

RDFauthor: Employing RDFa for Collaborative Knowledge Engineering

Sebastian Tramp, Norman Heino, Sören Auer, and Philipp Frischmuth

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany
lastname@informatik.uni-leipzig.de
<http://aksw.org>

Abstract. In this paper we present RDFauthor, an approach for authoring information that adheres to the RDF data model. RDFauthor completely hides syntax as well as RDF and ontology data model difficulties from end users and allows to edit information on arbitrary RDFa-annotated web pages. RDFauthor extends RDFa with representations for provenance and update endpoint information. RDFauthor is based on extracting RDF triples from RDFa annotations and transforming the RDFa-annotated HTML view into an editable form by using a set of authoring widgets. As a result, every RDFa-annotated web page can be made easily writeable, even if information originates from different sources.

1 Introduction

To a large extent the overwhelming success of the World Wide Web was based on the ability of ordinary users to author content easily. In order to publish content on the WWW, users had to do little more than to annotate text files with few, easy-to-learn HTML tags. Unfortunately, on the semantic data web the situation is slightly more complicated. Users do not only have to learn a new syntax (such as N3, RDF/XML or RDFa), but also have to get acquainted with the RDF data model, ontology languages (such as RDF-S, OWL) and a growing collection of connected RDF vocabularies for different use cases (such as FOAF, SKOS and SIOC).

Previously, many approaches were developed to ease the syntax side of semantic authoring [11,2]. In this paper we present an approach, which also hides the data model from ordinary users and thus allows absolute novices to create semantic representations easily.

The *RDFauthor* approach is based on the idea of making arbitrary XHTML views with integrated RDFa annotations editable. *RDFa* [1] is the W3C Recommendation, which allows to combine human and machine-readable representations within a single XHTML document. RDFauthor builds on RDFa by preserving provenance information in RDFa representations following the named-graph paradigm and by establishing a mapping from RDFa view representations to authoring widgets. On configurable events (such as the clicking of a button

or moving over a certain information fragment with the mouse) the widgets will be activated and allow the editing of all RDFa-annotated information on the Web page. While editing, the widgets can access background information sources on the Data Web in order to facilitate the reuse of identifiers or to encourage the interlinking of resources. Our resource editing widget, for example, suggests suitable, previously defined resources derived from calls to the Sindice Semantic Web index [12]. Once editing is completed, the changes are propagated to the underlying triple stores by means of the SPARQL/Update language.

RDFauthor is not at all limited to editing semantic representations from a single source. An RDFa view made editable with RDFauthor can contain statements from a variety of sources, which can be edited simultaneously and in a wholly transparent manner for the user. Based on an extended RDFa markup supporting named graphs and SPARQL/Update endpoint information, simultaneous changes of several graphs from different sources will be dispatched to the respective SPARQL/Update endpoints. RDFauthor is implemented in JavaScript so that it works entirely on the browser side and can be used together with arbitrary Web application development techniques.

In particular with this paper, we make the following contributions:

- We define a light-weight extension of RDFa to accommodate named graphs and an RDF vocabulary to provide metadata on these graphs, such as provenance information.
- We develop the RDFauthor library, which can be used to make arbitrary RDFa representations editable. We provide a number of widgets for authoring common datatypes and define a mechanism for plugging in and automatically configuring additional editing widgets.
- We demonstrate the benefits of RDFauthor in three use cases: semantic authoring in OntoWiki, editing information from multiple sources in a combined vCard/publications view as well as collecting semantic data from any RDFa-enhanced page and pushing it to a personal RDF store.

As a result, RDFauthor radically simplifies the authoring of semantic information. Users can interact with Semantic Web applications without having to learn a new syntax or even having to get acquainted with the RDF data model or other knowledge representation formalisms. This advantage adds easy write support to Semantic Web applications, which can help them to enlarge their user bases significantly and to achieve generally a higher penetration of Semantic Web technologies.

The paper is structured as follows: We describe the requirements which guided the development of RDFauthor in section 2. We present our RDFa extension for representing named graphs and provenance in section 3. A description of our approach regarding architecture and implementation is given in section 4, while the approach is demonstrated on the basis of three use cases in section 5. Finally, we survey some related work in section 6 and conclude with an outlook on future work in section 7.

2 Requirements

In this section, we gather and describe the most important requirements, which guided the development of RDFauthor.

The idea behind the development of RDFauthor was to provide a general framework to edit data chunks (triple or multiple triples) in XHTML pages by means of small authoring components called editing widgets (or, as in the present paper, just widgets). The framework, which should be usable with arbitrary Web applications, has to provide edit functionality on top of RDFa-annotated web pages with only minor modifications of the existing markup, i. e. there should be no need to create special edit views. This mode will reduce the effort required for the development and maintenance of (Semantic) Web applications significantly. Judging from our experience with developing web application that focus on collaboration and interaction, we can assume that probably more than 50 % of the effort regarding the user interface creation is spent on implementing and maintaining edit functionality.

To allow mashing-up content from different sources, the framework should preserve the provenance of all content chunks, even if combined on a single resulting XHTML page. This possibility allows to hide even more complexity from the user, since she does not have to care about where to edit certain information or about switching between different editing views. To achieve this goal, we have to provide a vocabulary in order to connect RDFa fragments with updatable SPARQL/Update endpoints. RDFauthor should provide functionality not only to edit existing information, but also to create new data. The framework should also allow to distinguish between writeable and non-writeable information sources. In this way authentication and access control is easily combinable with RDFauthor, without increasing the complexity of the implementation for Web developers.

Moreover, in order to make the general editing framework as flexible as possible, the goal was to provide a number of authoring widgets for specific content types, such as resource references, dates, locations, images/files etc. The Web developer/designer should not be limited in her possibilities to create Web designs. RDFauthor should be as unobtrusive as possible and provide flexible editing widgets (or allow different configurations, e. g. via CSS definitions) for different use cases, such as inline editing, popup/overlay editing etc. RDFauthor should also retrieve background information (such as schema/vocabulary information with domain/range restrictions) required for the selection of appropriate widgets. Furthermore, it should facilitate the interlinking of information on the basis of the Linked Data paradigm and incorporate services, such as Sindice, DBpedia and Geonames, for establishing links.

3 Named Graphs and Provenance in RDFa

RDFa enables the annotation of information encoded in XHTML with RDF. This ability allows to extract a set of RDF triples from an RDFa-annotated XHTML

page. RDFauthor makes these triples editable, but in order to store changes persistently in the triple store that was used to create the RDFa annotations, RDFauthor needs information about the data source (i. e. SPARQL and SPARQL/Update endpoint) regarding the named RDF graph from which the triples were obtained or where they have to be updated. In order to make this information available, we have defined a slight extension of the RDFa annotations.

To represent information about the information source, we follow the named graphs approach [4]. We created a vocabulary¹ to represent attributes and relations for the following purposes:

- In order to link certain RDFa annotations on the page to the respective querying/update services, namely SPARQL/Update and SPARQL endpoints, we propose the use of the `link` HTML tag with an `about`-attribute to identify the named graph, a `rel`-attribute with the value `update:updateEndpoint` and a `href`-attribute with the URL of the respective SPARQL/Update endpoint. Another option to declare graph metadata is the use of empty `span`- or `div`-elements together with the RDFa attributes inside the body of the page. This option is particularly useful, if the program, which generates the RDFa-enhanced HTML code from the RDF store, does not have access to the `head` of the page (which is typically true for small content plugins in CMS or CMS-like applications).
- For declaring which statements belong to which named graph, we propose the use of the `update:from`-attribute with the named graph as attribute value to which all nested RDFa annotations should belong. The `update:from`-attribute and the additional RDFa processing rules are inspired by [7]. The use of named graphs is optional and only required, if triples from multiple sources should be made editable.

The next listing is an example of an RDFa-enhanced XHTML snippet from the vCard and publications mashup (which we describe as a use case for RDFauthor more profoundly in section 5). All RDFa attributes as well as our update vocabulary extensions are highlighted.

```

1 <head xmlns:foaf="http://xmlns.com/foaf/0.1/"
2   xmlns:update="http://ns.aksw.org/update/"
3   xmlns:dc="http://purl.org/dc/elements/1.1/">[...]
4 </head>
5 <div update:from="http://showcase.ontowiki.net/"
6   about="http://sebastian.dietzold.de/terms/me" typeof="foaf:Person">
7   
8   <b property="foaf:name">Sebastian Dietzold</b>
9   <a rel="foaf:phone" href="tel:+49-341-9732366">tel:+49 341 9732366</a>
10 </div>
11 <div about="http://showcase.ontowiki.net/"
12   rel="update:updateEndpoint" resource="http://trunk.ontowiki.net/sparul/" />
13 <div about="http://showcase.ontowiki.net/"
14   rel="update:queryEndpoint" resource="http://trunk.ontowiki.net/sparql/" />
15 <div update:from="http://publications.aksw.org/">

```

¹ The RDFauthor vocabulary namespace is `http://ns.aksw.org/update/`. We use the prefix `update` for this namespace throughout this paper.

```

16 <p about="http://www2009.eprints.org/63/1/p621.pdf" typeof="foaf:Document">
17 
18 <span rel="foaf:maker" resource="http://sebastian.dietzold.de/terms/me" />
19 <span property="dc:description">...</span>
20 </p>
21 </div>

```

After declaring all required namespaces in the page `head` (lines 1-4), two `div`-sections (starting in lines 5 and 15) contain RDFa annotations derived from two different named graphs. The graph URIs are specified by using the `update:from`-attribute. All nested RDFa annotations are parsed into RDF triples which belong to the given named graphs. The first graph contained in lines 4-10 consists of a vCard description of a `foaf:Person` and the second graph in lines 15-21 consists of information about a `foaf:Document` resource which is connected to the person using the `foaf:maker` relation. In addition to the FOAF vocabulary, properties from Dublin core and LDAP are used.

In order to annotate the named graph resources with the service locations, two more `div`-sections per graph are included (lines 11-14 associate one graph with two different endpoints for updates and queries). Here we use our `update`-vocabulary to link the SPARQL/update service (in this case an OntoWiki instance).

The XHTML listing above represents the simplified source code of the example screenshot from the mashup in figure 5. The XHTML page is parsed by the RDFauthor RDFa+named-graph parser into the triples (represented in N3 notation) shown in the following listing²:

```

1 <http://showcase.ontowiki.net/>
2   update:updateEndpoint <http://trunk.ontowiki.net/sparul/>;
3   update:queryEndpoint <http://trunk.ontowiki.net/sparql/>.
4
5 <http://showcase.ontowiki.net/> = {
6   <http://sebastian.dietzold.de/terms/me> a foaf:Person;
7   foaf:depiction <http://aksw.org/img/...>;
8   foaf:name "Sebastian Dietzold";
9   foaf:phone <tel:+49-341-97-32366>;
10  #[...]
11 }.
12
13 <http://publications.aksw.org/> = {
14   <http://www2009.eprints.org/63/1/p621.pdf> a foaf:Document;
15   dc:description "Soren Auer, Sebastian Dietzold, [...]";
16   foaf:maker <http://showcase.ontowiki.net/SoerenAuer>,
17             <http://sebastian.dietzold.de/terms/me> .
18 }.

```

The extracted model consists of two named graphs and additional statements in the default graph. For both of these named graphs, update and query information is available. The RDFauthor widget library treats all statements from graphs without update information as read-only statements.

4 System Architecture and Implementation

In this section we describe the architecture and implementation of RDFauthor in more detail. The basic cycle of how web pages are edited with RDFauthor is

² For reasons of limited space, we omit the first lines with prefix definitions for `foaf`, `dc`, `update` and `ldap`.

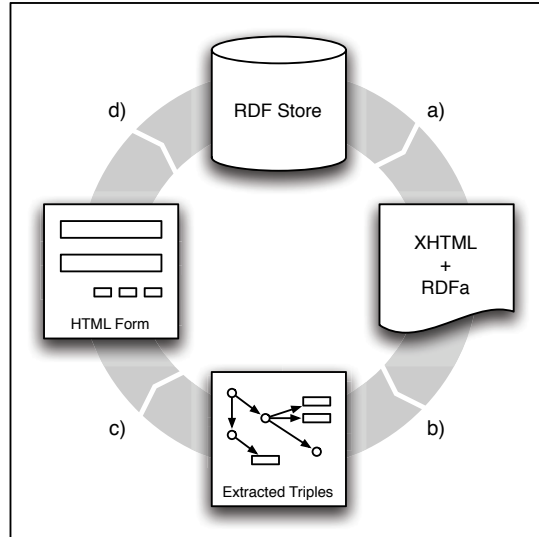


Fig. 1. Editing cycle for an RDFa-enhanced web page. The processes involved are a) page creation and delivery, b) client-side page processing, c) form creation and d) update propagation.

depicted in figure 1. It is composed of four distinct processes, three of which (b–d) are handled by RDFauthor components and are described in the subsequent sections.

Initiation of these processes can happen through a number of different trigger events. These events can be grouped into element-based events or page-wide events. In particular, the following triggers are supported:

- Clicking on an edit button next to an element containing the object of a statement,
- moving the pointer and hovering above an object element,
- an application-specified custom trigger similar to the button labelled “Edit Properties” in OntoWiki (see section 5),
- a bookmarklet which loads all RDFauthor components and runs all widgets at once,
- the universal edit button³.

4.1 Client-Side Page Processing

Upon user interaction or a programmatic trigger, RDFauthor starts processing the current page by extracting all RDF triples and placing them in an *rdfQuery databank*⁴ (cf. section 4.5); one for each named graph. Triples that

³ <http://universaleditbutton.org>

⁴ http://code.google.com/p/rdfquery/wiki/RdfPlugin#Creating_a_Databank_by_Hand

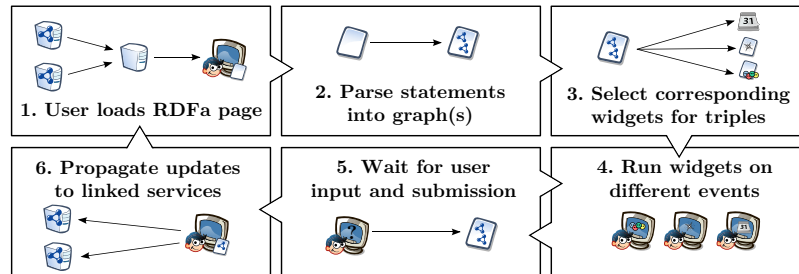


Fig. 2. Steps involved in the client-side processing of the page to be edited

describe the named graphs in the page by using the update vocabulary are excluded from editing. If no update information has been defined for a graph, it is considered non-editable, hence no form elements are created for the triples it contains.

Figure 2 depicts the default page processing procedure. Initially, the user loads an RDFa-annotated web page into her browser (1). She then triggers the parsing process by one of the possible edit triggers the developer of the page has decided to make available on his page (2). RDFa parsing and widget selection are performed lazily on the first of these events. For each statement on the page the corresponding widget is selected by an algorithm described in more detail in section 4.2 (3). An edit view is presented to the user in one of the ways described above. In which way it is shown is controllable by the author of the page (4). The user completes her editing tasks and submits her changes or cancels the whole process (5). In case of submission, the changes are propagated back to the services linked to each graph (6). In section 4.3 we describe this process in more depth.

4.2 Widget Selection and Form Creation

Widgets for editing existing statements are selected by exploiting the object's datatype and the property from the encoded RDFa model. If no datatype is present (plain literal or object property), a deployed selection cache of pre-calculated decisions is used.

For this cache, we analyzed 19 of the most frequently used namespaces listed by the *Ping the Semantic Web* service⁵. Together, these vocabularies describe 124 datatype properties and 176 object properties. For these 300 properties, we populated the widget selection cache with information on type and datatype of the properties used. This cache is made available as a JSON file. Most of the datatype properties requested a standard literal widget. Only 17 datatype properties had an integer range (float 8, date/time 4, boolean 2).

If the named graph from which the statement originates is linked to a SPARQL endpoint and neither the RDFa model nor our cache can provide useful hints as to which widget to use, RDFauthor tries to retrieve this information from

⁵ <http://www.pingthesemanticweb.com/>

the SPARQL endpoint by querying the `rdf:type` and `rdfs:range` of the property.

The selected widgets are combined into an edit view and are displayed to the user. Depending on the type of trigger, this can be done in one of the following ways:

- A single-statement overlay,
- a single-statement widget injected into the page or
- a bulk overlay containing widgets for all editable statements.

4.3 Update Propagation

When the user finishes the editing process, all widgets involved are asked to update the respective named graph with their changes. The difference between the original and modified graphs are calculated (i. e. added statements, removed statements), yielding a diff graph. The associated store to each graph is then updated with the respective diff graph by means of SPARQL/Update [9] operations. By explicitly listing all inserted or deleted triples using `INSERT DATA` and `DELETE DATA` syntax, sophisticated SPARQL/Update support is not required. In addition, RDFauthor can cope with several access control scenarios. It, therefore, evaluates the server's response to SPARQL/Update requests. For instance, in the case of an HTTP 401 (unauthorized) or 403 (forbidden) status code, a login form is displayed.

4.4 Statement Adding Methods

In addition to modifying the triple content of a page, it is possible to add new statements. This can happen either based on existing triples used as templates or by adding entirely new statements. If existing triples are used as templates, three cases can be distinguished:

- Creating a new statement that shares subject and property with an existing statement. Our approach supports this case via a small button beside each statement.
- Creating a new statement that shares the subject with an existing statement. At the end of a subject description a small button is shown which lets the user add a new statement to the subject's description.
- Creating a new resource using an existing resource as a template. Widgets for all properties found on the template resource are available on the new resource.

4.5 Architectural Overview

Putting the processes described above into perspective, three components can be identified that are involved in the cycle depicted in figure 1.

- An XHTML page annotated with RDFa and a named graph extension as described in the previous section,
- for each named graph that is intended to be writable: a SPARQL/Update endpoint to which updates are sent and an optional SPARQL endpoint to gather additional information (see below),
- the RDFauthor API with a set of editing components (called widgets) and included libraries.

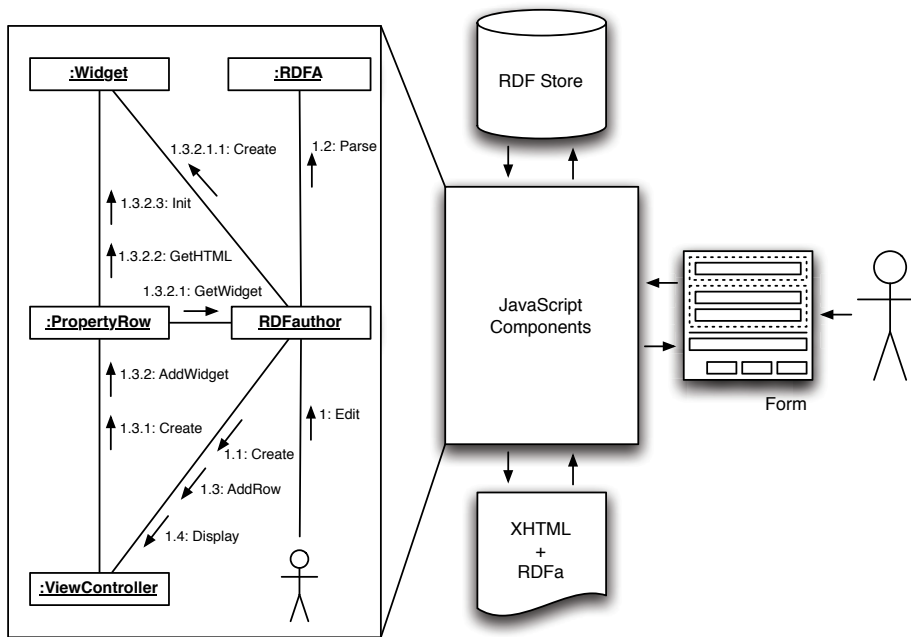


Fig. 3. RDFauthor architecture overview and UML communication sequence

In-page triple storage (databanks) and RDFa parsing are included from external projects. The JavaScript API has, thus, three components:

- RDFauthor JavaScript objects,
- an in-page RDF store based on the rdfQuery jQuery plug-in, developed by Jeni Tennison⁶,
- an RDFa parser component obtained from the W3C RDFa JavaScript implementation page⁷ (modified according to [7] in order to allow for parsing named graph attributes).

Our own contribution to this stack, namely the RDFauthor JavaScript objects, is a collection of scripts that allow the creation of an edit view and included

⁶ <http://code.google.com/p/rdfquery/>

⁷ <http://www.w3.org/2006/07/SWD/RDFa/impl/js/>

widgets. These widgets can be either included into the existing page or displayed as an overlay. The overlay approach provides sleek editing capabilities for even the most complex XHTML+RDFa markup, while the inline option can be used to integrate authoring functionalities seamlessly into existing pages.

5 Use Cases and Evaluation

In order to demonstrate the benefits of RDFauthor, we integrated the approach into two Semantic Web applications. Firstly, RDFauthor became the primary semantic authoring component in our Semantic Wiki OntoWiki. Secondly, we integrated RDFauthor into a text-based wiki application called WackoWiki, thus being able to demonstrate the simultaneous authoring of information from multiple sources. Finally, we describe a usage scenario facilitating the collection of RDF data from arbitrary RDFa-annotated websites.

5.1 OntoWiki

OntoWiki [2]⁸ is a tool for browsing and collaboratively editing RDF knowledge bases. It differs from other Semantic Wikis insofar as OntoWiki uses RDF as its natural data model instead of Wiki texts. Information in OntoWiki is always represented according to the RDF statement paradigm and can be browsed and edited by means of views, which are generated automatically by employing the ontology features, such as class hierarchies or domain and range restrictions. OntoWiki adheres to the Wiki principles by striving to make the editing of information as simple as possible and by maintaining a comprehensive revision history. It has recently been extended to incorporate a number of Linked Data features, such as exposing all information stored in OntoWiki as Linked Data as well as retrieving background information from the Linked Data Web. Apart from providing a comprehensive user interface, OntoWiki also contains a number of components for the rapid development of Semantic Web applications, such as the RDF API Erfurt, methods for authentication, access control, caching and various visualization components.

RDFauthor is used in OntoWiki both in the generic resource property view as well as in extensions which render resources in a domain-specific way (e.g. specific visualizations for SKOS concepts or FOAF persons). In order to perform the integration, we have extended OntoWiki in two ways:

1. We extended the default properties view for resources and all other views with RDFa attributes to annotate which data is presented as well as to link the graph to the internal update service. Since OntoWiki is entirely based on an RDF store, this extension was easy to implement. Likewise, all extension developers had to extend their views, e.g. for SKOS concepts.
2. We included RDFauthor by referencing it in the head of every OntoWiki page and adding JavaScript edit buttons on every page where data should be editable.

⁸ Online at: <http://ontowiki.net>

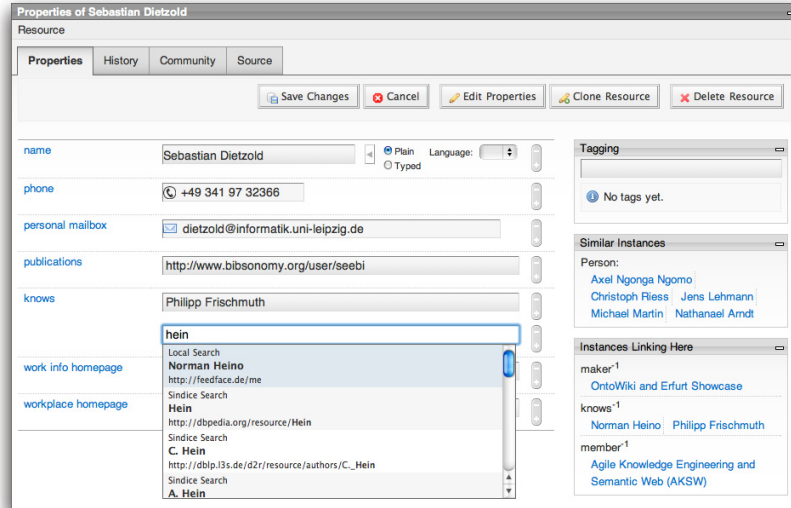


Fig. 4. OntoWiki with RDFauthor widgets in “inline mode”

The integration of RDFauthor into OntoWiki is displayed in figure 4. For all information displayed at the user interface, OntoWiki generates RDFa views which can be edited by using RDFauthor with a simple click on an edit button. In order to reuse previously defined resources as much as possible, we included a resource selector which searches for existing resources as the user is typing. A search for “Ber” would (amongst others) yield the DBpedia resource for Berlin⁹.

Adding new properties to an existing resource is accomplished in two steps. First, the user chooses a property which she wants to use. She types a name or description fragment into the search input field of the property widget and RDFauthor searches for properties in the referenced SPARQL endpoint of the given named graph. Subsequently, the corresponding widget is selected from the library as described in section 4.2.

As a result of the RDFauthor integration, OntoWiki is now able to handle not only different visualizations for specific content, but it can also use these views as a base for independent editing widgets, thereby achieving a new level of content versatility.

5.2 vCard and Publication Mashup

In order to showcase the simultaneous authoring of information from multiple sources, we integrated RDFauthor into the text-based wiki application WackoWiki¹⁰. WackoWiki is often used in small and medium companies as well as in small organizations such as research groups.

⁹ <http://dbpedia.org/resource/Berlin>

¹⁰ <http://wackowiki.org>

The AKSW research group uses WackoWiki for its entire web page (<http://aksw.org>) and integrates external data sources by means of so-called WackoWiki actions. Actions are small scripts which prepare some content and output it at the given position in the wiki page. Actions are also able to fetch data from external resources, allowing us to use structured information on different places in the wiki, e. g. by presenting the last publications selected by author, project or topic.

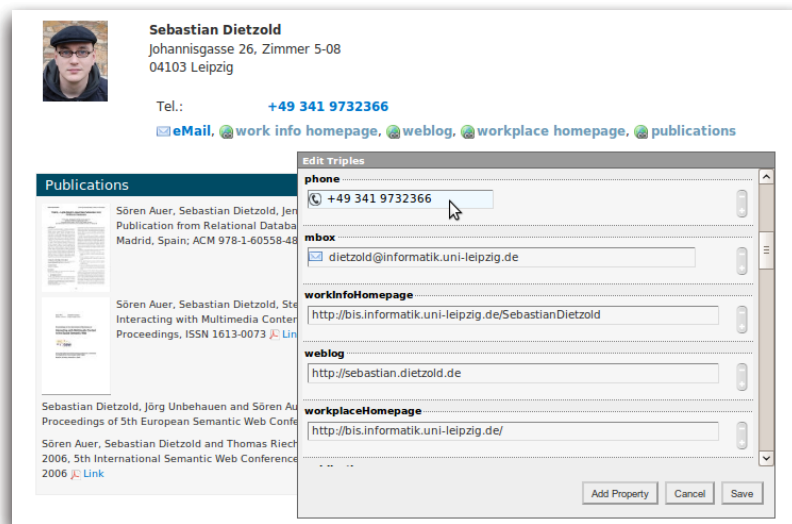


Fig. 5. RDFa-enhanced FOAF vCard and publications mashup with statements from different named graphs. In addition to the plain literal and resource widgets, we developed widgets for the special URI schemes `tel:` and `mailto:`, which hide the URI syntax behind a plain input field.

While integrating and presenting this information is easy and covered by many applications and techniques, the read/write integration of such external resources is tackled by RDFauthor. By employing RDFauthor, users of our wiki are able to edit both the wiki page and the structured information in one place and avoid using different web applications for one edit task and with different data.

We have developed two actions for integrating two different resources: public vCard information and a publication database. Both sources are available as RDF data and accessible via SPARQL endpoints. The output of these actions included in the author's wiki page is displayed on figure 5.

The displayed page is a mashup of three sources: static wiki content, vCard RDF data and RDF data about publications using the FOAF vocabulary. The output describes two named RDF graphs with RDFa attributes as introduced in section 3. Both graphs are annotated with corresponding SPARQL/Update

services. This annotation allows RDFauthor to pass the changes back to the databases from which they originate.

In doing so, a user who wants to edit her contact details (e.g. because she moved to another office room) can change this information directly where she notes the old and obsolete information.

5.3 Data Collection from RDFa Websites

Another interesting usage scenario, which is more concerned with collecting data instead of editing, is described in this section. Most of the RDFa-enabled pages on the web do not yet contain provenance and update information. However, RDFauthor also allows to use an arbitrary update endpoint, which does not necessarily have to match the originating endpoint.

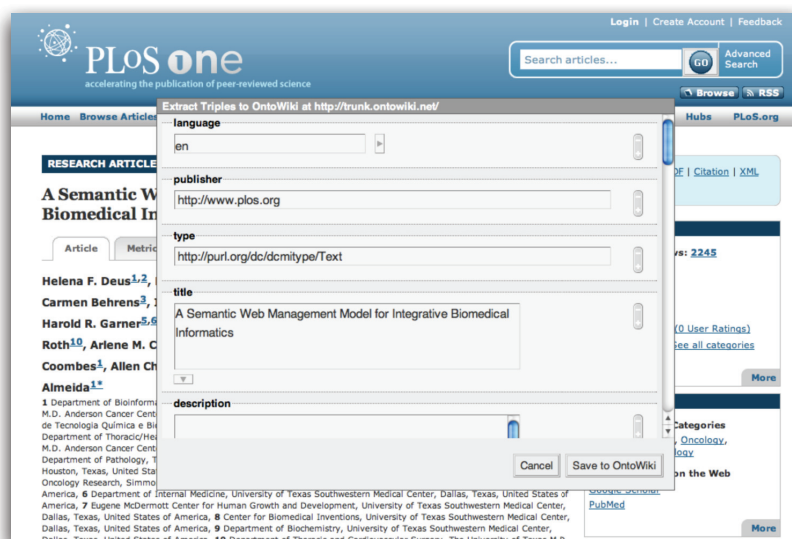


Fig. 6. An RDFauthor overlay view containing widgets for triples extracted from PLoS web page underneath

Since a SPARQL/Update-capable RDF store and a target graph is all the information required for using RDFauthor, it is easy to embed these into a bookmarklet used to initialize the editing process. In this case, the number of possible SPARQL/Update endpoints is limited to those under one's control. RDFauthor extracts the data from any page visited and displays the edit form. The data can be revised and unwanted statements can be removed from the view. Saving works, however, differently: instead of propagating the changed data back to the original source, it is sent to one's own RDF store and saved into the previously set-up graph.

6 Related Work

The problem of making Semantic Web content writable in an easy-to-use manner has been recognized by a number of authors. Pushback [6], for example, tackles this problem by providing a vocabulary and methodology for bi-directionally mapping Web 2.0 data sources and APIs to RDF. Since it relies on predefined vocabulary transformations from said sources into an RDF vocabulary describing edit forms (RDFForms), its use is limited to cases where such a mapping already exists.

Earlier in [5] we presented a JavaScript API that allows the independent creation of editing widgets for embedded RDFa. The ideas in this paper build upon the concepts discussed there. [10] present a document-style editing model over RDF data, which, like RDFauthor, is based on commonly available HTML manipulation tools and `rdqQuery`, a JavaScript RDFa library, to maintain an RDF model embedded in the page. We use part of this work (the `rdqQuery` library) in our client-side JavaScript stack. Likewise, Tabulator [3] allows modification and addition of information naturally within the browsing interface and allows to relay changes to the server. However, due to Tabulator's nature of being a generic data browser, little effort is made to cater users unfamiliar with the RDF data model.

Loomp [8] aims at providing a user interface for both creating textual content as well as annotating this content by using semantic representations. However, the focus of Loomp is not on authoring RDF content in the first place, but only as annotations of texts.

7 Conclusion and Future Work

We presented RDFauthor, a pragmatic and light-weight approach to make arbitrary RDFa views editable. RDFauthor does not only simplify the syntactic editing of semantic representations, but it also allows to hide the RDF and related ontology data models from novice users completely. Thus, RDFauthor contributes to enabling more users to employ and interact with Semantic Web applications successfully. Since RDFauthor converts an RDFa-annotated view directly into an editable form, which is an additional benefit, the costs for the development and maintenance of (Semantic) Web applications can be significantly lowered.

Regarding future work, we aim at integrating RDFauthor into more (Semantic) Web applications and at establishing a repository of forms and widgets for common vocabularies and datatypes. Based on such a comprehensive repository of common vocabulary renderings, RDFauthor could evolve into a participatory semantic mashup technology.

References

1. Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and Processing. Recommendation, World Wide Web Consortium, W3C (October 2008), <http://www.w3.org/TR/rdfa-syntax/>

2. Auer, S., Dietzold, S., Riechert, T.: *OntoWiki – A Tool for Social, Semantic Collaboration*. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) *ISWC 2006*. LNCS, vol. 4273, pp. 736–749. Springer, Heidelberg (2006)
3. Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Pru d’ommeaux, E., Schraefel, M.C.: *Tabulator Redux: Writing Into the Semantic Web*. Technical report, Electronics and Computer Science, University of Southampton (2007)
4. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: *Named graphs, provenance and trust*. In: *WWW 2005*. ACM, New York (2005)
5. Dietzold, S., Hellmann, S., Peklo, M.: *Using JavaScript RDFa Widgets for Model/View Separation inside Read/Write Websites*. In: *SFSW 2008*, CEUR, vol. 368 (2008)
6. Hausenblas, M., et al.: *Pushback – Write Data Back From RDF to Non-RDF Sources*. ESW wiki (2009), <http://esw.w3.org/topic/PushBackDataToLegacySources>
7. Inkster, T., Kjernsmo, K.: *Named Graphs in RDFa (RDFa Quads)* (January 2009), <http://buzzword.org.uk/2009/rdfa4/spec>
8. Luczak-Roesch, M., Heese, R.: *Linked Data Autoring for non-Experts*. In: *Workshop on Linked Data on the Web, Madrid* (2009)
9. Seaborne, A., Manjunath, G.: *SPARQL/Update: A language for updating RDF graphs*. Technical Report Version 5: 2008-04-29, Hewlett-Packard (2008), <http://jena.hpl.hp.com/~afs/SPARQL-Update.html>
10. Styles, R., Shabir, N., Tennison, J.: *A Pattern for Domain Specific Editing Interfaces Using Embedded RDFa and HTML Manipulation Tools*. In: *SFSW 2009*, CEUR, vol. 449 (2009)
11. Tudorache, T., Noy, N.F., Tu, S., Musen, M.A.: *Supporting Collaborative Ontology Development in Protégé*. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) *ISWC 2008*. LNCS, vol. 5318, pp. 17–32. Springer, Heidelberg (2008)
12. Tummarello, G., Delbru, R., Oren, E.: *Sindice.com: Weaving the Open Linked Data*. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) *ASWC 2007 and ISWC 2007*. LNCS, vol. 4825, pp. 552–565. Springer, Heidelberg (2007)