loomp – mashup authoring and semantic annotation using linked data

Annika Hinze a Markus Luczak-Rösch b Ralf Heese c Hannes Mühleisen d Adrian Paschke e
a University of Waikato, New Zealand, hinze@waikato.ac.nz
b Victoria University of Wellington, New Zealand, markus.luczak-roesch@vuw.ac.nz
c Humboldt University Berlin, Germany, rheese@informatik.hu-berlin.de
d Centrum Wiskunde & Informatica, Netherlands, hannes@cwi.nl
e Freie Universität Berlin, Germany, paschke@inf.fu-berlin.de

Abstract. Legacy full-text corpora and low precision of automatic annotation have long hindered widespread utilization of semantically-rich content. Best annotation quality on specialised collections can be achieved through manual annotation; most manual approaches assume that semantic annotations will be created by domain experts. However, the poor usability of many tools is a challenge to end-users who do not have extended knowledge of semantic techniques. For the semantic content authoring tool loomp, we followed a user-centred design approach to implement mashup and manual annotation of textual resources. loomp was developed in close collaboration with knowledge workers and domain experts from public service institutions. This article provides technical details of loomp’s design, architecture and data model based on requirements derived from real-world journalism use cases.

Keywords: semantic annotation, linked data, text mashup, semantic web, content authoring, manual annotation

1. Introduction

For a long time, semantic technologies were only available to a small community with little impact on the general public, although the Semantic Web was envisioned to be the “Future of the Web” [1,2,3]. The legacy of existing document archives and other full-text corpora without semantic annotations is one of the major hurdles for a broader acceptance and availability of semantic techniques, such as semantic search and authoring. Another hurdle is the lack of re-usability of text snippets that have been digitally curated and enriched with semantic knowledge.

To make existing documents accessible via semantic search, many (predominantly automatic) semantic annotation tools were proposed to enrich large legacy corpora (e.g., [4][5][6][7][8]). However, most of these tools follow indiscriminate brute-force methods that lead to low-precision markup [9]. The reasons are errors in disambiguation (i.e. linking wrong concepts) and over-annotation (i.e. linking correct but irrelevant concepts). The lack of sufficiently-developed ontologies further lowered recall, because relevant concepts were undefined.

An alternative approach has been followed in the development of manual annotation tools. By now, a range of manual or semi-automatic tools have been proposed (e.g., [10][11][12]), most of which assume that semantic annotations will be created by domain experts. However, we note that these tools were rarely developed in collaboration with “real” domain experts or even tested in a realistic setting [13]. For most tools, no end-user testing has been executed or only with unsuitable test subjects such as Computer Science students. Finally, semantic annotations can support the re-use of existing text fragments in new documents, e.g., via semantic-based filtering and access. Semantic enrichment of those new documents should be integrated into the authoring process.

In this article, we describe our manual annotation and authoring tool loomp, which is available as open source [14]. Different to most other tools, loomp fol-
owed a user-centred design process and has been developed in close collaboration with knowledge workers and domain experts from public service institutions. Its annotation capabilities have been extensively tested for its usability and suitability for technical non-experts.1

This article provides technical details of the conceptual approach underlying loomp and an overview of the core implementation and data model principles (for implementation details on early prototypes, see [15,16, 17,18]). We show how the user-centred design process shaped both the system’s design and implementation and briefly summarise the results of the end-user evaluations that were performed on loomp (for more details on the user evaluations, see [13,19,20]). Finally, we present a carefully crafted survey of tools that represent different generations of semantic annotation research and practice, from the early days of the Semantic Web vision up to the most recent developments on social Linked Data. In the light of this tool survey we critically re-evaluate the loomp system capabilities.

Throughout the project, our research has been grounded in focused end-user scenarios and rigorous requirements analysis. This article follows a similar structure: Section 2 gives an overview of the usage scenarios developed for loomp and the requirements derived from these scenarios. Section 3 outlines the end-user interaction for loomp, which was designed with focus on non-expert users, and describes how the end-user interfaces address our non-functional requirements for loomp. Section 4 introduces the anatomy of the loomp system with reference to the functional requirements. Screenshots of the loomp live prototype system are shown in Section 5. Section 6 discusses both the loomp tool and related approaches in the light of our user requirements. We summarise the contributions of this article and discuss next steps in this research in Section 7.

2. Scenario-based Requirements Analysis

Design considerations for any interactive tool or system need to be grounded in pertinent usage scenarios; this is especially true for systems that target non-expert end-users [21]. Here we briefly sketch one of loomp’s usage scenarios (in Section 2.1) and then outline the requirements derived from the scenario (in Section 2.2). Our scenario is taken from the domain of journalism and is representative for content-intensive work in a heterogeneous environment. It was developed based on interviews with journalists and further shaped by the results of a Delphi-study with 200 expert knowledge workers on the challenges of publishing in a digital economy [22].

2.1. Usage Scenario

In our interviews with journalists and publishing houses, we learned that journalists research specific topics on demand and utilise a mix of digital and print sources and human informants. At the time, notes and insights were recorded predominantly using paper and pencil in addition to digital tools. Finished articles were both created and transferred using digital communication media.

A journalist using loomp as a personal information management (PIM) and digital curation tool would have access to semantically enriched notes, interview logs, references, and articles. Many of these resources may have been created in the course of their work on specific topics. Annotation is not regarded the main task of the journalist, rather it is one of the steps in the creation of output artifacts (i.e. the written articles). Here we briefly outline the steps a journalist would take in the creation and annotation of text elements using semantic annotations while authoring, in contrast to established methods.

Step 1 (Integrated annotation and authoring): Instead of maintaining a traditional (card or digital) index, the journalists mark up persons of interest, locations, and events electronically to facilitate targeted search for future articles. That means, a journalist creates semantic annotations on personal notes and existing digital sources on-the-fly while researching and writing. The semantic markup to be created is predominantly used for later identification of entities (e.g., persons or places that occur in an article).

Step 2 (Search and re-use of text fragments): Journalists used to retrieve their notes and annotations from the index to facilitate their writing. In loomp, the journalists can execute targeted semantic searches on their data, for example, all articles about “Frankfurt” in county Brandenburg, not Frankfurt/Main. Moreover, these semantic searches would not only access their own but also externally-linked data content.

1We consider users that do not have extensive knowledge of Semantic Web technology and semantic concepts to be “technical non-experts.” These users may, however, be domain experts.
Step 3 (Publication of annotation): Publishing houses would use loomp as corporate information management (CIM) system for corporate smart content [23], offering the value-added services of targeted search based on expert semantic markup for their customers without additional effort. The journalists providing the semantic markup can be considered domain experts, but they are not technical experts on complex semantic markup.

Although our scenario also captures the search process, this is merely used as motivation for the requirements on the annotation process. Techniques of semantic search will not be discussed in the article as it has been implemented by many systems and is not the main focus of this work.

2.2. Results of requirement analysis

Through the loomp scenario, of which a simplified form has been outlined above, we explored the authoring and annotation tasks the users may perform with loomp. We derived the following functional and non-functional requirements, which are briefly summarised here.

2.2.1. Functional requirements

These requirements describe the intended behavior of the loomp system relating to specific functions of its components.

R1 – Mashup Authoring: Fragments are created or re-used to produce a text that is a mashup of existing and new parts. Retrieval and re-use of semantically-related text fragments (see Step 2 in Sect. 2.1) is a key driver of the requirement that the content and the added semantic markup needs to be stored and made available for semantic search.

R2 – Manual annotation: The system needs to provide an interface for manual creation of semantic markup (see Step 1 in Sect. 2.1). While the users are creating or reading texts, the system provides interface elements for the creation, display and deletion of semantic markup on text elements. Based on our conversations with journalists, the desired semantic markup to be created is the identification of named entities (i.e., links between text literals and ontology concepts).

R3 – Interoperability: To support the use of external data and the exchange of annotations between journalists (see Steps 2 and 3), the data representation needs to conform to open standards. The underlying schema needs to source from widely accepted vocabularies and existing linked data sets.

R4 – Sharing of annotations and fragments: Because the data analysis required for data journalism can often no longer be done by a single journalist, semantic annotations need to be shareable between annotators and other users (e.g., as in Steps 2 and 3).

R5 – Semantic Search and Navigation: Retrieval and re-use of semantically-related text fragments (see Steps 2 and 3) is a key driver of the requirement that the content (from R1) and the added semantic markup (from R2) needs to be stored and made available for semantic-based access, e.g., via search and browsing. Navigation may be done along semantic linkages, i.e., browse fragments by semantics in a text, suitable for users such as journalists and other readers.

R6 – Extensible Vocabulary: Because semantic annotations may be created in different contexts dealing with new types of data (e.g., collaborative investigations and article series as seen for the initial WikiLeaks), journalists may wish to create their own vocabularies best suitable for their purpose. Vocabularies may be pre-existing but not yet coded (e.g. specialised research fields) or they may be emergent as the data is processed.

R7 – Distributed information sources: Because systems need to support the annotation of documents, images, and videos from distributed large data sources (e.g., open government, digital libraries and legacy corpora of texts) as well as private notes and data aggregations as described above, an integrative approach is required that is agnostic to the actual location of the source data.

R8 – Provenance of annotation and fragments: From the need to share annotations follows the requirement to identify who annotated which data when. Providing context of the act of annotating (i.e., why or for what purpose) would allow further filtering of annotations.

The implementation of loomp (design and architecture) and details on how the eight technical requirements are addressed, is detailed in Section 4.

2.2.2. Non-functional requirements

These requirements describe the desired qualities and performance characteristics of loomp. They are informed by both the user scenario and the functional re-
requirements, as well as existing research into semantic web usability.

**R9 – Focus on user task:** The users are assumed to be technical non-experts that are primarily engaged in a task other than semantic annotation (e.g., writing articles). As suggested by Handschuh and Staab [24], the annotation elements need to seamlessly integrate into the text editor without distraction from the primary task.

**R10 – Simplified vocabularies:** Users will often require for their authoring and annotation (i.e., working on a specific article) only a small part of a large ontology with multiple hierarchies. It is also known that complex semantic category structures are disadvantageous for high-quality annotations [25][26]. For simplicity, loomp should therefore only display selected concepts.

**R11 – Established interaction patterns:** loomp targets non-expert end-users for tasks that ideally require technical expertise. The interface therefore needs to hide this complexity while offering intuitive interaction elements for semantic annotation that mirror known elements, e.g., from text editors. Journalists need to be encouraged to engage with the semantic annotation, without having to spend a long time learning a new task. Using familiar interaction paradigms in semantic web applications has been previously recommended [27]. Information tools are also known to be more successful when extending existing functionality rather than introducing entirely new ways of doing things [28].

**R12 – Contextual semantic:** Many end-users will be familiar with simple web terms such as URLs (or “web address”), but not with more complex concepts such as namespaces. The annotation interface needs to provide access to vocabularies while hiding the technical nomenclature.

One of the main challenges of the loomp design was to adequately address the non-functional requirements that embody the design focus on technical non-experts. We therefore first give an overview of the interface and interaction design of loomp in Section 3, before explaining in detail the functional and technical structure as well as the data model of the system in Section 4. While the non-functional requirements were defined in relation to the functional requirements, the technical implementation and data model is significantly influenced by the desired interaction design.

### 3. Interaction Design for loomp

The end-user interaction for loomp was designed with technical non-experts in mind. The interface for authoring documents composed of text fragments is called Fragment Mashup Editor (FME). The fragment-level interface for authoring and annotating each text fragment is called One Click Annotator (OCA).

Here we describe the interaction design of both FME and OCA by means of a walk-through. As we describe our design decisions, we relate each to the list of non-functional requirements (R9 to R12) that are defined in Section 2.2.2. The details of how the functional requirements were addressed in the system implementation is presented in Section 4 and discussed in relation to related approaches in Section 6.

#### 3.1. Fragment Mash-up via FME

Figure 1 shows our original interface concept for the Fragment Mashup Editor (FME), which provides the outer framework for the authoring and annotation process. The left window, labelled (I), shows the concept of the mashup editor component indicating the order of fragments within the currently-edited mashup.

In our example, the journalist is creating an article about “Life in Berlin/Brandenburg”. Three fragments are compiled into the document mashup: an introduction, some fragment about Brandenburg and a re-used fragment about Frankfurt (Oder). Once the fragments have been assembled within the mashup, their origin (new or re-used) can no longer be deduced from the interface.

The right window, labelled (II), is the management component that supports creation of new fragments as well as access to existing fragments. Access to fragments would be gained via searching and browsing functions; these rely on the semantic annotation of existing fragments. For example, the text fragment about
Frankfurt (Oder) is known as one of the most important transit points from Germany to Poland. Frankfurt is located about 90 kilometres east from the capital of Germany, Berlin. People confuse Frankfurt/Main and Frankfurt (Oder) but they are a long way away from each other, about 500 km. In the town live approximately 63,000 inhabitants. It is embedded between woods and lakes. In downtown we have a wonderful renewed promenade from where you can see the river Oder. When you are on the west bank you can see the town Slubice (Poland).

3.2. Semantic annotation via OCA

Figure 2 shows our original interface concept for the One Click Annotator (OCA); the left window is the main screen (with elements (A)–(D)), while the right-hand window describes a pop-up element, labelled (E).

The example shows the fragment about Frankfurt (Oder) in the central authoring pane (see (A)), which is currently edited by a journalist. The user selected a vocabulary (here: Geography, see (B)) that provides vocabulary terms appropriate for the text and planned annotations (see (C)). The right-hand pane shows previously-identified semantic concepts (see (D)), which the user might wish to re-use for ease of annotation.

Assigning an annotation to a text literal is done in two steps. Firstly a text literal is marked in the text (e.g., "Frankfurt"). Then a concept from the offered vocabulary is selected (here: Frankfurt (Oder)). A third step may be required for creating new concepts that are not contained in DBpedia. The annotated literals are marked green in the authoring window (C) in Figure 2. The user has also selected the concept Frankfurt (Oder) in the selection pane (D). All literals that have been previously annotated with this concept are highlighted in red.

3.3. Interaction vs non-functional Requirements

We now briefly discuss how the non-functional requirements R9 to R12 shaped the design process. Because these four requirements focus on the main activities of writing and annotating, they were predominantly addressed through the OCA design (while the FME supports search and mashup).

R9 determined the overall layout of the OCA by requiring it to be focused on the user task of writing or reading of a text. As a consequence, we integrated the loomp OCA toolbar for creating semantic annotations seamlessly into the text editor, so that the user can add annotations without being distracted from their primary task. R10 requested the display of simplified vocabularies that hide complex technical nomenclature. We therefore decided to only offer an appropriate subset of categories, and provide support in choosing the appropriate annotation. (B) and (C) illustrate the design: the user is shown a selection of concepts that are pertinent to the article they are reading. R11 required the interface to use established interaction patterns. We adopted the well-known interaction elements of text highlighting and formatting (similar to their use in MS Word®). Figure 2 (A) shows the highlighting in different colours. We explored a number of markup highlighting options [19]. The effectiveness of our strat-
egy has been tested in our evaluation (reported in the next subsection). R12 identified the need for contextual semantic. To hide the complexity of name spaces, we needed to provide other means to clearly identify each concept. Humans use labels to refer to entities (e.g., “baker”) and disambiguate meaning by the textual context (e.g., as reference to the person Baker or the profession baker). The OCA aims to bridge this gap between objective knowledge (as encoded in the RDF data model) and subjective knowledge of human cognition by presenting labels and contextual information for identification of semantic identity, see Figure 2 E.

3.4. Effectiveness of OCA design

As loomp aims to enable technical non-experts to semantically annotate textual content, our efforts for analysing the tool’s usability centered around this feature. The effectiveness of loomp’s OCA interaction design was extensively tested in a series of user studies, which were the first of their kind. End-user evaluations of annotation tools are very rare and typically do not seek feedback on interaction issues or participants’ mental models of the system interaction (e.g. [29][30]). In contrast, our studies focused on the challenges of the annotation process. We performed usability tests and interviews with real users (not student stand-ins), with the aim to (1) evaluate the suitability of the tool for non-experts, and (2) explore how non-expert users experience and apply the concept of annotating texts. Note that our target group of users are domain experts (e.g., journalists or scholars) but non-experts with respect to semantic web concepts.

We here briefly summarise the lessons learned; with details in [13,19,20].

Suitability for non-experts. All participants were able to select the correct concept from the list if the entries contained enough information to identify the entity they had in mind. Problems arose when participants were unable to disambiguate recommended entities. If in doubt about literals, participants created new concept IDs, which highlights the importance of assisting annotators in applying the conceptual model of semantic identity and its difference to tagging.

From our studies, we thus identified three groups of participants: acceptable annotators, annotators with room for improvement, and failed annotators. Members of the first two groups were found to be knowledge workers (such as our journalists from the scenario) or had technical knowledge (though this alone did not guarantee quality annotations). Some participants expected the system to have in-depth ‘understanding’ of complex relationships. Similar observations have been made w.r.t novice computer users [31] and might indicate the novelty of semantic content in this field. Similarly, the task of providing conceptual, semantic knowledge seemed so foreign to some participants that they switched to consuming the provided information.

Experience for non-experts. Some participants found it difficult to decide which concepts to select and when to stop. Here, a reduction in size of available vocabularies and annotations could help keeping annotators focused on identifying (named) entities and increase the quality of annotations. The dynamic history of annotations (cf. Figure 2 D) in the annotation sidebar had mixed results and was therefore implemented differently to the original design, separating the faceted viewing from the history element. The prototype purposefully resembled the look and feel of MS Word to allow easy recognition of known interaction patterns. However, the participants’ mental model had strong references to Web-based interactions. A number of participants wanted to create nested annotations, which are currently not supported. We explored nested annotations and found their visualization to be challenging due to a conflict between precise visual representation and the readability of the text [19].

In summary, we found loomp to be suitable for non-experts, especially those familiar with knowledge work (whom we call task-experts). The interaction experience for non-experts was found to be sufficient but improvements are possible.

4. Anatomy of the loomp System

We now introduce loomp’s architecture and data model in detail. The loomp system was designed as a web application2 and its management of data is inspired by hypertext principles. In loomp, annotated texts may be whole documents or text elements (so-called fragments) that are combined to form documents (so-called mashups). The loomp data model is described in Section 4.2.

2Its first instantiation used PHP and the model-view-controller pattern, whereas the second instantiation used a service-oriented paradigm.
4.1. Loomp architecture and data flow

Figure 3 shows the architecture and dataflow of loomp, which we describe following the circled numbers. The loomp dataflow begins when a user creates a text fragment, which may become part of a text mashup (see 1a). A given (or newly written) text may then be annotated via the OCA (see 1b in Figure 3) or it may be used in a mashup to create further texts.

We first describe the authoring and annotation data flow for a single text fragment and then describe the creation of more complex mashups below.

A user creates initially an empty fragment (via the FME 1a) into which they then insert text using the OCA (see 1b), which may additionally be annotated (the concept of this user interaction was described in Section 3). On completion of authoring and/or annotation, the OCA sends the fragment’s annotated text (mix of HTML and RDFa\(^3\)) via the web application interface to the Annotation extractor, see 2. This component identifies the RDFa annotations and transfers them into the RDF Store (via an RDF API to a MySQL database), see 3. The document text with embedded annotations is also stored in this database for later retrieval or further annotation in the OCA, see 4. The vocabularies (encoded in RDF) that are presented to the user in the OCA for use in annotations are pre-installed in the database and loaded into the OCA on demand (also 4).

We now describe how the annotated text fragment can be used in a mashup of fragments (see 1a). All fragments in a mashup are displayed and managed via the fragment mashup editor FME. Any fragment for which the text is to be changed, can be edited using the OCA within the mashup editor. Additional fragments may be identified via searching and browsing, and inserted into the mashup. For this, loomp supports faceted viewing of annotated texts (i.e., highlighting of selected types of annotations) and faceted browsing of text collections (i.e., filtering/linking of documents in a collection according to annotated concepts), see 5.

Both faceted browsing and faceted viewing exploit the semantic annotations of the content, querying the RDF Store via SPARQL [32] (see 6). The authoring functionality also relies on the full-text index, see 7, for retrieval of documents and fragments for mashup. The index is implemented using Lucene.\(^4\) The structure of a mashup is encoded using RDFa: the FME transfers

[^3]: http://www.w3.org/TR/rdfa-syntax/
[^4]: http://lucene.apache.org/
html and RDFa information to the Annotation Extractor \(^2\), which is then also stored in the RDF Store \(^3\).

Interoperability is supported at both the interface level as well as the data level, for external applications and linked data clients, respectively (see \(^8\)). External web applications may gain read-and-write access to annotated content depending on access permissions as defined by the document owner. While these applications have access via a web-based API (part of the loomp web application interface shown centrally in Figure 3), the Linked Data clients have access to loomp data via a Linked Data Server (i.e., the RDF Store via a Linked Data interface, see \(^9\)). Both the RDF API and Linked Data interface components are realized using the RAP Pubby library for PHP [33]. Not shown in Figure 3 are components for user authentication, system administration and security.

Based on this outline of the architecture and dataflow, we now briefly summarise how loomp implements requirements R1 to R5. R1 is addressed at end-user level through the authoring and annotation tools (i.e., editor functions in the FME and OCA) and the authoring tools. R2 is addressed by the design of the OCA annotation functions as well as loomp’s server side components. R3 is addressed through the interoperability interfaces and by using W3C standardised data formats (e.g., RDFa, RDF, RDFS, SPARQL, and Linked Data principles). Sharing of fragments and annotations (addressing R4) is supported through fragment re-use via search and filtering (RDFa structure). Any annotations contained in fragments are automatically shared as well. Additionally, loomp allows collaborative editing of mashups through remote applications if the owner of the instance shares a dedicated manipulation API key for a particular mashup resource. R5 is addressed through the authoring tools, the RDF Store and the text and annotation index.

4.2. Domain ontology and data model

The loomp data model describes the relationships between documents (i.e., mashups), text fragments, literals occurring in the text, and semantic annotations (i.e., concepts in external ontologies). It uses the loomp domain ontology as the formal schema for content markup in loomp.

4.2.1. Loomp Domain Ontology

Documents in loomp may have been authored within loomp (i.e., they are native loomp documents) or imported from external sources. Figure 4 illustrates the main entities and relationships in the loomp Domain Ontology. Note that the ontology does not encode the textual content of fragments but rather the relationship between annotated text elements and their semantic annotations (i.e., named entities) in fragments.

![Figure 4. Elements in the loomp Domain Ontology](image)

Each loomp document is a mashup of document fragments. For a document \(D\) loomp assigns a document identifier \(DocID\) and stores a sequence of fragment identifiers \([FragID_1, \ldots, FragID_n]\). Fragments have user-defined granularity and may be chapters, sections, paragraphs, or sentences. The explicit order of fragments (i.e., defined as sequence) within a mashup ensures reproducible document structures. A newly imported document is converted into a loomp document by assigning a new \(DocID\) and transforming its content into a single fragment, whose \(FragID\) is then linked to the new document’s \(DocID\). Fragments may be re-used in various document mashups, ensuring that loomp addresses requirement R1.

Semantic annotations in loomp identify named entities for selected text elements (addressing requirement R2).\(^5\) For example, the text literal “Frankfurt” from our journalism example (shown in the second line of the fragment being annotated in Figure 2 (A)) is being annotated with the named entity Frankfurter Oder. loomp uses DBpedia [34] as a pre-defined set of named entities that may be selected for annotations. That is, the named entity Frankfurt (Oder) is sourced from DBpedia with URI http://dbpedia.org/resource/Frankfurt_(Oder). For interoper-

\(^5\) loomp also supports annotations of relationships between two annotated entities. Because this feature is only available to advanced users and technical experts, it is not described in detail here.
ability (requirement R3), the link between a text literal and the assigned named entity (i.e., the semantic annotation) is encoded in loomp through a property concept. These property concepts are encoded in vocabularies (e.g., Geography in Figure 2(C) and its related terms in (B)) that link simple terms to property concepts defined in external ontologies, such as RDF or FOAF. loomp additionally supports user-defined ontologies, e.g., for corporate vocabularies. In our journalism example, the annotation of "Frankfurt" with the entity Frankfurt (Oder) is encoded as the property rdf:label with the domain http://dbpedia.org/resource/City.

We assume that end-users who are technical non-experts would be overwhelmed if exposed to the full scale and intricacies of these (external) ontological schemata. loomp therefore encodes a translation between selected concepts of an (external) ontology and a local loomp vocabulary, for each ontology used. For example, instead of presenting the aforementioned property with both URI and domain details, the user is presented with the vocabulary term "City" (as seen in Figure 2(B)). A vocabulary therefore consists of a list of terms; each term \( T \) defines a literal \( \text{TermLabel}_T \), which is identified by the term ID \( \text{TermID}_T \) and links to an external ConceptID. The vocabulary is a concept solely created for interface and interaction purposes. loomp also aims to support end-users in the selection of DBpedia entities through an interactive filtering process (described in Section 3).

loomp has a number of vocabularies installed by default; others can be loaded as needed. Figure 6 shows a graphic representation of selected elements of the default multi-lingual vocabulary for personal information. Depending on the language selected for loomp, either the German or English labels are shown in the vocabulary toolbar. The multi-linguality of a loomp vocabulary is relevant for supporting semantic search in different languages independently of the languages used in the fragments and mashups themselves. This enables easy internationalisation of the loomp interface.

Finally, the loomp domain ontology is encoded as RDFS vocabulary.\(^6\) For clarity, we refrain from showing a complete serialisation of the ontology, but rather list the main classes and properties in the table shown in Figure 5. Identifiers of documents (DocID), fragments (FragID), and terms (TermID) are encoded using URIs in the namespace of the loomp data ontology; TermLabels are RDF literals and the connection between TermID and TermLabel is an RDFS label. The ConceptIDs of external concepts are encoded using

\(^6\)http://www.w3.org/TR/rdf-schema/
URIs in the namespace of the respective external ontology; the named entities are encoded as URIs using the DBpedia namespace.

4.2.2. Loomp data model

The loomp data model contains user documents and annotations, the relationship between which is encoded as RDF statements using concepts from the loomp domain ontology. Figure 7 shows a section of an example data model instantiation. This example implements and extends the scenario introduced in our design concept discussion (see Section 3).

Two mashups of fragments are depicted as stylized green documents, see 1 and 2 in Figure 7. The first mashup contains one fragment only and the second mashup contains three fragments; one of the fragments is used in both mashups (indicated through the bold links to fragment1. In the RDF Store, these fragments and mashups are typed as loomp:Fragment and loomp:Mashup, respectively (cf. Figure 5).

The loomp data elements use the prefix data in Figure 7. The content model expresses the structure of both the mashup content (see 3) and the in-text annotations (see 4). Mashups are typed as rdf:Seq, which are shown in Figure 7 via links. For example, the shared fragment1 is linked via rdf:_1 to mashup1 and via rdf:_3 to mashup2. Fragments can be re-used in multiple mashups.

The documents’ full-texts are kept in the RDF Store as XHTML serialisations with embedded annotations using RDFa markup. Each fragment links to its associated literal of text with embedded annotations via the loomp:hasRDFa property (see Figure 5 and Figure 7, part 3). This encoding enables the display of the documents’ full-texts to the user. All fragments additionally link to their annotations via a loomp:contains property. An example is shown in Figure 7 for annotations on fragment1. This second encoding enables retrieval, highlighting and exchange of annotations.

The loomp annotation data links to external ontologies for property concepts (see 5) and to DBpedia...
for named entities (see [6]). The vocabulary used to select the appropriate concepts and properties is only partially shown (see [7]).

**Example fragment encoding**

In Figures 8 and 9, we show the raw RDFa data and the RDF graph for the mashup containing the original fragment about Frankfurt (Oder), i.e., mashup2, respectively. Figure 8 shows an excerpt of the RDFa raw data stored for fragment1, linked as rdf: _3 (export from loomp Linked Data Interface, see Figure 3).

The RDF graph of mashup2 is shown with one fragment only (for readability reasons) in Figure 9. Encoded green are those resources and properties that are internal to loomp; encoded in orange are those that are external to loomp.

5. The **loomp** tool: Screenshots

Screenshots of a loomp live installation are shown in Figures 10 to 13.7 Figure 10 shows the loomp dashboard after the user created two articles. The first article is a single-fragment mashup about transport points in Frankfurt (Oder), the second one is a mashup consisting of three fragments, reusing the Frankfurt (Oder) fragment and adding two new ones. We now show in screenshots the steps for creating and semantically annotating these two articles.
Article with single (new) fragment

Creating a new mashup from the dashboard requires opening the editor (see left-hand box in Figure 10), which appears as a new tab (see open editor tab at top of Figure 11). In Figure 11, we see the FME after the user created the article (i.e., the document mashup), gave it the title “Frankfurt (Oder)” and created a single fragment titled “Transport Points”. We concentrate now on the OCA interface, which is used to create text and semantic annotations for the fragment.

The lower half of Figure 11 shows the OCA editor with the authoring pane (A) and the annotation elements (B) and (C). The user is in the process of authoring and semantically annotating the fragment. Fig-

Fig. 11. loomp authoring with annotation elements (screenshot)

ure 12 shows the pop-up window for detailed selection of semantic concepts (E). The originally-planned pane for previously-used annotations has not been included in this version of loomp; faceted viewing of annotations is instead implemented separately and shown in Figure 13, referred to as Alt-(D). This function can be reached from the dashboard when selecting view for an article. In the example shown, three annotation concepts have been selected for highlighting: rivers (green), cities (red) and countries (blue).

Article with re-used fragment

In a second article about “Life in Berlin/Brandenburg”, the journalist decides to re-use the fragment about Frankfurt (Oder) that was created before. Figure 14 shows a screenshot of the loomp fragment mashup editor, in which the journalist starts writing the article. The first two fragments (i.e., the introduction and a fragment about Brandenburg) can be seen. Here the OCA view is hidden for both fragments as they are not currently in editing mode.

Figure 15, the user is now searching for an existing fragment about Frankfurt (Oder), see right-hand column. In a drag-and-drop interaction, the identified fragment can then be inserted in the document mashup.

The final document mashup containing all three fragments can be exported via the dashboard (shown in Figure 10) as PDF document. The resulting short document is shown in Figure 16. Note that the third fragment still carries its original formatting from the first article. Adjusting the formatting or any textual changes for this fragment would result in a new copy of the fragment being created (instead of directly linking to the original fragment).

Fig. 12. loomp resource selector (screenshot)

Fig. 13. loomp faceted viewing of text with highlighted annotation types (alternative implementation of design of (D)) (screenshot)
6. Discussion and Related Work

In this section, we first briefly summarise how the functional requirements outlined in Section 2.2 have been implemented in loomp. We compare the loomp tool and its requirements with related approaches and synthesise our observations in Table 17. Finally, we position loomp as a hypertext tool in Section 6.1.2.

6.1. Loomp requirements review

All eight functional requirements were addressed in loomp’s conceptual architecture. Requirements R1 to R5 were fully implemented (for details see Section 4.1). Requirements R6 and R7 were implemented partially. While extensible vocabularies are fully supported in both the loomp design and implementation (R6), no end-user interface is available yet. Distributed sources (R7) are supported in the data model; the current prototype implementation can only handle document fragments that have been imported into loomp. The provenance requirement (R8) is addressed conceptually and could be incorporated into the data model. This aspect needs further work (discussed in Section 7). The table shown in Figure 17 encodes an overview of how the requirements were addressed.

loomp inherently addresses the four non-functional requirements by design. We refrain here from casting judgement on other system’s non-functional aspects as these would require a more in-depth analysis with user studies that go much further than is relevant for this article.

6.1.1. Related tools for semantic text annotation

With the uptake of the Semantic Web, an increasing demand evolved to develop tools to create semantic data [36,37]. As discussed in the introduction, there are tools for automatic content annotation and those supporting manual annotation [38]. Because loomp targets at highly-accurate manual annotation, we focus here on existing research with similar goals. Again, the results of our analysis are summarised in the table in Figure 17 for comparison. We compare here those systems that are most closely related to loomp and those that are of current or historic relevance for the area.

One of the earliest and most widely-regarded semantic annotation tool is Annoeta [39], which supports creation of annotations for any retrievable Web document. Annotations become retrievable Web resources themselves; they stored as RDF on a user-defined annotation server. Similar to loomp, Annoeta builds upon the use of open standards in order to increase re-usability of content and decoupling of content and representation. However, Annoeta was designed before Linked Data principles became the driving force for the Semantic Web and technologies such as RDFa (or microdata) were introduced. Hence, Annoeta conforms to much older standards, several of which are now out of date. Finally, Annoeta does not allow to create new
content, but solely targets the annotation of existing Web resources.

The MnM tool supports both automatic and semi-automatic semantic annotation of Web resources [4]. At its core, MnM aims to combine Web browsing with ontology engineering. MnM provides APIs that link with ontology repositories as well as information extraction tools. In comparison to loomp, MnM follows a fundamentally different assumption about into which user task the annotation process should be embedded (authoring vs browsing). loomp is driven by a strong focus on the simplicity of the manual annotation process while authoring. For MnM, the annotation process itself is the task at hand.

S-CREAM is framework supporting manual and automatic annotation of content to populate the Semantic Web [6]. The reference tool OntoMat has been implemented using the S-CREAM framework. The core use-case of S-CREAM is discourse representation, which seeks to support the reading of a document by representing the logical interdependence of parts of the text. While both loomp and S-CREAM rely on Semantic Web standards, S-CREAM was developed before Linked Data gained momentum while loomp embraces Linked Data concepts. Both tools incorporate document management capabilities but S-CREAM does not support the creation of new mashups of annotated Web resources nor the linking of annotated entities within texts through semantic relationships.

The KIM platform supports automatic creation of semantic annotations for Web resources. It uses a generic upper ontology focusing on named entities and RDFS and OWL Lite representation. Source documents, annotations, and annotation vocabulary are decoupled. The differences between loomp and KIM are manifold, because loomp is focused on manual annotations and content re-use, while KIM focuses on technical extraction capabilities and indexing. Again, even though KIM supports web standards, it was developed before Linked Data became the standard paradigm.

C-PANKOW is a tool for automatic and semi-automatic annotations of Web resources [7]. It uses linguistic pattern matching to rank competing candidate annotations. C-PANKOW focusses on automation only, while loomp targets manual annotations as well as annotation sharing and content authoring. The C-PANKOW annotation mechanism is quite heavy-weight, making it unsuitable for environments with low technical expertise.

LORE [40] is a Firefox browser plugin that allows to annotate and link existing objects in a digital library. As it aims to support literary research, LORE’s development was guided by the IFLA Functional Requirements for Bibliographic Records [41] and the OAI-Object Reuse and Exchange [42]. LORE addresses six requirements: collaborative authoring; open-access publishing of the created compound objects; discovery of related objects based on annotations; expression of relationships between resources; intuitive graphical user interface to visualize relationships; and standardized annotation vocabulary. While LORE also heavily uses open standards (OAI-ORE) it is less integrated with Linked Data principles. This becomes particularly obvious for the annotation vocabulary, which can be exchanged by the user but by default does not include any reference to existing Web resources. This means that rather than motivating vocabulary reuse, LORE motivates the development of new domain vocabularies, which is considered a problem for an effective Linked Data lifecycle [43].

DOMEO [44] is a Web application for annotating Web resources using the domain-independent Annotation Ontology schema. The tool supports manual, semi-automatic, and automatic annotation with strong emphasis on keeping provenance of the annotations. The DOMEO use case is the integration of biomedical knowledge bases through annotations. Different to loomp, DOMEO only allows annotations of existing Web resources, particularly domain-specific knowledge bases to enable cross-system metadata exchange.

Semantic Turkey [45] supports the capturing and semantically annotation of deep links to information found in visited Web pages. The main aim is to close the gap between domain experts and knowledge engineers in the knowledge acquisition process. While clearly addressing a group that is usually considered technical non-experts in semantic technologies (i.e., the domain experts) [46] Semantic Turkey is positioned quite differently to loomp. It focuses on pure knowledge acquisition, while loomp focuses on the users’ task and any knowledge being the byproduct of what the user is primarily interested in doing.

The FLERSA tool supports the creation of hyper-text documents in which semantic metadata is embedded as RDFa [11]. Both manual and automatic annotation are supported. FLERSA offers a combination of text-based and ontology-based information retrieval. It supports existing standards and semantically links both instance and concept level. It has publicly accessible annotations and uses an evolving knowledge base for automatic annotation.
There are a number of parallels between loomp and FLERSA, the main one being that both systems combine the annotation and authoring processes. The main difference is FLERSA’s focus on classic monolithic Web documents vs loomp’s content mashups. We also observe that the FLERSA implementation masks the technicalities of semantic annotations (e.g. namespaces or the subject-predicate-object pattern of RDF triples) to a far lesser extend than loomp, which raises questions about the actual suitability of the tool for non-experts. Unfortunately, no end-user evaluations are available.

RDFaCE [9] is a hypertext editor with built-in RDFa annotation functionality. RDFaCE typically stores annotation within the user’s Web browser, it can also be used in integration with content management platforms such as WordPress. A wide set of APIs is supported for automatic annotation of text. Both, RDFaCE and loomp are developed based on the TinyMCE hypertext editor. In contrast to RDFaCE, loomp stores annotations with the annotated content in a RDF store, making both available as Linked Data. Finally, loomp’s OCA hides the subject-predicate-object pattern of RDF from the user to address the needs of technical non-experts.

Pundit is another tool for the annotation of existing Web content [12,47]. Pundit’s data representation combines RDF with Open Annotation data model. Collections of annotated web resources can be browsed using faceted filters. Sharing of annotations is supported in Pundit to enable particularly collaborative scholarly work in the digital humanities. In contrast to loomp, Pundit does not support content authoring. While the granularity of annotated content in Pundit is equally flexible as in loomp through deep linking, loomp additionally allows to mashup such fragments to form new Web resources.

One of the most recent and fundamental advances in the Linked Data field is the development of SOLID. SOLID aims to enable fully-decentralized applications in conformance with the Web architecture. A showcase application developed using SOLID is Dokieli [48]. Dokieli is a tool for authoring, commenting and semantically annotating (markup) Web content. The Dokieli use case is integrated eScience [49], including the support for nanopublications. Nanopublications could be regarded as very short fragments. So far, the Dokieli UI for enriching content is rather raw and requires considerable technical expertise.

### 6.1.2. Loomp as hypertext tool

loomp offers support for (1) decomposing monolithic text documents into atomic re-usable fragments of user-defined granularity; (2) annotating entities mentioned within those text fragments with reference to instance and schema resources served elsewhere on the World Wide Web; (3) allowing information consumers to leverage those annotations for user-defined visual highlighting and multi-faceted navigation within the information network.

Hence, our work on loomp builds upon the original vision of hypertext [50,51]. In contrast to the hypertext environments of the early days [52,53], loomp was designed with the Linked Data principles [54,55] as well as the Semantic Web vision [1,2] in mind. That means that not only the authored content can be consumed as a hypertext experience (in conformance to...
the Web architecture), but all parts of the semantic enrichment and interlinking are shared according to the same standards. loomp thus inherently supports strong decoupling of content and presentation at both content and semantic level. Ultimately, loomp has the potential become both a client and a server of semantically-enriched content – an intelligent read-write Semantic Web agent [56].

7. Conclusion

loomp is a tool for mashup authoring and semantic annotation building on the original vision of hypertext. It combines the manual annotation with the user task of authoring texts. loomp follows a strong focus on requirements by non-technical end-users. The requirements were developed in collaboration with knowledge workers and journalists and based on real-world journalism use-cases. The loomp design and development process followed a user-centred design approach to implement mashup and manual annotation of textual resources. This article provided technical details of loomp’s design, architecture and data model based on the identified requirements.

Future work on systems similar to loomp need to further explore the tracking of provenance for mashups and semantic annotation. While provenance has long been recognised as a core requirement for semantic annotations, the recommended W3C standard PROV [57] is not yet widely adopted.

loomp is designed as a lightweight tool with minimal installation requirements, so that it can widely be deployed on standard Web servers without specialists knowledge. For best support of small-scale server configurations, we envision an integration of loomp with Triple Pattern Fragments [58] to avoid heavy server load for complex query processing.

loomp integrates annotations with the authoring process in a manner typical for content management systems. This means that the system in its current design makes a rigorous distinction between the content publisher (also the content owner and owner of a particular loomp server instance) and the content consumer using a client. We see here promising options for fully embracing the Linked Data Platform standard [59] by additionally supporting external data manipulation in the loomp architecture. This can help to make a significant step towards fully decentralised content authoring and consumption enabling users to seamlessly switch between roles and removal of the distinction between clients and servers in this process. However, loomp’s approach to fragments and mashups goes beyond the current concept of static and atomic resources of the Linked Data Platform. It thus combines concepts of both read-write Linked Data (as supported in the Linked Data Platform standard) and non-linear reading (such as in microcosm [52] and Xanadu [50,51]). We believe it to be worthwhile to explore further how to to support the mashup of text fragments as an extension of the Linked Data Platform.

Overall, strong collaboration with end-users has shaped loomp’s design process as well as technical decisions. This integrated approach combining technical expertise with end-user consultation has served the loomp tool well, and allowed us to address aspects that would otherwise have remained hidden.

Acknowledgements

The authors thank the German Federal Ministry of Education and Research (BMBF) and the Royal Society of New Zealand for their support of this work as part of the research project “User-guided Semantic Enrichment”. This work has further been partially supported by the “InnoProfile Corporate Semantic Web” and “Corporate Smart Content” project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder-Entrepreneurial Regions.

References


