The RDFa Content Editor –
From WYSIWYG to WYSIWYM

Ali Khalili and Sören Auer
Universität Leipzig, Institut für Informatik, AKSW,
Johannisgasse 26, 04103 Leipzig, Germany
{lastname}@informatik.uni-leipzig.de
http://aksw.org

Abstract. Recently practical approaches for managing and supporting the life-cycle of semantic content on the Web of Data made quite some progress. However, the currently least developed aspect of the semantic content life-cycle is the user-friendly manual and semi-automatic creation of rich semantic content. In this paper we present the RDFaCE approach for combining WYSIWYG text authoring with the creation of rich semantic annotations. WYSIWYG text authoring is meanwhile ubiquitous on the Web and part of most content creation and management workflows. Our approach is based on providing four different views to the content authors: a classical WYSIWYG view, a WYSIWYM (What You See Is What You Mean) view making the semantic annotations visible, a fact view and the respective HTML/RDFa source code view. The views are synchronized such that changes made in one of the views automatically update the others. They provide different means of semantic content authoring for the different personas involved in the content creation lifecycle. For bootstrapping the semantic annotation process we integrate five different text annotation services. We evaluate their accuracy and empirically show that a combination of them yields superior results.

Keywords: RDFa, semantic content authoring, text annotation, rNews

1 Introduction

Recently practical approaches for managing and supporting the life-cycle of semantic content on the Web of Data made quite some progress. On the backend side, a variety of triple stores were developed and their performance and maturity improves steadily. Similarly tools and algorithms for linking data and schemata are progressing and approaches are deployed for the use on the emerging Web of Data. The quantity of semantic content being made available on the Data Web is rapidly increasing, mainly due to the use of automated knowledge extraction techniques or due to the semantic enrichment and transformation of existing structured data. Despite many interesting showcases (e.g. Sindice, Parallax or PowerAqua), we still lack more user friendly and scalable approaches for the exploration, browsing and search of semantic content. However, the currently least
developed aspect of the semantic content life-cycle is from our point of view the user-friendly manual and semi-automatic creation of rich semantic content.

In this paper we present the RDFaCE approach for combining WYSIWYG text authoring with the creation of rich semantic annotations. WYSIWYG text authoring is meanwhile ubiquitous on the Web and part of most content creation and management workflows. It is part of Content Management Systems, Weblogs, Wikis, fora, product data management systems and online shops, just to mention a few. Our goal with this work is to integrate the semantic annotation directly into the content creation process and to make the annotation as easy and non-intrusive as possible. This is achieved by accompanying the classical WYSIWYG and source views with views facilitating the semantic annotation. We devise the concept of a What You See Is What You Mean (WYSIWYM) view, which extends the WYSIWYG view with highlighted semantic annotations. In addition a fact view helps content authors, curators as well as content and knowledge engineers to quickly review and possibly revise the semantic annotations. The rationale behind our WYSIWYM concept is that we have to provide an environment to the user, which she is sufficiently familiar with, but at the same time enables her to understand, access and work with semantic annotations. Hence, WYSIWYM is an extension of the WYSIWYG concept, where users can observe and manage the semantic annotations right in the familiar context of the WYSIWYG environment. Depending on the focus of work (or the persona using the tool) the various RDFaCE views provide a different balance between the visualization of the semantic annotations and the formatted textual content.

Despite the improved usability of semantic content creation we aim to achieve with this work, the manual semantic annotation of longer articles can still be a tedious and time-consuming task. In order to kick-start the annotation, we integrated five automatic annotation services into RDFaCE. During an extensive evaluation of these services, we observed that each individual tool has either weaknesses regarding precision, recall or the support of certain information domains. Our evaluation also reveals, that a substantial increase in precision and recall can be achieved, when combining the output of several services.

The contributions of this work are in particular:

1. An architecture and implementation of an RDFa authoring environment called RDFaCE (RDFa Content Editor) based on different views on the semantic content including a WYSIWYM view.
2. An extensive evaluation of five automatic text annotation APIs wrt. precision and recall in the domains wiki, blog and news articles.
3. An approach for the combination of different text annotation services, that yields superior performance compared to each individual approach.

The remainder of this article is structured as follows: Section 2 describes the architecture of the system. Section 3 presents the different views for semantic text authoring. In Section 4 the combination of different NLP APIs for bootstrapping the semantic annotation process is discussed. Section 5 introduces three RDFaCE use cases. Related work is presented in Section 6 and finally Section 7 concludes with an outlook on future work.
2 Architecture and Implementation

The RDFaCE system architecture is depicted in Figure 1 and consists of three layers. The foundation layer on which we ground the RDFaCE plugin includes the TinyMCE Rich Text Editor\(^1\). This open source HTML editor was chosen because it is very flexible to extend and is used in many popular Content Management Systems (CMS), blogs, wikis and discussion forums, etc. Therefore, by focusing efforts on this one particular editor, it is possible to quickly propagate accessible RDFa authoring practices to a number of other tools\(^1\). The RDFaCE implementation is open-source and available for download together with an explanatory video and online demo at http://aksw.org/Projects/RDFaCE. RDFaCE includes the following components:

Annotator UI. This component uses the TinyMCE API as well as jQuery UI to create user friendly user interfaces for RDFa content editing. As shown in Figure 2, the normal annotation procedure consists of four steps: 1) Defining appropriate namespaces. 2) Selecting a fragment of the text. 3) Assigning the subject (and type) to be used for the selected fragment. 4) Inserting triples by assigning properties. Besides these steps, RDFaCE provides some shortcuts to

\(^1\) http://tinymce.moxiecode.com
expedite the creation of new triples. For instance users can use the context menu and select from a list of predefined properties to instantly add a triple. After annotations are received by users, they are delivered to RDFa DOM manipulator.

**RDFa DOM Manipulator.** This component is responsible for manipulating the Document Object Model (DOM) according to the desired RDFa annotation. The simplest solution to add RDFa attributes to content is using `<span>` tags. For each new annotation, a new `<span>` can be created containing related RDFa attributes. Although this approach is simple to implement it generates a lot of redundant `<span>` tags. It might also result in invalid HTML code when annotating a block of content which already has a `<div>` tag. This is due to the fact, that `<div>` is a block-level element whereas `<span>` is an inline element according to the HTML standard. To cope with these issues, the RDFa DOM Manipulator component tries to find the valid and optimized annotation which manipulates original content as minimally as possible. Before adding a new tag for annotation, it tries to see whether it is possible to add the annotation to an existing tag. If this is possible, it will update the current tag rather than adding a new HTML tag. In case a new tag is required, it also employs either `<span>`

---

**Fig. 2.** Annotation user interface.
or `<div>` tags depending on whether the content is a block or inline element to prevent invalidness of HTML code.

**Inline Semantic Visualizer.** The main goal of the inline semantic visualizer is to provide a kind of on-demand visualization which can be included/excluded on the fly within the WYSIWYG content editing. This component uses a set of predefined CSS styles to distinguish the semantically annotated content from the normal content. To visualize semantic annotations without modifying the content, dynamic style sheets are used. Different types of borders with different colors are used to present RDFa annotated content which might be overlapping. To show the value of RDFa attributes which are not visible in normal text, CSS tooltips are used. To prevent altering content, tooltips are created on the fly each time the user moves the mouse pointer over annotated content. Each time a new annotation is added by RDFa DOM manipulator, this component is called to visualize the editor.

**RDF Triple Browser and Editor.** This component extracts the RDF triples embedded in the text and provides the edit and delete functionality for these triples. This component is in a mutual relation to rich text editor and is dynamically updated when a new annotation is added to the text (also the text editor is updated when a triple is modified here). When user edits or deletes a triple, these changes are delivered to RDFa DOM manipulator to update the content correspondingly.

**Online Resource Suggester.** This component provides the user with a set of accessible online resources. In order to perform this task, it accesses a number of external Web APIs (a detailed explainable follows below). The Online Resource Suggester works in a close relation to Annotator UI. It facilitates the task of annotating content by searching the terms which are selected by user and suggesting corresponding URIs.

**RDFa Proxy for Enricher APIs.** This component acts as a proxy to make the output of enricher APIs (i.e. NLP text annotation services) consumable as RDFa. Most of the current text enricher APIs do not provide any RDFa output. Therefore, we need to convert their generated output into RDFa. To do this, the RDFa proxy first sends the content to an external semantic text enrichment service. The output of the service is then converted to a standard format which includes label, URI, type, positions and properties related to the extracted entities. Then a mapping to a desired vocabulary is performed in order to make appropriate annotations. These annotations are delivered to the RDFa DOM manipulator to update the content correspondingly. In case an URI is needed for an entity, the online resource suggester is used to assign an URI to the entity.

All the former components use Javascript utility libraries like jQuery\(^2\) and RDFQuery\(^3\) to implement their functions. To facilitate semantic annotation of

---

\(^2\) [http://jquery.com](http://jquery.com)

\(^3\) [http://code.google.com/p/rdfquery/](http://code.google.com/p/rdfquery/)
Fig. 3. The main RDFaCE WYSIWIM view (left) with the RDF Triple Browser and Editor (right).

content, RDFaCE also uses a number of external Web APIs. Online APIs are invoked to carry out the following functions:

- **RDF Namespace Lookup**: In order to avoid that users have to type complete URIs, common namespace prefixes can be used everywhere in RDFaCE. These are looked-up using the prefix.cc service. Furthermore, in case users want to add a new property for which they do not even know an appropriate vocabulary, RDFaCE can look-up an appropriate vocabulary and property resource using the Swoogle service.

- **Online Resource Locating**: Finding an appropriate URI for the resources which are selected by users can facilitate annotation process to a good extend. When users select a part of the text and want to create a statement about the respective entity, the online resource locator will do a Sindice search to find suitable resources that match with the users selected item.

- **Semantic Text Enrichment**: Starting to annotate a document from the scratch is very tedious and time consuming. There are already some Natural Language Processing (NLP) APIs available on the Web which extract specific entities and relations from the text. By using these APIs, we can provide a good starting point for further user annotations. Users then can modify and extend these automatically pre-annotated content. RDFaCE currently uses

the OpenCalais, Ontos, Alchemy, Extractiv and Evri NLP API\(^6\) to enrich the text.

3 Views for Semantic Text Authoring

The main innovation of RDFaCE is the support of different views on the semantically annotated content. RDFaCE supports four different views for semantic text authoring, which are shown in Figure 4 and explained in more detail in the sequel. The user can easily switch between these views and even use them in parallel. The views are synchronized so that applying changes in one of the views automatically updates other views.

**WYSIWYG View.** The What-You-See-Is-What-You-Get view is the classical interface for rich-text authoring and used by authors, journalists etc. WYSIWYG text authoring is meanwhile ubiquitous on the Web and part of most content creation and management workflows. Users authoring content are used to interact with a WYSIWYG views and there exists a wide variety of WYSIWYG editors and editing components, which can be used on the Web or offline.

---

In particular WYSIWYG text authoring is already part of Content Management Systems, Weblogs, Wikis, fora, product data management systems and online shops, just to mention a few.

**WYSIWYM View.** The What-You-See-Is-What-You-Mean view is an extension of the WYSIWYG view, which highlights named entities and other semantic information. The highlighting is realized with special CSS3 selectors for the RDFa annotations. They are thus easily configurable in terms of color borders, backgrounds etc. When pointing with the mouse on a highlighted annotation RDFaCE shows additional information concerning the particular annotation as a tooltip. RDFaCE also supports editing in the WYSIWIM view by letting a user select entities she wants to annotate and provisioning of respective annotation functionality either via the context menu or a specific form, which opens as an overly.

**RDF Triple View.** This view summarizes all the facts, which can be extracted from the annotated text. It provides a deeper semantic view when compared to WYSIWYM view. There might be some triples not visible in the WYSIWYM view (e.g. annotations hidden using the CSS `display:none` style) but visible in this view. Since the triple view reveals all the triples embedded in the text, it can be called as WYMIWYS (What-You-Mean-Is-What-You-See) view. The triple view is (as all other views) updateable, i.e. facts can be directly deleted, which results in the removal of the corresponding RDFa annotations. The triple view is useful for curators and to a lesser extend for the authors for quickly verifying that entities and facts were correctly annotated.

**Source Code View.** Finally, the source code view shows the HTML source of the article including the RDFa annotations. This view is primarily intended for software engineers supervising the publication workflow as well as knowledge engineers. Since all formatting and interactive functionality (e.g. tooltips) is integrated via dynamic linking of CSS3 stylesheets with special selectors for the RDFa annotations, the source code view is not spoiled with any markup related to the WYSIWIM visualization.

4 Combining NLP-API results

One of the main features supported by RDFaCE is combining the results of multiple NLP APIs. Using this approach, we can harness synergies arising from the combination of different approaches for automatic text annotation. Users can select a set of NLP APIs and determine how they want to combine the results. The combination can be performed based on the agreement between two or more of the involved APIs.

Figure 5 shows the annotation results of the 5 different NLP APIs Alchemy, Extractive, OpenCalais, Ontos and Evri for a sample text. On the left, a heatmap
visualization reflects the list of items recognized by each API. Black and dark green cells indicate cases which need disambiguation. Black cells indicate that there is a conflict between two or more APIs when recognizing the type of a common entity. In this case we have to investigate what the correct type is. Dark green cells indicate that an entity is recognized only by one API. In this case, the error probability is high and further investigation is required.

![Visualization of API results]

**Fig. 5.** Generated results of different NLP APIs for article #1.

We use *Precision*, *Recall* and *F-measure* \[8\] as metrics for evaluating the correctness of the recognized entities found by each API as well as combined APIs:

\[
Recall = \frac{\text{Correctly Recognized Entities}}{\text{Actual Entities in the Text}} \tag{1}
\]

\[
Precision = \frac{\text{Correctly Recognized Entities}}{\text{Entities Recognized by the API}} \tag{2}
\]

\[
F = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}
\]
To compare the results of the different APIs, 31 articles were collected in the three categories news articles, weblog posts and Wikipedia articles. For each article, the following analysis was performed:

We carefully analysed the text and manually annotated it by recognizing references to location, person and organization entities. As a result we obtained a list of actual entities together with their types. Then we used the RDFaCE enrichment feature to automatically annotate the text by employing the external NLP APIs. By analyzing the RDFaCE generated heatmaps (cf. Figure 5) and the disambiguation of recognized entities, we extracted the number of recognized entities, correctly recognized, wrongly recognized and missing entities. Based on these values, Recall (1), Precision (2) and F-Score (3) were calculated for each API as well as for various combinations of APIs. Table 1 presents the detailed results of these metrics for each API as well as for the situation when 2, 3 or 4 of the APIs agree on a recognized entity. The table also shows the average value of these metrics for each category of the analyzed texts as well as the average for all analyzed texts. In Figure 6 the results are visualized as bar charts.

The results show that Alchemy, OpenCalais, Ontos and Evri deliver comparable results, while Extractive is a little behind. The ranking with regard to F-Score for all individual categories as well as for the average over all categories is: 1. Evry, 2. OpenCalais, 3. Alchemy, 4. Ontos, 5. Extractive. That the ranking is the same for all categories as well as the overall average indicates that all services perform homogeneously across the different categories. Another interesting observation is that all services deliver the best F-Score for news articles followed by Wiki articles and blog posts. A plausible reason for this is the degree of formality and quality checks, which are more likely with news articles than with blog posts.

As we consider more agreement on an entity to be recognized (i.e. two, three or four APIs have to agree), we obtain a higher precision but lower recall (Figure 6). The interesting result of our analysis is that we have the highest F-score when two or more APIs agree on an entity. In this case, we also get the highest recall and the result is independent from the type of text (i.e. news, weblog or wiki article). Further increasing the requirement of agreement, however, dramatically decreases recall.

5 Use Cases

The RDFaCE approach is very versatile and can be applied in a vast number of use cases. Also, our implementation based on the widely used TinyMCE editor makes RDFaCE directly applicable in many usage scenarios. In this section we introduce three complementary use cases for RDFaCE which exemplify the versatility of the approach.

7 Available at http://rdface.aksw.org/samples/
5.1 rNews

rNews\(^8\) is a proposed standard for using RDFa to annotate HTML documents with news-specific metadata. rNews is proposed by International Press Telecommunications Council (IPTC)\(^9\), which is a consortium of the world’s major news agencies, publishers and industry vendors.

rNews defines a small set of core concepts for annotating news articles and a few properties for each concept. Concepts include NewsItem, Tag, Person, Article, Media, Headline, Location, Organization, Party, TickerSymbol and Comment. These annotations are derived from the best practices found in the news industry.

RDFaCE is well suited for the rNews vocabulary. It provides an auto-suggestion feature for the classes and properties defined in the rNews vocabulary. RDFaCE also provides a context menu for the rNews vocabulary so that users can easily annotate their news articles using the rNews vocabulary. Furthermore,
RDFaCE maps the output of different annotation APIs to the rNews vocabulary thereby providing a base set of automatically annotated content for journalists and content managers.

The adaption of RDFaCE for rNews are performed in a way, that the same strategies are easily applicable for similar domain specific annotation vocabularies. RDFaCE can be configured to employ an annotation vocabulary for auto suggestion, context menu and NLP API output mapping.

5.2 OntoWiki

OntoWiki\textsuperscript{10} is a tool that provides support for agile, distributed knowledge engineering scenarios. Ontowiki facilitates the visual presentation of a knowledge base as an information map, with different views on instance data. Furthermore, it enables intuitive authoring of semantic content, with an inline editing mode for editing RDF content. OntoWiki supports two complementary edit strategies for the knowledge base: a) *Inline editing* which enables users to edit the smallest possible information chunks (i.e. statements). b) *View editing*, which enables users to edit common combinations of information (such as an instance of a distinct class) in one single step. Ontowiki uses RDFAuthor\textsuperscript{9} to make generated RDFa views editable.

Since OntoWiki is other than other semantic Wikis based on the RDF data model in the first place (i.e. facts/triples are the main artifacts instead of annotated wiki texts), the semantic annotation of texts is not possible. By integrating RDFaCE into OntoWiki text literals containing rich text can now be enriched with semantic annotations. These annotations can subsequently automatically extracted and explicitly represented in the knowledge base being edited with OntoWiki. In addition to the form-based editing of information, RDFaCE provides a WYSIWYM editor component to OntoWiki for selecting and editing rich text including RDFa annotations. Thus, RDFaCE contributes significantly

\textsuperscript{10} http://ontowiki.net
to OntoWiki’s aim of facilitating a pay-as-you-go knowledge engineering strategy: users can firstly add textual content, annotated this textual content in a second step and later on explicitly materialize the facts, which can be extracted from the annotations and thus attain a rich semantic representation.

5.3 Wordpress

Wordpress is an open source Weblog tool and publishing platform. Wordpress is often customized into a Content Management System (CMS) and is used by over 14% of the 1,000,000 biggest websites (54.4% of CMS market share). Wordpress uses TinyMCE as its content editor. That makes it extremely easy to add the RDFaCE plugin for semantic content authoring within this CMS (cf. Figure 7). With the integration of RDFaCE into the Wordpress, the availability of semantically annotated content on the Web can be substantially increased. The semantically annotated content can be indexed by the new generation of search engines, will result in more accurate search results and facilitate the knowledge extraction from the Web in general.

6 Related Work

Semantic authoring aims to facilitate the composition of intelligent content (i.e. content with explicit semantic structure) based on a set of shared vocabularies. Semantic content is created by annotating the content with appropriate vocabularies. With the emergence of the Web of Data, some efforts have been
undertaken to blur the border between authoring and annotation of content as much as possible. *Semantic Wikis* such as, for example, Semantic MediaWiki\(^6\) represent an approach that combines traditional wiki systems with semantic annotation. They provide a collaborative environment for semantic content authoring. Microformats\(^12\) RDFa\(^13\) and most recently Microdata\(^14\) are existing options used by semantic editors to embed the semantic annotations within the Web content. Among them, RDFa as a W3C recommendation is the most expressive and versatile option.

There are already a few RDFa editors available. *RADiFy*\(^15\) is a bookmarklet for annotating a web page with RDFa. As a bookmarklet, it provides a side bar to view, edit and load ontologies. RADiFy provides a list of predefined predicates that can be extended by loading new ontologies. For a user familiar with ontologies and semantic web concepts, it seems to be a useful and simple tool to annotate web pages. However, RADiFy has the following drawbacks: There is no editing option to edit the created triples. Also, the annotation mechanism of simply adding `<span>` tags is not very efficient and in some cases erroneous. For instance, if a link is selected, it adds a `<span>` tag inside the `<a>` tag. In addition, RADiFy does not support the use of the "rel" attribute but uses the "property" attribute for all predicates without considering their values and data types.

*WYMeditor*\(^16\) is a web-based XHTML editor with support for RDFa. WYMeditor's main concept is to leave details of the document's visual layout, and to concentrate on its structure and meaning, while trying to give the user as much comfort as possible (similar as WYSIWYG editors). The emphasis on strict XHTML compliance, makes it a good option to add RDFa annotations but its RDFa editor does not include many features. It considers blocks as the unit of annotation which makes it difficult to annotate a part of text within a block. It also uses different background colors to distinguish different annotations which can be problematic in cases when there are too many overlapping annotations. Furthermore, editing and deleting annotations in a WYSIWYG way is not supported.

*Loomp*\(^17\) is a tool representing a prove-of-concept for the One Click Annotation (OCA) strategy. OCA is a concept addressing the issue of creating semantic content by non-expert users\(^8\). The main difference between Loomp and RDFaCE is that Loomp relies on the functionality of a server managing the semantic content while RDFaCE provides client-side annotation for modifying RDFa content directly. Moreover, Loomp uses a triple store on the server side but in RDFaCE, triples are created on the fly in the user browser. At the time

\(^{12}\) [http://microformats.org](http://microformats.org)

\(^{13}\) [http://www.w3.org/TR/rdfa-syntax/](http://www.w3.org/TR/rdfa-syntax/)

\(^{14}\) [http://www.w3.org/TR/microdata/](http://www.w3.org/TR/microdata/)

\(^{15}\) [http://duncangrant.co.uk/radify/](http://duncangrant.co.uk/radify/)

\(^{16}\) [http://www.wymeditor.org](http://www.wymeditor.org)

\(^{17}\) [http://loomp.org](http://loomp.org)
of writing, we were not able to find any demo of Loomp to further investigate its implemented features.

*RDFauthor*[^18] is another related tool that aims at editing RDFa contents. The RDFauthor approach is based on the idea of making arbitrary XHTML views with integrated RDFa annotations editable[^9]. RDFauthor converts an RDFa-annotated view directly into an editable form thereby hiding the RDF and related ontology data models from novice users. Although RDFauthor has as RDFaCE the goal to make RDFa editing simple by abstracting the details of RDFa authoring both differ in two crucial aspects: Firstly, RDFauthor assumes that the RDFa content is already existing while RDFaCE provides the feature to creating new RDFa annotations. Secondly, instead of using forms to edit RDFa contents, RDFaCE employs inline editing of contents by providing a rich semantic text editor.

*OntosFeeder*[^19] and *Epiphany*[^20] are two partially related tools. Epiphany is a service that annotates web pages automatically with Linked Data by creating an RDFa enhanced versions of the input HTML page[^1]. OntosFeeder[^5] is also a TinyMCE plugin which uses the Ontos Web API to provide context information (in RDFa) to be integrated into content management systems. These tools use the same annotation mechanism as RADiFY with the above mentioned issues. Furthermore, they do not provide editing functionality for RDFa generated content. They can be used as complementary tools to RDFaCE which deliver a set of initial RDFa annotations to be edited and extended later on by RDFaCE.

7 Conclusions and Future Work

The user friendly authoring of semantically enriched content is a crucial aspect for the ultimate success of semantic technologies. With RDFaCE we presented in this article an approach and its implementation of a WYSIWYM (What You See Is What You Mean) editor based on complementing the classical WYSIWYG view with three additional views on the semantic representations. We showed that with RDFaCE the semantic annotation and enrichment can be easily integrated into the content authoring pipelines commonly found in many content centric scenarios.

We see the work presented in this article as an initial step in a larger research agenda aiming at simplifying the authoring and annotation of semantically enriched textual content. Regarding future work we envision to extend the WISI-WYM concept towards different modalities, such that the annotation of images and multimedia object is supported. With regard to the NLP functionality made available through RDFaCE we aim at extending the current implementation towards supporting relationship and keyword extraction.

[^18]: http://aksw.org/Projects/RDFauthor
[^19]: http://wordpress.org/extend/plugins/ontos-feeder/
[^20]: http://projects.dfki.uni-kl.de/epiphany/
Acknowledgments

We would like to thank our colleagues from AKSW research group for their helpful comments and inspiring discussions during the development of RDFaCE. This work was supported by a grant from the European Union’s 7th Framework Programme provided for the project LOD2 (GA no. 257943).

References