Content-sensitive User Interfaces for Annotated Web Pages

Florian Schmedding
Department of Computer Science
Albert-Ludwig-University Freiburg, Germany
schmeddi@informatik.uni-freiburg.de

Abstract: By means of RDFa it is possible to embed semantic meaning into standard XHTML web pages. Using the meaning, we provide content-sensitive user interfaces for web pages integrating the embedded annotations with information from other accessible data sources. To illustrate this concept, we show an example where appointment data gets related to the entries of the user’s personal calendar. For the implementation of our approach we propose an extension to the XML document object model that provides access to the annotations by SPARQL and we introduce new use cases for RDFa that use our approach.

1 Introduction

A broad range of different semantic technologies have been developed and standardized over the last years, but browsing on web sites rarely takes advantage of them although semantic descriptions can be supplied with RDF. We see one reason for it in the missing linkage between this data and the page content being presented to the user. Consequently, the underlying data cannot be addressed via an interaction with the visual representation. Despite having already found what one was looking for in the document, selecting the according data in a separate location is necessary for further automatic processing.

Recently, a new formalism—RDFa [AB08, Hau09]—that seeks to close this gap has been standardized by the W3C. It defines some additional attributes for XHTML and a mapping from the attribute values and the document content into RDF triples. An important feature in our context is the usage of selected text parts for literal properties and hence the explicit linkage between the human- and the machine-readable representations. In contrast to the mapping, the handling of this linkage is left open in the reference [ABMP08].

1.1 Web Page Annotation

Generally viewed, RDFa is an annotation approach like Annotea1, MnM [VVMD+02], S-Cream [HSC02], or GRDDL (Gleaning Resource Descriptions from Dialects of Languages) [Con08]. Yet we think that RDFa is best suited to build user interfaces that react

---

1http://www.w3.org/2001/Annotea/
on the semantic meaning for the following reasons. First, annotations in RDFa are primarily designed to make the document content machine-readable and rather not to make additional meta statements about it. Approaches like Annotea do not make this distinction. Second, the necessary linkage is explicitly given due to the direct embedding into the document. This means also that changes of the content are reflected in the semantic descriptions. In Annotea, annotations are stored externally and the linkage has to be established by XPointer expressions. GRDDL includes the linkage in XSL transformations. Third, RDFa straightly encodes RDF triples. MnM and S-Cream do also make a direct embedding of their annotations but further knowledge is required to convert them into RDF, as it is in GRDDL and Annotea, too. The recent increase of systems and web sites applying RDFa (e.g. Drupal\(^2\), Slideshare\(^3\), Yahoo SearchMonkey\(^4\), STW Thesaurus for Economics\(^5\)) also supports our point of view.

### 1.2 Contribution

In this work, we will introduce a novel way of handling RDFa within the document object model and show its utility for several new use cases for RDFa documents. We propose an integration of the RDF statements into the document object model of the XHTML pages because we think that a generic way to handle RDFa annotations covers much of the resemblances between the use cases and therefore eases their handling.

In particular, we focus on applications and user interfaces that relate the annotations on web pages to knowledge from other sources. This means they operate in two directions: extracted data from the page can be forwarded and response data can be integrated right

\(^2\)http://drupal.org
\(^3\)http://www.slideshare.net
\(^4\)http://developer.yahoo.com/searchmonkey/
\(^5\)http://zbw.eu/stw/versions/latest/about.en.html
into the displayed page in the browser, close to the respective annotated elements. Of course, this approach is not limited to web browsers but applies to HTML documents in general. For instance, it could be used also for semantic e-mails (cf. [SDHH09]) when using HTML messages.

Based on the first use case, we will sketch out our prototype application (a plugin for Microsoft’s Internet Explorer, partly implemented in [Sch09], see Fig. 1) that implements all but two of our use cases and show how our integration helps to accomplish the task.

2 Related Work

There are several other tools for RDFa available at the moment. A recent extension for the Firefox browser is Fuzz, provided as a generic substructure exposing RDFa triples to other plugins that make further use of them. In contrast to our approach it is not integrated into the DOM. Operator provides context menu actions on annotated document text and menus to handle different types of data. It allows the user to add new actions into these menus. It is mainly used to export data and does not propose a generic handling of RDFa.

The RDFa Bookmarklets are a set of examples for RDFa data based on a JavaScript library to parse RDFa. The library does not maintain the linkage between annotations and elements, however, it is accessible during the parsing via callbacks. Another JavaScript approach is rdfQuery following the jQuery-style to handle RDF within web pages. It features SPARQL-like queries and gleans RDF statements from RDFa annotations. Compared to our approach, rdfQuery does not aim to integrate RDFa into the document object model and is less about the linkage between semantic statements and document elements.

In [GWB08], a visualization for annotated documents is developed. However, the linkage in the model seems to be specialized to their application domain, patent documents. A completely different approach is taken by [CHM08]. The authors propose a conversion from HTML documents into an RDF model. We think that such an approach could be useful for reasoning about the document structure itself, but it seems less appropriate for dealing with the document’s visual representation.

We see our approach as a step towards a native implementation of RDFa into web browsers and other XML processors. We also provide an integration of RDFa into sequential (SAX) XML parsers and offer a generic API to process RDFa documents independently from a browser, e.g. as part of a Semantic Web crawler.

6http://rdfa.digitalbazaar.com/fuzz, the derivative of Fuzzbot
7http://www.kaply.com/weblog/operator/
8http://www.w3.org/2006/07/SWD/RDFa/impl/js/
9http://code.google.com/p/rdfquery/
10http://jquery.com/
3 Access to Annotated Documents

Modern web browsers operate on the DOM-model of an XHTML document to display it for human consumption, and user interactions with the document content are mapped to elements in the DOM-tree. To enhance a browser with new features for semantically annotated data, we have to integrate this data into the browser’s document representation, hence into the DOM-model. Our extension to the DOM-model thus concerns the document object itself and its elements. Both need to expose methods that provide an easy access to the contained annotations. According to the tree structure and the DOM behavior, the methods should work on global, local, and subtree level. We will illustrate the details by reference to the markup in Fig. 2. The contained statements are:

\[
dbisl1:11 \text{ a cal:Vevent ; cal:summary "Introduction" ; cal:dtstart "2008-10-21T09:15:00"^^xsd:dateTime ; cal:duration "PT95M"^^xsd:duration .}
\]

3.1 SAX Parser

Before building the DOM extension we integrated the RDFa parsing into a sequential XML parser. We decided to add the properties only to element nodes. The new properties are\(^{11}\): subject, predicates, object, literal, language, datatype, types. Two boolean properties, isIncomplete and isSkip, indicate whether the current element contains incomplete triples or whether it does not contain any RDFa markup at all, respectively. Statements are stored in collection attached to the element node where they are completed. The only exception to this rule are statements consuming inner text for their literal values. Such statements cannot be finished before the closing tag for the element where they are defined has been read; therefore they are added to a collection assigned to the end element node. Of course, no data is accessible until the base URI has been determined.

\(^{11}\)Properties written in plural may contain more than a single value according to RDFa.
3.2 Integration into the DOM

Constructing the DOM tree when having the SAX parser is straightforward. After subclassing an existing DOM implementation we have to adapt only the construction of new elements. We extend the element objects with similar properties as above to make the parts of RDFa annotations visible. All statements that are available for the current element are put into its local triple store\textsuperscript{12} Statements. We say that the scope of these statements is the containing element and its ancestors and define the property \textit{Store} as: For document \(d\) and all elements \(e_i \in d\), \(c_1 \ldots c_n\) being the child elements of \(e_i\): 
\[
e_i.\text{Store} = e_i.\text{Statements} \cup c_1.\text{Store} \cup \ldots \cup c_n.\text{Store}
\] 
In the document object itself we let \textit{Store} refer to the root element’s triple store. The document subsumes all statements contained in its markup. To illustrate the concept, Fig. 3 shows the DOM-tree for the markup in Fig. 2. The according allocations of \textit{Statements} and \textit{Store} of elements \(e_2\) and \(e_3\) are given in the following equations (datatypes are omitted).

\[
e_{2}.\text{Statements} = \{ (\text{dbis1:ll,cal:dtstart}, \text{2008-10-21T09:15:00}) \} \quad (1)
\]
\[
e_{2}.\text{Store} = \{ (\text{dbis1:ll,cal:dtstart}, \text{2008-10-21T09:15:00}), \quad (2)
\]
\[
\{ (\text{dbis1:ll,cal:duration}, \text{PT95M}) \}
\]
\[
e_{3}.\text{Statements} = \{ (\text{dbis1:ll,cal:summary}, \text{Introduction}) \} \quad (3)
\]
\[
e_{3}.\text{Store} = \{ (\text{dbis1:ll,cal:summary}, \text{Introduction}) \} \quad (4)
\]

3.3 Linkage between elements, statements, and resources

The linkage is implemented by methods similar to the DOM-method \texttt{getElementsByTagName}. When invoked on an element, \texttt{getElementsByTagName} returns all child elements that contain a certain resource, whereas \texttt{getElementsByTagName} returns all child elements that match the specified statement or template. If the element

\textsuperscript{12}We use the SemWeb library, cf. \url{http://razor.occams.info/code/semweb}.
itself fulfills the given condition, too, it will be also part of the result. The method `getElementsByQuery` runs a given SPARQL query over an element’s store and returns lists with elements containing the resources in the query variable bindings.

To relate statements with elements, we first used the meta property of statements. But to keep it free for named graphs (its basic intention) we opt to add an extra property to statements for the linkage purpose. It stores a reference to the element where the statement was completed. For resources we proceed likewise and put a reference to the annotated element into it. Thus for any resource, as part of a statement or query result, the origin within the DOM-tree can be resolved. This interweaving of statements, resources, and elements saves separate structures to maintain the linkage.

When SPARQL queries are not required, Statements and Store could be implemented with simple collections like in the SAX parser. However, SPARQL is useful in many cases—for instance, to filter statements with literal objects or to compare literals.

4 Content-sensitive User Interfaces

A variety of RDFa use cases have been elaborated in [AH07] by the respective W3C working group. In the following, we will contribute some new scenarios that lead to innovative user interfaces for web pages. We call them content-sensitive because they adapt their behavior to the meaning of the encountered data. The meaning is defined in terms of RDF, and knowledge is built into the browser to handle different domains. The use cases called unit conversion and input suggestions are not part of the current prototype.

In a glance, we use RDFa not only in one but in two directions: we let the web browser extract data and receive further data from other sources to provide advanced feedback for users. Knowing the linkage between text and meaning is especially necessary in the second case. The first use case exemplifies the application of our previously described approach.

**Appointments.** Our browser plugin contains some knowledge about appointments and knows where to find other appointments of the user. When encountering matching data on a web page, it queries the user’s personal calendar for overlappings between his entries and the dates of the page. Conflicts are indicated by highlighting the (literal) properties of an event on the web page. With this approach, the user is only notified about the information he is actually reading, hence interested in. Retrieving more details on conflicting elements or sending appointments from the page to the calendar can be done via a context menu.

The realization of this task is as follows: First, all resources classified as `cal:Vevent` have to be retrieved by querying the document’s global store. Second, after obtaining information about the conflicts from the personal calendar, the literal (visible) properties of the involved resources have to be determined. This is achieved by the method `getElementsByQuery` and the following query (here for `dbis1:l1`):

```sql
select ?literal
where { dbis1:l1 ?p ?literal
  FILTER(isLiteral(?literal)) } 
```
Then the resulting literals are highlighted. Third, additional information, e.g. the summary of the conflicting entry or the calendar that contains it, can be retrieved from the personal calendar. It can be shown as tooltip for the before identified literal elements. This functionality is illustrated in Fig. 4.

The connection between the browser and the calendar is currently hard-coded within the respective plugins (it suffices to run the above use case). For a better integration into semantic desktop technologies, the communication bus system and the RDF adaptors of an environment like Gnowsis [Sau03], or Haystack [QHK03] could be used in future work.

Coordinates. The toolbar provides a context menu also on page elements which have associated geographical coordinates to open an extra window showing the location on Google Maps. The context menu also appears when the mouse points at the subject of coordinate properties. In [SHH08], we described this use case for our RDFa wiki about geo-political data.

Unit conversion. Automatic unit conversion according to user preferences can take place for annotated data. Due to the link between value and measure both literals are converted consistently in the following example.

```
<tr><td typeof="commerce:Price">Your price:
   <span property="commerce:amount">2.99</span>
   <span property="commerce:currency" content="USD">$</span></td></tr>
```
Information browsing. Using a back-end database, our toolbar is able to relate information from different web pages. We implement support for an online shopping scenario in our prototype. While browsing shops all product data is collected. By pointing with the mouse on products from new shops, the prices of items encountered before are shown. In contrast to other price comparing services, this approach is useful when the customer does not yet know exactly what to buy but wants to read descriptions or to get an impression of different stores. Not only the price but also various other criteria may influence his decision, as for example the trustworthiness and the product range of a shop.

Piggy Bank [HMK07], a well-known browser plugin, offers a similar functionality. However, it displays the related information beside the document because it does not have linkage information from RDFa.

Input suggestions. Some of the most popular web applications are search engines. To avoid effort, many people enter only few keywords in search forms although they often have more information at their disposal to limit the search space. If someone, for example, enters a surname in a telephone directory, and the input field is annotated like

```html
<input type="text" about="#person" rel="form:accepts" resource="foaf:name" />
```

an agent can suggest forenames from matching entries in the user’s address book, expecting more precise results.

The same mechanism applies to forms with multiple input fields. Even if no further input is provided, the agent will be able to highlight matching results on the returned page. This is particularly useful when the user wants to complete data records in his personal store. For online shopping, the agent can also automatically enter the user’s shipping and billing information into the respective forms. Critical parts like credit card numbers can thus be stored in a secure place rather than in the browser or in the online shop itself without the need to provide it manually for every order.

Rearrangements. By rearrangements we mean changing the order of some document elements according to certain criteria. Our idea is different from approaches like Greasemonkey\textsuperscript{13}, its extension Platypus\textsuperscript{14} and Chickenfoot [BWR\textsuperscript{05}] in that we do not use document-specific scripts but operate on the meaning of the page elements given by RDFa. We envision generic rearrangements for datastructures like lists as explained in the following example. The page snippet shows a table with information about movies (title and runtime) and is annotated as RDF container:

```html
<table typeof="rdfs:Container" rel="rdfs:member">
<tr><th>Title</th><th>Duration</th></tr>
<tr about="#nemo" typeof="mv:Movie">
<td property="dc:title">Finding Nemo</td>
<td property="mv:length" datatype="xsd:int">100</td></tr> [...]
```

Sorting the data records according to the runtime to find out the shortest movie can be done with a small SPARQL query using \texttt{order by}:

\texttt{order by:}

\texttt{13http://greasemonkey.mozdev.org/}
\texttt{14http://platypus.mozdev.org/}
select ?e
where { ?b a rdfs:Container.
    ?b rdfs:member ?e.
    ?e mv:length ?l }
order by asc(?l)

Likewise, the order may depend on a personal profile of interests. For instance, result pages from web search engines can be re-ordered according to such a profile if the given sorting does not fit the user’s needs.

5 Discussion and Future Work

We have presented a novel model that integrates DOM and RDF to handle RDFa-annotated web pages in web browsers and other XML processors. New applications are enabled by means of this model and their benefit for various use cases has been shown: Including feedback from personal calendars into web pages, visualization of geographic locations, unit conversion, support for browsing related information, and input suggestions as well as personalization of pages based on semantic profile data. Generally speaking, our model is able to provide additional functionality on focused objects on a page due to the relation between RDF data and DOM elements. But it is not restricted to RDFa. Other annotation approaches may also insert statements into the described properties to enable content-sensitive user interfaces over their data.

Currently we are working on an adaption of the approach in [SSS09] for this prototype. We intend to recognize possible patterns around annotated literals in order to enable the described rearrangements even without explicitly given collections.

RDFa has the possibility of masking visible literals with hidden values. This is intended for literals that require a certain representation for RDF statements (e.g. dates) but may be misused to create a mismatch between the human- and the machine-readable content. This rises the question for methods for preventing users being led astray because otherwise they may lose confidence in semantic data.

References


