The RDFa Content Editor - From WYSIWYG to WYSIWYM

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. 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 life-cycle. For bootstrapping the semantic annotation process we integrate five automatic 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;

I. 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. Furthermore, we performed an experimental evaluation with human users to further assess the usability of RDFaCE.

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.

4) An evaluation of the RDFaCE content authoring environment using a sizable user group and measuring subjective as well as objective usage characteristics.

The remainder of this article is structured as follows: In Section II and III we describe the background of our work and discuss the requirements targeted in RDFaCE system. Section IV describes the architecture of the system. Section V presents the different views for semantic text authoring. In Section VI the combination of different NLP APIs for bootstrapping the semantic annotation process is discussed. Section VII introduces two RDFaCE use cases. Section VIII reports on RDFaCE usability evaluation results. Related work is presented in Section IX and finally Section X concludes with an outlook on future work.

II. SEMANTIC CONTENT AUTHORING

In the context of this paper, we define semantic content authoring (SCA) as the tool-supported manual composition process aiming at the creation of documents which are:

- fully semantic in the sense that their original data model uses a semantic knowledge representation formalism (such as RDF, RDF-Schema or OWL) or
- based on a non-semantic representation form (e.g. text or hypertext), which is enriched with semantic representations during the authoring process.

An SCA user interface (UI) is a human accessible interface with capabilities for writing and modifying semantic documents. Semantic documents facilitate a number of important aspects of information management:

- For search and retrieval enriching documents with semantic representations helps to create more efficient and effective search interfaces, such as faceted search [1] or question answering [2]. Ultimately, users are empowered to counter the increasing information overload and gain better access to relevant documents and answers related to their information needs.

- In information presentation semantically enriched documents can be used to create more sophisticated ways of flexibly visualizing information, such as by means of semantic overlays as described in [3].

- For information integration semantically enriched documents can be used to provide unified views on heterogeneous data stored in different applications by creating composite applications such as semantic mashups [4].

- To realize personalization, semantic documents provide customized and context-specific information which better fits user needs and will result in delivering customized applications such as personalized semantic portals [5].

- For reusability and interoperability enriching documents with semantic representations (e.g. using the SKOS and Dublin Core vocabularies) facilitates exchanging content between disparate systems and enables building applications such as executable papers [6].

There are already many approaches and tools available for semantic content authoring which address different aspects of this task by proposing appropriate user interfaces. With regard to explicit semantic content authoring recent approaches can be roughly classified into the categories Top-Down and Bottom-Up. The classification is based on the starting point of the authoring process which can be ontologies (with upper level of expressiveness) or unstructured content (with lower level of expressiveness).

1) Bottom-Up Approaches: These approaches which are usually called semantic annotation techniques (a.k.a. semantic markup [7]) aim to annotate existing documents using a set of predefined ontologies. The basic ingredients of a semantic annotation system are ontologies, the documents and the annotations that link ontologies to documents [8]. The result of the annotation process is a document that is marked-up semantically. Microformats [9], RDFa [10] and Microdata [11] are existing markup technologies for content annotation.

2) Top-Down Approaches: These approaches which are also called Ontology Population [9] techniques aim to create semantic content based on a set of initial ontologies which are extended during the population process. When compared with the bottom-up approaches, these approaches deal with semantic representations from the beginning instead of lifting unstructured content to a semantic level.

These approaches combine ontological rigour with flexible user interface constructs to create semantic content. Semantic templates as discussed in [10], [11], [12] are one technique to realize this goal. In this approach each class of the ontology has an associated template. Each instance of a class is represented by a page using that template. Data properties are displayed as simple text while object properties are displayed as links to other pages (representing other instances of the ontology). Users can also edit the underlying ontology which will result in changes of the corresponding templates.

http://microformats.org
http://www.w3.org/TR/rdfa-syntax/
http://www.w3.org/TR/microdata/
In order to develop an effective and efficient SCA system, we aim to fulfill a set of quality attributes which represent the areas of concern from the viewpoint of its consumers. In [13], we introduced set of quality attributes, which are widely used in architecture development and assessment of SCA systems. Usability, customizability, proactivity, automation and scalability are the main attributes which we addressed with the development of RDFaCE.

Usability: Usability is a measure of the quality of a user’s experience in interacting with a system. In [14], usability is defined as consisting of the six factors:

1) **Fit for use (or functionality).** The system can support the tasks that the user has in real life.
2) **Ease of learning.** How easy is the system to learn for various groups of users?
3) **Task efficiency.** How efficient is it for the frequent user?
4) **Ease of remembering.** How easy is it to remember for the occasional user?
5) **Subjective satisfaction.** How satisfied is the user with the system?
6) **Understandability.** How easy is it to understand what the system does?

**Ease of use** (or user friendliness) is defined as the combination of factors 2 to 6.

Providing a **Single Point of Entry Interface** with support of **Inline Editing** of annotations improves the usability of an SCA system. It means the environment in which users annotate documents should be integrated with the one in which they create, read, share and edit them. So, there is no added user effort involved in creating a semantic content versus a conventional approach, because the real work is done by the software through capturing semantics that is already being provided by the user [15], [10], [8]. Inline editing allows editing items in one single step by clicking on them.

**Customizability:** Customizability is the ability of a system to be configured according to users’ needs and preferences. Instead of being a static form strictly dependent on a given schema, an SCA system should provide mechanism to tailor its functionalities based on the user needs [11]. In [10] the concept of “semantics in the eyes of the end-user” is introduced which means an SCA system should provide different views for different personas using the system.

Providing **Different Semantic Views** is a mechanism for improving the customizability of SCA systems: Semantic views allow the generation of different views on the same metadata schema and aggregations of the knowledge base based on the roles, personal preferences, and local policies of the intended users [10], [11], [6].

**Proactivity:** Proactivity is the ability of a system to act in advance of a future situation, rather than just reacting. It means taking control and making things happen rather than just adjusting to a situation or waiting for something to happen. An SCA system should provide users with pre-filled form fields, suggestions, default values etc. These facilities simplify the authoring process, as they reduce the number of actions users have to perform. Moreover, they reduce the possibility that users provide incomplete or empty metadata [11].

**Automation:** Automation is the ability of a system to automatically perform its intended tasks thereby reducing the need for human work. In the context of semantic authoring it means the provision of facilities for automatic mark-up of documents to facilitate the economical annotation of large document collections [8]. The automatic process of annotating is composed basically of finding terms in documents, mapping them against an ontology, and disambiguating common terms.

**Scalability:** Scalability refers to the capability of a system to maintain performance under an increased work load. An SCA system should support scalability as for example, the number of users, data or annotations increase. Support of caching and implementing a suitable storage strategy play an important role in achieving an scalable SCA system [10], [8], [17], [18].

IV. RDFaCE SYSTEM 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[^7]. 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 [19]. 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 provide user friendly 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 expedite the creation of new triples. For instance users can use the context menu and select from a list of predefined annotations.

[^7]: http://tinymce.moxiecode.com

---

**III. REQUIREMENTS FOR A SEMANTIC CONTENT AUTHORING SYSTEM**

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[^7]. 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 [19]. 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 provide user friendly 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 expedite the creation of new triples. For instance users can use the context menu and select from a list of predefined annotations.

[^7]: http://tinymce.moxiecode.com
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>` 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 facil-
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[8] and RDFQuery[9] to implement their functions. To facilitate semantic annotation of 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 APIs[11] to enrich the text.

V. 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 3 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 WYSIWYM 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

---

**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 WYSIWYM visualization.

**VI. 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 4 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.

We use Precision, Recall and F-measure [20] as metrics for evaluating the correctness of the recognized entities found by each API as well as combined APIs:

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

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

\[
F = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)
\]
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 4) 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. Results of calculating these metrics for each API as well as for the situation when 2, 3 or 4 of the APIs agree on a recognized entity are available at http://rdface.aksw.org/samples/results.html.

The results (cf. Figure 5) 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. Evri. 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. 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.

VII. USE CASES OF RDFaCE

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 two complementary use cases for RDFaCE which exemplify the versatility of the approach.

rNews: rNews is a proposed standard for using RDFa to annotate HTML documents with news-specific metadata. rNews is proposed by International Press Telecommunications Council (IPTC), 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.

12 Available at http://rdface.aksw.org/samples/

13 http://dev.iptc.org/rNews
Wordpress: Wordpress\footnote{http://wordpress.org/} 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)\footnote{Available at: http://wordpress.org/extend/plugins/rdfa/}. Wordpress uses TinyMCE as its content editor. That makes it extremely easy to add the RDFaCE plugin\footnote{Available at: http://wordpress.org/extend/plugins/rdfa/} for semantic content authoring within this CMS. With the integration of RDFaCE into the Wordpress, the availability of semantically annotated content on the Web can be substantially increased.

VIII. Usability Evaluation

Since releasing RDFaCE, the tool has been downloaded over 300 times and the online demo page has received more than 1000 unique visits. we also could collect considerable feedbacks from RDFaCE end-users on the Social Web. For a concrete evaluation of RDFaCE usability, we conducted an experiment with 16 participants of ISSLOD 2011 summer school\footnote{Summer school on Linked Data: http://lod2.eu/Article/ISSLOD2011}. For the experiment, we developed a usability test platform\footnote{Available online at: http://rdface.aksw.org/usability}. The experiment consisted of the following steps: First, some basic information about semantic content authoring along with a demo showcasing different RDFaCE features was presented to the participants as a video. Then, participants were asked to use RDFaCE to annotate three text snippets – a wiki article, a blog post and a news article (News#2, Blog#4, Wiki#3 from our sample articles). For each text snippet, a timeslot of five minutes was available to use different features of RDFaCE for annotating occurrences of persons, locations and organizations with suitable entity references (i.e. Linked Data URIs). Subsequently, a survey was presented to the participants were they were asked some questions about their experience while working with RDFaCE. Questions were targeting six factors of usability\footnote{http://projects.dfki.uni-kl.de/epiphany/} namely Fit for use, Ease of learning, Task efficiency, Ease of remembering, Subjective satisfaction and Understandability. Results of individual user annotations as well as the results of the survey were carefully analyzed for extracting subjective and objective usage characteristic of RDFaCE, respectively. In the following we report about the result of this experiment:

Participants. Participants included students (85\%) and researchers (15\%) working on different aspects of computer science and informations systems. As shown in Table I they bear different level of knowledge in Semantic Web and in particular RDFa, varying from basic to expert knowledge.

Usability Factors. During the experiment we collected considerable qualitative feedbacks from the end-users. As shown in Table II the overall feedback of users was positive and they provided constructive feedback to enhance the usability of RDFaCE. They frequently told us that they are impressed with the functionality of RDFaCE to support their desired tasks. They found the UI easy to learn but in some cases had difficulties to distinguish between property and subject suggestions. Some users suggested to change triple insertion to property insertion and some suggested to improve the visualization of URI suggestion results so that they can easily choose the appropriate one. Most of the users found the UI easy to remember and a few suggested to change some RDFaCE toolbar icons to more descriptive ones.

Annotations. Figure 6 reflects the number of annotations (triples) as well as the time of annotation per user for each of the text fragments. From the results we can see that almost (except two cases) all users have been able to create semantic text content. The annotation time for the last text snippet has decreased for most of the users which is an indicator for increased familiarity of the users with RDFaCE.

IX. Related Work

There are already many SCA systems available. RDA-iF\footnote{http://dh cancgrant.co.uk/radify/} WYMeditor\footnote{http://www.wymeditor.org/} DataPress\footnote{http://wordpress.org/extend/plugins/ontos-feeder/}, Loomp\footnote{http://wordpress.org/extend/plugins/ontos-feeder/} and FLERSA\footnote{http://www.etri.re.kr/FLERSA/} are some examples of SCA systems which adopt the bottom-up approach. We can also mention RDFauthor\footnote{http://rdfauthor.deri.ie/} and SAHA 3\footnote{http://projects.dfki.uni-kl.de/epiphany/} as two examples which adopt the top-down approach for semantic authoring. OntosFeeder\footnote{http://projects.dfki.uni-kl.de/epiphany/} and Epiphany\footnote{http://projects.dfki.uni-kl.de/epiphany/} are also two partially related tools. They do not provide editing functionality for RDFa generated content but 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. As an another related work we can mention NERD\footnote{http://projects.dfki.uni-kl.de/epiphany/} which is an evaluation framework which records and analyzes ratings of Named Entity extraction and disambiguation tools. The main difference between RDFaCE and NERD is that RDFaCE employs the voting approach to combine the results of NLP APIs for

<table>
<thead>
<tr>
<th>Skill/ Level heard of it</th>
<th>basic</th>
<th>advanced</th>
<th>expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill in Semantic Web</td>
<td>18.75%</td>
<td>37.50%</td>
<td>37.50%</td>
</tr>
<tr>
<td>Skill in RDFa</td>
<td>18.75%</td>
<td>37.50%</td>
<td>37.50%</td>
</tr>
</tbody>
</table>

Table I

<table>
<thead>
<tr>
<th>Usability Factor/Grade</th>
<th>Poor</th>
<th>Fair</th>
<th>Neutral</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fit for use</td>
<td>0%</td>
<td>12.50%</td>
<td>31.25%</td>
<td>43.75%</td>
<td>12.50%</td>
</tr>
<tr>
<td>Ease of learning</td>
<td>0%</td>
<td>12.50%</td>
<td>50%</td>
<td>31.25%</td>
<td>6.25%</td>
</tr>
<tr>
<td>Task efficiency</td>
<td>0%</td>
<td>0%</td>
<td>56.25%</td>
<td>37.50%</td>
<td>6.25%</td>
</tr>
<tr>
<td>Ease of remembering</td>
<td>0%</td>
<td>0%</td>
<td>37.50%</td>
<td>50%</td>
<td>12.50%</td>
</tr>
<tr>
<td>Subjective satisfaction</td>
<td>0%</td>
<td>18.75%</td>
<td>50%</td>
<td>25%</td>
<td>6.25%</td>
</tr>
<tr>
<td>Understandability</td>
<td>6.25%</td>
<td>18.75%</td>
<td>31.25%</td>
<td>37.50%</td>
<td>6.25%</td>
</tr>
</tbody>
</table>

Table II

Usability Factors Derived from the Survey.
automatic annotation but NERD expects a human being to manually compare the results of different NLP APIs and choose the right one for annotation.

![Figure 6](image_url) Results of usability test. (top) Number of annotations per user. (bottom) Annotation time per user.

**Figure 7** provides a comparison between the three popular SCA systems (RDFauthor, SAHA 3 and Loomp) and RDFaCE based on the quality attributes discussed in section III. RDFauthor is a tool for editing RDFa contents. The RDFauthor approach is based on the idea of making arbitrary XHTML views with integrated RDFa annotations editable [23]. RDFauthor converts an RDFa-annotated view directly into an editable form thereby hiding the RDF and related ontology data models from novice users. It is backend independent to some extend and supports two different types of storage engines. Although RDFauthor has as RDFaCE the goal to make RDFa editing simple by abstracting the details of RDF 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. Saha 3 is another meta data editor which is very similar to RDFauthor but supports real-time validation in addition.

Loomp is a tool representing a prove-of-concept for the *One Click Annotation* (OCA) strategy. The Web-based OCA editor allows for annotating words and phrases with references to ontology concepts and for creating relationships between annotated phrases. It supports a faceted viewing feature which highlights user-selected annotations in the Web browser. 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.

The main advantages of RDFaCE comparing to other tools are twofold: Providing different views for authoring semantic documents as well as supporting automatic content annotation which improve the customizability and automation remarkably. Furthermore, since RDFaCE processes the annotations client-side within the user’s browser and does not require any central storage backend, it is highly scalable.

**X. 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 WYSIWYM concept towards different modalities, such that the annotation of images and multimedia object is supported. With regard to the NLP functionality made available through RDoCE we aim at extending the current implementation towards supporting relationship and keyword extraction.

REFERENCES


