository software to expose complex objects in a machine-readable manner, since human-readable splash pages that describe an object already exist in all common repository software. [7] gives an impression of the necessary modifications. A further advantage of using RDFa is that the machine-readable

metadata is available at the same URL as the human-readable version, naturally making it available along with the archived ressource.

4 Processing RDFa in LOCKSS

As mentioned above, LOCKSS is already capable of extracting metadata from the harvested content. The usual extraction procedure is based on extracting the metadata from the human-readable description, which can be error prone. A change in the structure of the page for example can easily result in changes becoming necessary on the plugin-level, even more so changes in the data model that is being used. This is where the advantage of RDFa-enriched HTML-pages becomes evident: the underlying RDF-model stays the same, even when elements are moved around etc., and RDF can be extracted no matter which vocabularies are being used.

With regards to storing metadata extracted from RDFa-enriched web pages, there are several options:

- mapping the metadata to a relational database,
- writing RDF/XML to an XML-database and
- using a triplestore.

While the first possibility enables reuse of the relational database already available in LOCKSS, it would imply unacceptable constraints on the flexibility of the system. Changes in the ontologies used to describe the content in the repository would necessarily imply changes to the database schema. Using an XML-store would be a solution for this, but limits the query possibilities to languages focusing on syntax rather than semantics, such as XPath or XQuery. The usage of a triplestore, ideally with an enabled reasoning component, is the most natural solution for RDF data and provides a powerful SPARQL interface.

The components that have been identified as necessary to enable the storage of RDFa metadata exposed by repository software in LOCKSS using a triplestore are:

- RDFa-enabled repository software that includes semantic markup for structural information,
- a LOCKSS plugin that is able to crawl according to the resource maps exposed by the repository,
- an article iterator that is able to make complex objects internally available,
- an article metadata extractor to extract RDF data about the complex object from the RDFa in the resource map,
- a file metadata extractor to extract and merge RDF data about individual bitstreams from the RDFa in the resource map and optionally in the individual files of an article,
- a metadata manager to add, update and delete RDF data from an embedded triplestore and

– a SPARQL endpoint to expose the metadata.

Figure 2 shows how these elements interact within a LOCKSS node and inbetween the node and a repository that provides the content that is to be archived. For more information on the concepts of plugins, article iterators, metadata extractors and metadata managers in LOCKSS components see [8]. While the work on a MetadataManager for RDF data has already begun as a side project of the author, the other parts are still missing. Once they are in place, the system will need to undergo extensive testing, especially with regards to the performance of the triplestore in a real-word environment.

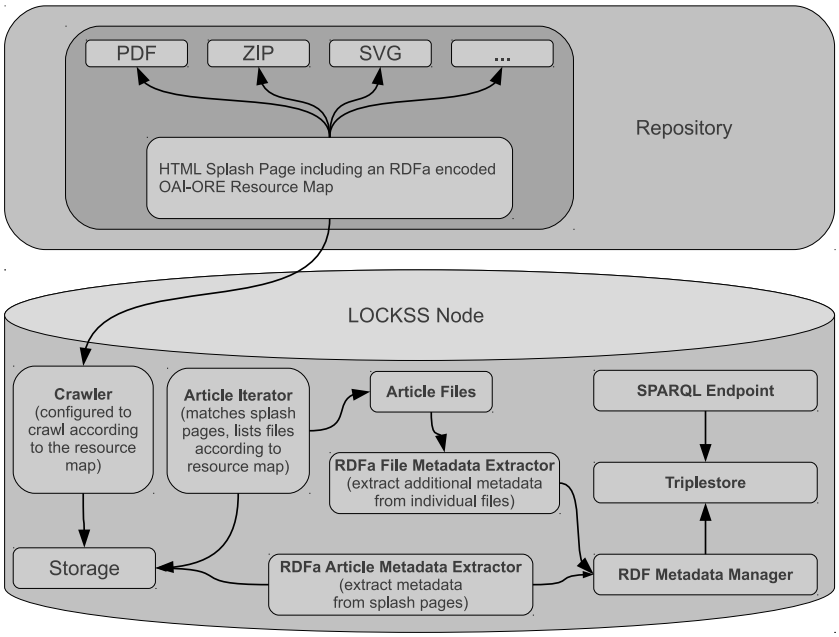

Fig. 2. Interaction of components necessary to store RDF data extracted from RDFa-enriched HTML-pages

5 Conclusion

This has been a rough and purely technical view on a modular digital preservation system that makes use of Semantic Web standards. Question such as "Which metadata belongs into a digital archive?", "Which part of the system is responsible to generate technical metadata?" and "Does descriptive metadata belong into the preservation layer?" remain open. The model sketched out above theoretically makes it possible to add – along with the content it describes – arbitrary, but highly expressive metadata to a modular long-term archiving system.

The possibility to access that metadata through an HTTP-SPARQL-interface provides the means to add a management layer to the system without tying it in too tightly. With the advent of Semantic Web technologies and standards in the library domain, further investigations in this direction seem promising.

References

1. About kopal, http://kopal.langzeitarchivierung.de/index.php.en
2. A New Way of Preserving Cultural Heritage and Cumulative Knowledge, http://www.exlibrisgroup.com/category/RosettaOverview
3. Boulal, Anouar et al.: Report on Enhancing Interoperability between existing Open Access Publication Infrastructures. Draft available at http://www.eco4r.org/downloads/eco4r_report_compoundobjects_draft.pdf (2010)
4. CASPAR Rights Ontology, http://www.casparpreserves.eu/publications/ontologies/RightsOntology.html
5. DFG-Projekt: LuKII (LOCKSS und KOPAL Infrastruktur und Interoperabilität), http://www.lukii.hu-berlin.de/
6. Lagoze, Carl et al.: ORE User Guide - Primer, http://www.openarchives.org/ore/1.0/primer (2008)
7. Lagoze, Carl et al.: ORE User Guide - Resource Map Implementation in RDFa, http://www.openarchives.org/ore/1.0/rdfa (2008)
8. LOCKSS API documentation, http://www.lockss.org/lockssdoc/gamma/daemon/index.html
9. Lots of Copies Keep Stuff Safe), http://lockss.stanford.edu/lockss/Home
10. Making Your Titles LOCKSS Compliant, http://www.lockss.org/lockss/Making_Your_Titles_LOCKSS_Compliant#Permission_to_the_LOCKSS_Software
11. RDFa Primer. Bridging the Human and Data Webs, http://www.w3.org/TR/xhtml-rdfa-primer/ (2008)
12. Rosenthal, David S.H.: Bit Preservation: A Solved Problem? In: International Journal of Digital Curation, vol. 1, no.5. http://www.ijdc.net/index.php/ijdc/article/viewFile/151/224 (2010)
13. Rosenthal, David S.H.: The Half-Life of Digital Formats, http://blog.dshr.org/2010/11/half-life-of-digital-formats.html (2010)
14. Steinke, Tobias: Universal Object Format. An archiving and exchange format for digital objects, http://kopal.langzeitarchivierung.de/downloads/kopal_Universal_Object_Format.pdf (2006)