Proceedings of the Fourth Workshop on Semantic Wikis – The Semantic Wiki Web

6th European Semantic Web Conference Hersonissos, Crete, Greece, June 2009

edited by Christoph Lange

June 1, 2009

Preface

Dear Reader,

after a successful year, the semantic wiki community is meeting again to have their 4th workshop. A lot has happened after the previous workshop: People have invested further research and development into existing systems to make them more mature: Three out of 15 presentations in this workshop are about evolutions of systems presented in 2008. The community has further grown and become aware of new application areas, mainly thanks to the Ontolog teleconference series on semantic wikis[1]. Another indication of maturity is that, while research is still as creative and dynamic as ever, implementation of semantic wiki technologies largely concentrates on few successful platforms that are extensible by plugins, most notably (Semantic) MediaWiki and KiWi (8 out of 15 talks in this workshop). And these technologies are not necessarily limited to wikis. Using a wiki is not an end in itself, but, depending on the application, any semantic social software will get the task done: Systems that are no longer called "wikis" have adopted key principles of wikis, such as ubiquitous interlinking and easy collaboration. Thus, the semantic wiki community is giving their findings back into the larger Semantic Web. In 2006, when our first workshop took place, many semantic web researchers jumped onto the semantic wiki bandwagon, as they were hip. Now, semantic wikis rather serve as incubators for testing and evaluating new approaches and technologies in a manageable setting, before releasing them to the Semantic Web.

We wish to thank *all* authors and reviewers who spent their nights and days contributing to this topic and thereby made this workshop possible. Many thanks also to the ESWC organisation team, which set the stage for this workshop as one out of 8. Let us continue to bring the lively wiki spirit to the Semantic Web and enjoy reading the proceedings.

Bremen, June 2009
Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli and Max Völkel

[1] `http://ontolog.cim3.net/cgi-bin/wiki.pl?SemanticWiki`

Contents

Programme

09:00 – 10:30 Session 1: Lightning panels

09:00 – 09:15 Opening Ceremony
Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel

Lightning panels: a group of short talks on related topics,
≈ 10 min. per full paper, 5 min. per short paper,
followed by a joint discussion (5 min. per paper)

09:15 – 09:45 **Language Processing**
How Controlled English can Improve Semantic Wikis
(Best Paper)
Tobias Kuhn

Information Extraction in Semantic Wikis
Pavel Smrz and Marek Schmidt

—discussion—

09:45 – 10:30 **Wiki Architectures**
Undo in Peer-to-peer Semantic Wikis
Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli

Enabling cross-wikis integration by extending the SIOC ontology
Fabrizio Orlandi and Alexandre Passant

What the User Interacts With: Reflections on Conceptual Models for Semantic Wikis
François Bry, Michael Eckert, Jakub Kotowski, and Klara Weiand

—discussion—

10:30 – 11:00 **Coffee Break**

11:00 – 13:00 Session 2: Lightning Panels

11:00 – 11:30 **Knowledge Representation and Reasoning**

Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis
Rolf Sint, Stephanie Stroka, Sebastian Schaffert and Roland Ferstl

WIKITAAABLE: A semantic wiki as a blackboard for a textual case-base reasoning system
Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli and Yannick Toussaint

Engineering on the Knowledge Formalization Continuum
Joachim Baumeister, Jochen Reutelshöfer, and Frank Puppe

—discussion—

11:30 – 11:55 **Applications**

MoKi: the Modelling wiKi
Marco Rospocher, Chiara Ghidini, Viktoria Pammer, Luciano Serafini, and Stefanie Lindstaedt

Brede Wiki: Neuroscience data structured in a wiki
Finn Årup Nielsen

—discussion—

11:55 – 12:15 **Social Software**

Metasocial Wiki – Towards an interlinked knowledge in a decentralized social space
Amparo E. Cano, Matthew Rowe, and Fabio Ciravegna

Analysis of Tag-Based Recommendation Performance for a Semantic Wiki
Frederico Durão and Peter Dolog

—discussion—

12:15 – 12:50 **Platforms and Plugins**

An Extensible Semantic Wiki Architecture
Jochen Reutelshöfer, Fabian Haupt, Florian Lemmerich, and Joachim Baumeister

KiWi – A Platform for Semantic Social Software
Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint and Stephanie Stroka

VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis
Mariano Rico, David Camacho and Oscar Corcho

—discussion—

12:50 – 13:00 Wrapping up the talks, preparing the Demo Session

13:00 – 14:30 **Lunch**

14:30 – 15:10 Session 3: Keynote

Semantic Wikis for Software Knowledge Management
Josef Holy, Sun, Prague

—discussion—

15:10 – 16:00 Session 4: Demo Session

(demos listed in the order of the morning talks)
AceWiki: Controlled English
Tobias Kuhn

Information Extraction in KiWi
Pavel Smrz and Marek Schmidt

Peer-to-peer Undo in Swooki
Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli

SIOC-MediaWiki Exporter
Fabrizio Orlandi and Alexandre Passant

Case-based Reasoning in WIKITAAABLE
Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli and Yannick Toussaint

MoKi: the Modelling wiKi
Marco Rospocher, Chiara Ghidini, Viktoria Pammer, Luciano Serafini, and Stefanie Lindstaedt

Brede Wiki: Neuroscience data structured in a wiki
Finn Årup Nielsen

Tag-Based Recommendation in KiWi
Frederico Durão and Peter Dolog

The KnowWE Architecture
Jochen Reutelshöfer, Fabian Haupt, Florian Lemmerich, and Joachim Baumeister

KiWi – A Platform for Semantic Social Software
Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint and Stephanie Stroka

VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis
Mariano Rico, David Camacho and Oscar Corcho

16:00 – 16:30 **Coffee Break (poster/demo session open to visitors)**

16:30 – 18:00 Session 5: Interactive

16:30 – 16:50	Problem Presentation
16:50 – 17:35	Teamwork
17:35 – 17:55	Presentation of results
17:55 – 18:00	Concluding Remarks

18:30 – 19:30

LarKC Reception

20:45 SemWiki 2009 Social Event

Organisation

- Christoph Lange
- Sebastian Schaffert
- Hala Skaf-Molli
- Max Völkel

How Controlled English can Improve Semantic Wikis

Tobias Kuhn

Department of Informatics & Institute of Computational Linguistics,
University of Zurich, Switzerland
tkuhn@ifi.uzh.ch
http://www.ifi.uzh.ch/cl/tkuhn

Abstract. The motivation of semantic wikis is to make acquisition, maintenance, and mining of formal knowledge simpler, faster, and more flexible. However, most existing semantic wikis have a very technical interface and are restricted to a relatively low level of expressivity. In this paper, we explain how AceWiki uses controlled English — concretely Attempto Controlled English (ACE) — to provide a natural and intuitive interface while supporting a high degree of expressivity. We introduce recent improvements of the AceWiki system and user studies that indicate that AceWiki is usable and useful.

1 Introduction

We present an approach how semantic wikis and controlled natural language can be brought together. This section gives a short introduction into the fields of semantic wikis and controlled natural languages.

1.1 Semantic Wikis

Semantic wikis are a relatively new field of research that started in 2004 when semantic wikis were introduced in [16] describing the PlatypusWiki system. During the last years, an active community emerged and many new semantic wiki systems were presented. Semantic wikis combine the philosophy of wikis (i.e. quick and easy editing of textual content in a collaborative way over the Web) with the concepts and techniques of the Semantic Web (i.e. enriching the data on the Web with well-defined meaning). The idea is to manage formal knowledge representations within a wiki environment.

Generally, two types of semantic wikis can be distinguished: On the one hand, there are text-centered approaches that enrich classical wiki environments with semantic annotations. On the other hand, logic-centered approaches use semantic wikis as a form of online ontology editors. Semantic MediaWiki [7], IkeWiki [12], SweetWiki [4], and HyperDEWiki [13] are examples of text-centered semantic wikis, whereas OntoWiki [1] and myOntology [15] are two examples of logic-centered semantic wikis. Web-Protégé [17] is the Web version of the popular

Protégé ontology editor and can be seen as another example of a logic-centered semantic wiki, even though its developers do not call it a "semantic wiki". In general, there are many new web applications that do not call themselves "semantic wikis" but exhibit many of their characteristic properties. Freebase[1], Knoodl[2], SWIRRL[3], and Twine[4] are some examples.

Semantic wikis seem to be a very promising approach to get the domain experts better involved in the creation and maintenance of ontologies. Semantic wikis could increase the number and quality of available ontologies which is an important step into the direction of making the Semantic Web a reality. However, we see two major problems with the existing semantic wikis. First, most of them have a very technical interface that is hard to understand and use for untrained persons, especially for those who have no particular background in formal knowledge representation. Second, existing semantic wikis support only a relatively low degree of expressivity — mostly just "subject predicate object"-structures — and do not allow the users to assert complex axioms. These two shortcomings have to be overcome to enable average domain experts to manage complex ontologies through semantic wiki interfaces.

In this paper, we will argue for using controlled natural language within semantic wikis. The Wiki@nt[5] system follows a similar approach. It uses controlled natural language (concretely Rabbit and ACE) for verbalizing OWL axioms. In contrast to our approach, users cannot create or edit the controlled natural language sentences directly but only the underlying OWL axioms in a common formal notation. Furthermore, no reasoning takes place in Wiki@nt.

1.2 Attempto Controlled English

Controlled natural languages are subsets of natural languages that are controlled (in both syntax and semantics) in a way that removes or reduces the ambiguity of the language. Recently, several controlled natural languages have been proposed for the Semantic Web [14]. The idea is to represent formal statements in a way that looks like natural language in order to make them better understandable to people with no background in formal methods.

Attempto Controlled English (ACE)[6] is a controlled subset of English. While looking like natural English, it can be translated automatically and unambiguously into logic. Thus, every ACE text has a single and well-defined formal meaning. A subset of ACE has been used as a natural language front-end to OWL with a bidirectional mapping from ACE to OWL [6]. This mapping covers all of OWL 2 except data properties and some very complex class expressions.

ACE supports a wide range of natural language constructs: nouns (e.g. "country"), proper names ("Zurich"), verbs ("contains"), adjectives ("rich"), singu-

[1] see [3] and `http://www.freebase.com`
[2] `http://knoodl.com`
[3] `http://www.swirrl.com`
[4] `http://www.twine.com`
[5] see [2] and `http://tw.rpi.edu/dev/cnl/`
[6] see [5] and `http://attempto.ifi.uzh.ch`

lar and plural noun phrases ("a person", "some persons"), active and passive voice ("owns", "is owned by"), pronouns ("she", "who", "something"), relative phrases ("who is rich", "that is owned by John"), conjunction and disjunction ("and", "or"), existential and universal quantifiers ("a", "every"), negation ("no", "does not"), cardinality restrictions ("at most 3 persons"), anaphoric references ("the country", "she"), questions ("what borders Switzerland?"), and much more. Using these elements, one can state ACE sentences like for example

Every person who writes a book is an author.

that can be translated into its logical representation:

$$\forall A \forall B(person(A) \wedge write(A, B) \wedge book(B) \rightarrow author(A))$$

In the functional-style syntax of OWL, the same statement would have to be expressed as follows:

```
SubClassOf(
  IntersectionOf(
    Class(:person)
    SomeValuesFrom(
      ObjectProperty(:write)
      Class(:book)
    )
  )
  Class(:author)
)
```

This example shows the advantage of controlled natural languages like ACE over other logic languages. While the latter two statements require a considerable learning effort to be understood, the statement in ACE is very easy to grasp even for a completely untrained reader. We could show in an experiment that untrained users (who have no particular background in knowledge representation or computational linguistics) are able to understand ACE sentences very well and within very short time [10].

2 AceWiki

We developed AceWiki that is a logic-centered semantic wiki that tries to solve the identified problems of existing semantic wikis by using a subset of ACE as its knowledge representation language. The goal of AceWiki is to show that semantic wikis can be more natural and at the same time more expressive than existing systems. Figure 1 shows a screenshot of the AceWiki interface. The general approach is to provide a simple and natural interface that hides all technical details.

AceWiki has been introduced in [8] and [9]. Since then, many new features have been implemented: support for transitive adjectives, abbreviations for proper names, passive voice for transitive verbs, support for comments, client-side OWL export, a completely redesigned lexical editor, and proper persistent

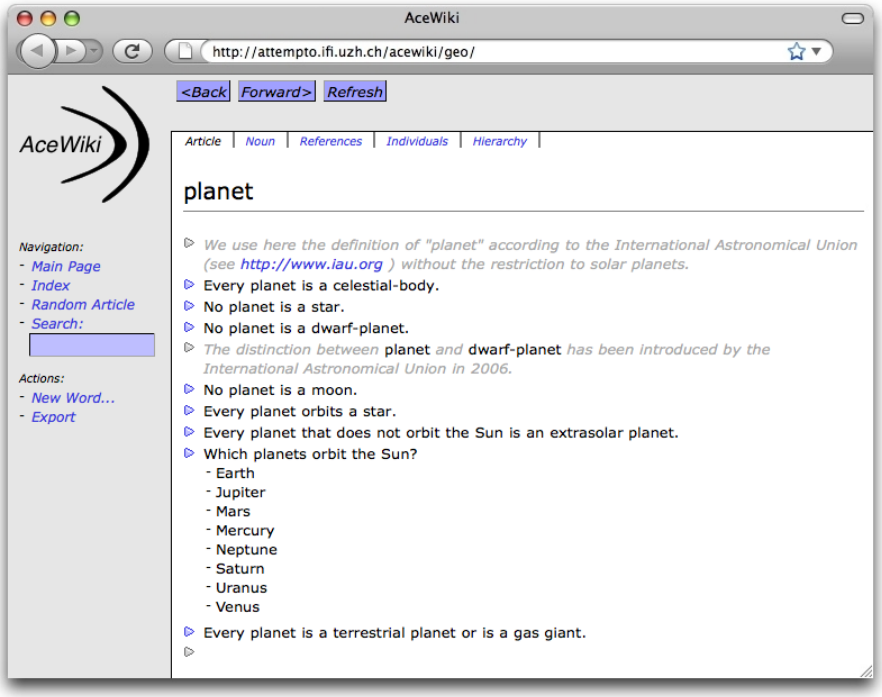


Fig. 1. This screenshot of the AceWiki interface shows an article about planets. Articles in AceWiki consist of declarative ACE sentences and ACE questions (both in black color) and of unrestricted natural language comments (in gray color).

storage of the wiki data. There is a public demo available[7] and the source code of AceWiki can be downloaded under an open source license. However, AceWiki has not yet reached the stage where it could be used for real-world applications. Crucial (but scientifically not so interesting) parts are missing: history/undo facility, user management, and ontology import.

One of the most interesting new features in AceWiki is the support for comments in unrestricted natural language, as it can be seen in Figure 1. Since it is unrealistic that all available information about a certain topic can be represented in a formal way, such comments can complement the formal ACE sentences. The comments can contain internal and external links, much like the text in traditional non-semantic wikis.

In order to enable the easy creation of ACE sentences, users are supported by an intelligent predictive text editor [11] that is able to look ahead and to show the possible words and phrases to continue the sentence. Figure 2 shows a screenshot of this editor.

[7] see `http://attempto.ifi.uzh.ch/acewiki`

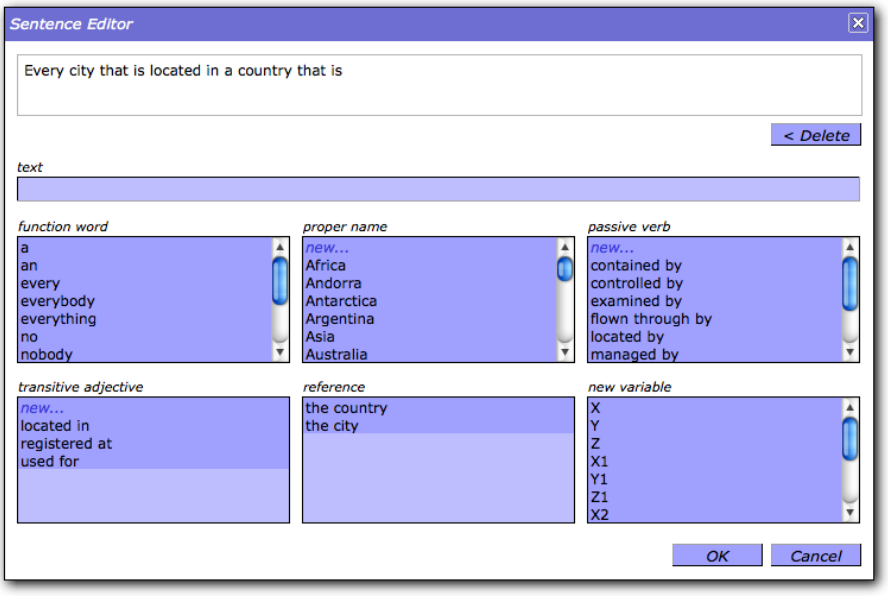


Fig. 2. This is the predictive editor of AceWiki. The partial sentence "every city that is located in a country that is ..." has already been entered and now the possible continuations are shown. In this way, the users can conveniently create syntactically correct sentences without learning the language in advance.

AceWiki supports an expressive subset of ACE. Some examples of sentences that can be created in AceWiki are shown here:

- Every country that borders no sea is a landlocked-country.
- Switzerland borders exactly 5 countries.
- No city that is located in Europe is controlled by the USA.
- If X borders Y then Y borders X.
- Every moon orbits a planet or orbits a dwarf-planet.

AceWiki is designed to seamlessly integrate a reasoner that can give feedback to the users, ensures the consistency of the ontology, can show its semantic structure, and answers queries. At the moment, we are using the OWL reasoner Pellet[8] and apply the ACE-to-OWL translator that is described in [6]. However, AceWiki is not restricted to OWL and another reasoner or rule engine might be used in the future.

The subset of ACE that is used in AceWiki is more expressive than OWL, and thus the users can assert statements that have no OWL representation. Because we are using an OWL reasoner at the moment, such statements are not

[8] `http://clarkparsia.com/pellet/`

considered for reasoning. In order to make this clear to the users, the sentences that are outside of OWL are marked by a red triangle:

- No ocean borders every continent.
- Every person that has a car owns the car or leases the car.
- If Berlin is a capital then Germany is a stable country.
- Every trip that starts at X and that ends at X is a round trip.

The most important task of the reasoner is to check consistency because only consistent ontologies enable to calculate logical entailments. In previous work [9], we explain how consistency is ensured in AceWiki by incrementally checking every new sentence that is added.

Not only asserted but also inferred knowledge can be represented in ACE. At the moment, AceWiki shows inferred class hierarchies and class memberships. The hierarchy for the noun "country", for example, could look as follows:

Upward

- Every country is an area.
- Every country is an object.

Downward

- Every alpine country is a country.
- Every baltic-state is a country.

Furthermore, ACE questions can be formulated within the articles. Such questions are evaluated by the reasoner and the results are listed directly after the question:

- Which cities are located in a country that borders Switzerland?
  - Berlin
  - Milano
  - Paris
  - Rome
  - Vienna
- What is Switzerland?
  - an alpine country
  - an area
  - a country
  - a landlocked-country
  - an object

If the question asks for a certain individual (represented in ACE by proper names) then the named classes (represented by nouns) of the individual are shown as the answer. In the cases where the question asks for a class (represented by a noun phrase), the individuals that belong to this class are shown as the answer.

Thus, AceWiki uses ACE in different ways: as an expressive knowledge representation language for asserted knowledge, to display entailed knowledge generated by the reasoner, and as a query language.

In AceWiki, words have to be defined before they can be used. At the moment, five types of words are supported: proper names, nouns, transitive verbs,

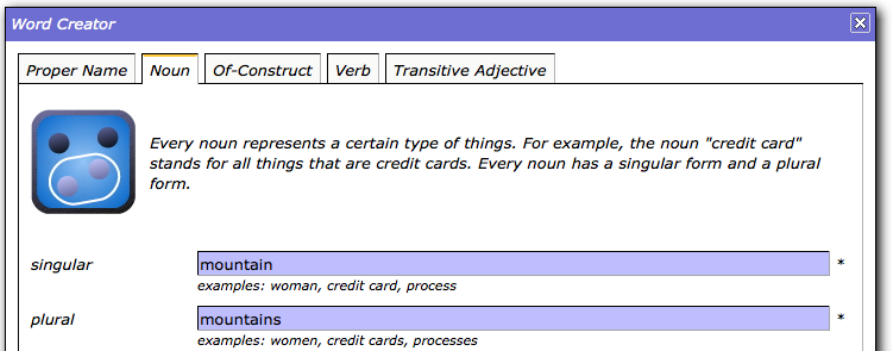


Fig. 3. The lexical editor of AceWiki helps the users to define the word forms. The example shows how a new noun "mountain" is created.

of-constructs (i.e. nouns that have to be used with *of*-phrases), and transitive adjectives (i.e. adjectives that require an object). A new feature of AceWiki is that proper names can have an abbreviation that has exactly the same meaning as the long proper name. This is very helpful for proper names that are too long to be spelled out each time.

Figure 3 shows the lexical editor of AceWiki that helps the users in creating and modifying word forms in an appropriate way. An icon and an explanation in natural language help the users to choose the right category.

3 Evaluation

In order to find out how usable and how useful AceWiki is, we performed several tests. From time to time, we set up small usability experiments to test how well normal users are able to cope with the current version of AceWiki. Two such experiments have been performed so far. In order to find out whether AceWiki can be useful in the real world, we additionally conducted a small case study in which we tried to formalize the content of the existing Attempto project website in AceWiki. The results are explained in the following sections. Table 1 shows an overview.

3.1 Usability Experiments

Two usability experiments have been performed so far on AceWiki. The first one took place in November 2007 and has been described and analyzed in [8]. The second experiment — that is introduced here — was conducted one year later in November 2008. Both experiments have the nature of cheap ad hoc experiments with the goal to get some feedback about possible weak points of AceWiki. Since the settings of the two experiments were different and since the number

Table 1. This table compares the settings of the three tests that have been performed on AceWiki.

	Experiment 1	Experiment 2	Case Study
time	Nov 2007	Nov 2008	Nov/Dec 2008
AceWiki version	0.2.1	0.2.9	0.2.10
number of subjects (n)	20	6	1
subjects	mostly students	students	AceWiki developer
level of preexisting knowledge about AceWiki	none	low	highest
domain to be represented	"the real world"	universities	Attempto project

of subjects was relatively low, we cannot draw strong statistical conclusions. Nevertheless, these experiments can give us valuable feedback about the usability of AceWiki.

In both experiments, the subjects were told to create a formal knowledge base in a collaborative way using AceWiki. The task was just to add correct and meaningful knowledge about the given domain without any further constraints on the kind of knowledge to be added. The subjects — mostly students — had no particular background in formal knowledge representation. The domain to be represented was the real world in general in the first experiment, and the domain of universities (i.e. students, departments, professors, etc.) in the second experiment.

In the first experiment, the subjects received no instructions at all how AceWiki has to be used. In the second experiment, they attended a 45 minutes lesson about AceWiki. Another important difference is that the first experiment used an older version of AceWiki where templates could be used for the creation of certain types of sentences (e.g. class hierarchies). This has been removed in later versions because of its lack of generality.

Table 2 shows the results of the two experiments. Since the subjects worked together on the same knowledge base and could change or remove the contributions of others, we can look at the results from two perspectives: On the one hand, there is the community perspective where we consider only the final result, not counting the sentences that have been removed at some point and only looking at the final versions of the sentences. On the other hand, from the individuals perspective we count also the sentences that have been changed or removed by another subject. The different versions of a changed sentence count for each of the respective subjects. However, sentences created and then removed by the same subject are not counted, and only the last version counts for sentences that have been changed by the same subject.

The first part of the table shows the number and type of sentences the subjects created. In total, the resulting knowledge bases contained 179 and 93 sentences, respectively. We checked these sentences manually for correctness. S^+ stands for the number of sentences that are (1) logically correct and (2) sensible to state.

Table 2. This table shows the results of the first (Exp. 1) and the second (Exp. 2) experiment. The results can be seen from the individuals perspective (ind.) or the community perspective (comm.)

		Exp. 1		Exp. 2	
		ind.	comm.	ind.	comm.
total sentences created	S	186	179	113	93
correct sentences	S^+	148	145	76	73
correct sentences that are complex	S^+_x	91	89	54	51
sentences using "a" instead of "every"	S^e	9	9	23	12
sentences using misclassified words	S^w	9	8	0	0
other incorrect sentences	S^-	20	17	14	8
% of correct sentences	S^+/S	80%	81%	67%	78%
% of (almost) correct sentences	$(S^+ + S^e)/S$	84%	86%	88%	91%
% of complex sentences	S^+_x/S^+	61%	61%	71%	70%
total words created	w	170	170	53	50
individuals (i.e. proper names)	w_p	44	44	11	10
classes (i.e. nouns)	w_n	81	81	14	14
relations total	w_r	45	45	28	26
transitive verbs	w_v	39	39	20	18
of-constructs	w_o	6	6	2	2
transitive adjectives	w_a	–	–	6	6
sentences per word	S/w	1.09	1.05	2.13	1.86
correct sentences per word	S^+/w	0.87	0.85	1.43	1.46
total time spent (in minutes)	t	930.9	930.9	360.2	360.2
av. time per subject	t/n	46.5	46.5	60.0	60.0
av. time per correct sentence	t/S^+	6.3	6.4	4.7	4.9
av. time per (almost) correct sentence	$t/(S^+ + S^e)$	5.9	6.0	3.6	4.2

The first criterion is simple: In order to be classified as correct, the sentence has to represent a correct statement about the real world using the common interpretations of the words and applying the interpretation rules of ACE.

The second criterion can be explained best on the basis of the sentences of the type S^e. These sentence start with "a ..." like for example "a student studies at a university". This sentence is interpreted in ACE as having only existential quantification: "there is a student that studies at a university". This is certainly a logically correct statement about the real world, but the writer probably wanted to say "every student studies at a university" which is a more precise and more sensible statement. For this reason, such statements are not considered correct, even though they are correct from a purely logical point of view.

Sentences of the type S^e have been identified in [8] as one of two frequent error types when using AceWiki. The other one — denoted by S^w — are sentences using words in the wrong word category like for example "every London is a city" where "London" has been added as a noun instead of a proper name.

It is interesting that the incorrect sentences of the types S^e and S^w had the same frequency in the first experiment, but evolved in different directions in the

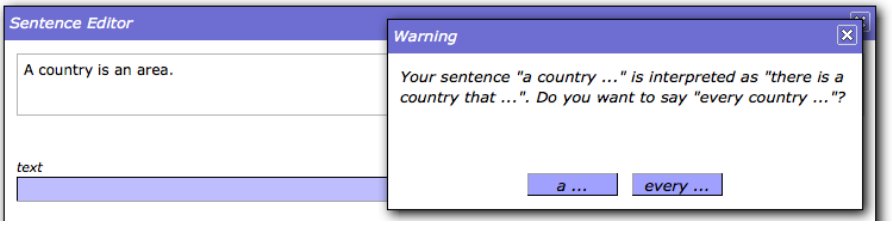


Fig. 4. This figure shows a solution to the problem that the users often state sentences starting with "a ..." when they should say "every ...". This is included in the latest version of AceWiki.

second experiment. There was not a single case of S^w-mistakes in the second experiment. This might be due to the fact that we learned from the results of the first experiment and enriched the lexical editor with icons and explanations (see Figure 3).

On the other hand, the number of S^e-mistakes increased. This might be caused by the removal of the templates feature from AceWiki. In the first experiment, the subjects were encouraged to say "every ..." because there were templates for such sentences. In the second experiment, those templates were not available anymore and the subjects were tempted to say "a" instead of "every". This is bad news for AceWiki, but the good news is that there are two indications that we are on the right track nevertheless. First, while none of the S^e-sentences has been corrected in the first experiment, almost half of them have been removed or changed by the community during the second experiment. This indicates that some subjects of the second experiment recognized the problem and tried to resolve it. Second, the S^e-sentences can be detected and resolved in a very easy way. Almost every sentence starting with "a ..." is an S^e-sentence and can be corrected just by replacing the initial "a" by "every". After the second experiment, we added a new feature to AceWiki that asks the users each time they create a sentence of the form "a ..." whether it should be "every ...". The users can then say whether they really mean "a ..." or whether it should be rather "every ...". In the latter case the sentence is automatically corrected. Figure 4 shows a screenshot of the dialog shown to the users. Future experiments will show whether this solves the problem.

An interesting figure is of course the ratio of correct sentences S^+/S. As it turns out, the first experiment exhibits the better ratio for both perspectives: 80% versus 67% for the individuals and 81% versus 78% for the community. However, since S^e-sentences are easily detectable and correctable (and hopefully a solved problem with the latest version of AceWiki), it makes sense to have a look at the ratio of "(almost) correct" sentences consisting of the correct sentences $S+$ and the S^e-sentences. This ratio was better in the second experiment: 84% versus 88% for the individuals perspective; 86% versus 91% for the community perspective. However, the settings of the experiments do not allow us to

draw any statistical conclusions from these numbers. Nevertheless, these results give us the impression that a ratio of correct and sensible statements of 90% and above is achievable with our approach.

Another important aspect is the complexity of the created sentences. Of course, syntactically and semantically complex statements are harder to construct than simple ones. For this reason, we classified the correct sentences according to their complexity. S_c^+ stands for all correct sentences that are complex in the sense that they contain a negation ("no", "does not", etc.), an implication ("every", "if ... then", etc.), a disjunction ("or"), a cardinality restriction ("at most 3", etc.), or several of these elements. While the ratio of complex sentences was already very high in the first experiment (around 60%), it was was even higher in the second experiment reaching 70%. Looking at the concrete sentences the subjects created during the second experiment, one can see that they managed to create a broad variety of complex sentences. Some examples are shown here:

- Every lecture is attended by at least 3 students.
- Every lecturer is a professor or is an assistant.
- Every professor is employed by a university.
- If X contains Y then X is larger_than Y.
- If somebody X likes Y then X does not hate Y.
- If X is a student and a professor knows X then the professor hates X or likes X or is indifferent_to X.

The last example is even too complex to be represented in OWL. Thus, the AceWiki user interface seems to scale very well in respect to the complexity of the ontology.

The second part of Table 2 shows the number and types of the words that have been created during the experiment. All types of words have been used by the subjects with the exception that transitive adjectives were not supported by the AceWiki version used for the first experiment. It is interesting to see that the first experiment resulted in an ontology consisting of more words than correct sentences, whereas in the second experiment the number of correct sentences clearly exceeds the number of words. This is an indication that the terms in the second experiment have been reused more often and were more interconnected.

The third part of Table 2 takes the time dimension into account. On average, each subject of the first experiment spent 47 minutes, and each subject of the second experiment spent 60 minutes. The average time per correct sentence that was around 6.4 minutes in the first experiment was much better in the second experiment being only 4.9 minutes. We consider these time values very good results, given that the subjects were not trained and had no particular background in formal knowledge representation.

In general, the results of the two experiments indicate that AceWiki enables unexperienced users to create complex ontologies within a short amount of time.

3.2 Case Study

The two experiments presented above seem to confirm that AceWiki can be used easily by untrained persons. However, usability does not imply the usefulness

Table 3. This table lists the results of the AceWiki case study where the content of the Attempto website was formalized in AceWiki.

total sentences created	S	538	
complex sentences	S_x	107	
% of complex sentences	S_x/S	19.9%	
total words created	w	261	
individuals (i.e. proper names)	w_p	184	
classes (i.e. nouns)	w_n	46	
relations total	w_r	31	
transitive verbs	w_v	11	
of-constructs	w_o	13	
transitive adjectives	w_a	7	
sentences per word	S/w	2.061	
time spent (in minutes)	t	347.8	(= 5.8 h)
av. time per sentence	t/S	0.65	(= 38.8 s)

for a particular purpose. For that reason, we performed a small case study to exemplify how an experienced user can represent a strictly defined part of real-world knowledge in AceWiki in a useful way.

The case study presented here consists of the formalization of the content of the Attempto website[9] in AceWiki. This website contains information about the Attempto project and its members, collaborators, documents, tools, languages, and publications, and the relations among these entities. Thus, the information provided by the Attempto website is a piece of relevant real-world knowledge.

In the case study to be presented, one person — the author of this paper who is the developer of AceWiki — used a plain AceWiki instance and filled it with the information found on the public Attempto website. The goal was to represent as much as possible of the original information in a natural and adequate way. This was done manually using the AceWiki editor without any kind of automation.

Table 3 shows the results of the case study. The formalization of the website content took less than six hours and resulted in 538 sentences. This gives an average time per sentence of less than 40 seconds. These results give us some indication that AceWiki is not only usable for novice users but can also be used in an efficient way by experienced users.

Most of the created words are proper names (i.e. individuals) which is not surprising for the formalization of a project website. The ratio of complex sentences is much lower than the ones encountered in the experiments but with almost 20% still on a considerable level.

Basically, all relevant content of the Attempto website could be represented in AceWiki. Of course, the text could not be taken over verbatim but had to be rephrased. Figure 5 exemplary shows how the content of the website was formalized. The resulting ACE sentences are natural and understandable.

[9] `http://attempto.ifi.uzh.ch`

Fig. 5. This figure shows an example text that occurs on the Attempto website (top) and how it was represented in AceWiki (bottom).

However, some minor problems were encountered. Data types like strings, numbers, and dates would have been helpful but are not supported. ACE itself has support for strings and numbers, but AceWiki does not use this feature so far. Another problem was that the words in AceWiki can consist only of letters, numbers, hyphens, and blank spaces[10]. Some things like publication titles or package names contain colons or dots which had to be replaced by hyphens in the AceWiki representation. We plan to solve these problems by adding support for data types and being more flexible in respect to user-defined words.

Figure 6 shows a wiki article that resulted from the case study. It shows how inline queries can be used for automatically generated and updated content. This is an important advantage of such semantic wiki systems. The knowledge has to be asserted once but can be displayed at different places. In the case of AceWiki, such automatically created content is well separated from asserted content in a natural and simple manner by using ACE questions.

As can be seen on Figure 6, the abbreviation feature for proper names has been used extensively. The answer lists show the abbreviations in parentheses after the long proper names. The most natural name for a publication, for example, is its title. However, sentences that contain a spelled-out publication title become very hard to read. In such cases, abbreviations are defined which can be used conveniently to refer to the publication.

The fact that the AceWiki developer is able to use AceWiki in an efficient way for representing real world knowledge does of course not imply that every experienced user is able to do so. However, we can see the results as an upper boundary of what is possible to achieve with AceWiki, and the results show that AceWiki *in principle* can be used in an effective way.

[10] Blank spaces are represented internally as underscores.

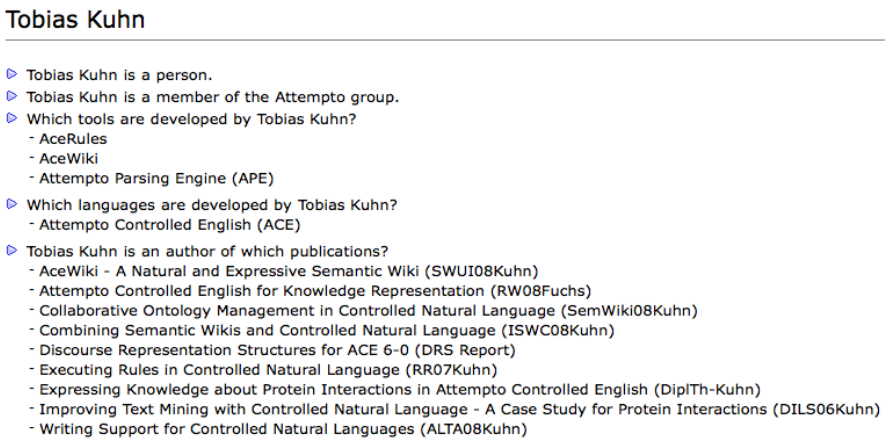


Fig. 6. This figure shows an exemplary wiki article that resulted from the case study. Inline queries are used to generate content that is automatically updated.

4 Conclusions

We presented the AceWiki system that should solve the problems that existing semantic wikis do not support expressive ontology languages and are hard to understand for untrained persons. AceWiki shows how semantic wikis can serve as online ontology editors for domain experts with no background in formal methods.

The two user experiments indicate that unexperienced users are able to deal with AceWiki. The subjects managed to create many correct and complex statements within a short period of time. The presented case study indicates that AceWiki is suitable for formalization tasks of the real world and that it can be used — in principle — by experienced users in an efficient way. Still, more user studies are needed in the future to prove our claim that controlled natural language improves the usability of semantic wikis.

In general, we showed how controlled natural language can bring the Semantic Web closer to the end users. The full power of the Semantic Web can only be exploited if a large part of the Web users are able to understand and extend the semantic data.

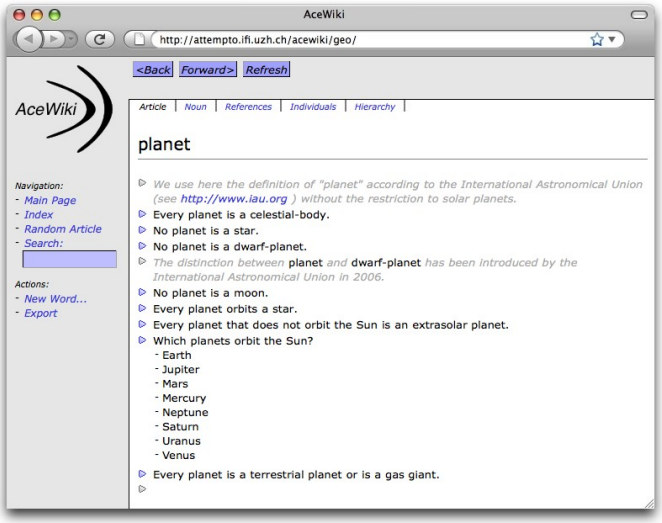
References

1. Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki — A Tool for Social, Semantic Collaboration. In *Proceedings of the 5th International Semantic Web Conference*, number 4273 in Lecture Notes in Computer Science, pages 736–749. Springer, 2006.

2. Jie Bao and Vasant Honavar. Collaborative Ontology Building with Wiki@nt — a Multi-agent Based Ontology Building Environment. In *ISWC Workshop on Evaluation of Ontology-based Tools (EON)*, pages 37–46, 2004.
3. Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.
4. Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):84–97, 2008.
5. Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7–11, 2008, Tutorial Lectures*, number 5224 in Lecture Notes in Computer Science, pages 104–124. Springer, 2008.
6. Kaarel Kaljurand. *Attempto Controlled English as a Semantic Web Language*. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu, 2007.
7. Markus Krötzsch, Denny Vrandečić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic Wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):251–261, December 2007.
8. Tobias Kuhn. AceWiki: A Natural and Expressive Semantic Wiki. In *Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*, 2008.
9. Tobias Kuhn. AceWiki: Collaborative Ontology Management in Controlled Natural Language. In *Proceedings of the 3rd Semantic Wiki Workshop*, volume 360. CEUR Proceedings, 2008.
10. Tobias Kuhn. How to Evaluate Controlled Natural Languages. Extended abstract for the *Workshop on Controlled Natural Language 2009*, (to appear).
11. Tobias Kuhn and Rolf Schwitter. Writing Support for Controlled Natural Languages. In *Proceedings of the Australasian Language Technology Workshop (ALTA 2008)*, 2008.
12. Sebastian Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *Proceedings of the First International Workshop on Semantic Technologies in Collaborative Applications (STICA 06)*, pages 388–396, 2006.
13. Daniel Schwabe and Miguel Rezende da Silva. Unifying Semantic Wikis and Semantic Web Applications. In Christian Bizer and Anupam Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008)*, volume 401. CEUR Workshop Proceedings, 2008.
14. Rolf Schwitter, Kaarel Kaljurand, Anne Cregan, Catherine Dolbear, and Glen Hart. A Comparison of three Controlled Natural Languages for OWL 1.1. In *4th OWL Experiences and Directions Workshop (OWLED 2008 DC)*, Washington, 1–2 April 2008.
15. Katharina Siorpaes and Martin Hepp. myOntology: The Marriage of Ontology Engineering and Collective Intelligence. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007)*, pages 127–138, 2007.
16. Roberto Tazzoli, Paolo Castagna, and Stefano Emilio Campanini. Towards a Semantic Wiki Wiki Web. In *Poster Session at the 3rd International Semantic Web Conference (ISWC2004)*, 2004.
17. Tania Tudorache, Jennifer Vendetti, and Natalya F. Noy. Web-Protégé: A Lightweight OWL Ontology Editor for the Web. In *5th OWL Experiences and Directions Workshop (OWLED 2008)*, 2008.

Natural

AceWiki is a semantic wiki using the controlled natural language ACE (Attempto Controlled English).



ACE supports a wide range of natural language constructs:

- Proper names, Nouns, Verbs, Adjectives
- Of-constructs: part of, child of, owner of
- Number restrictions: at most 3 countries
- Relative phrases: ... that orbits the Sun
- Anaphoric references: the country, the planet
- Existential and universal quantifiers: a, every
- Negation: no, does not, is not, it is false that
- Pronouns: something, everybody, what
- Conditional sentences: if ... then ...

... and much more.

Expressive

ACE is a formal language that is translatable into logic and other languages, for example OWL. AceWiki is designed to seamlessly integrate a reasoner (currently Pellet). The reasoner is used to ensure that the ontology is always consistent:

- Every country is a part of exactly 1 continent.
- Every country that borders Switzerland is a part of Europe.
- Germany borders Switzerland.
- Germany is a part of Asia.

Because ACE is more expressive than OWL, one can express complex statements that are beyond OWL (currently not used for reasoning):

- No ocean borders every continent.
- Every person that has a car owns the car or leases the car.
- If Berlin is a capital then Germany is a stable country.
- Every trip that starts at X and that ends at X is a round trip.

Questions in ACE can be used to express queries that are answered by the reasoner:

- Which cities are located in a country that borders Switzerland?
  - Berlin
  - Milano
  - Paris
  - Rome

The reasoner is used to infer class memberships and class hierarchies:

Upward

- Every country is an area.
- Every country is an object.

Downward

- Every baltic state is a country.
- Every city-state is a country.
- Every landlocked-country is a country.

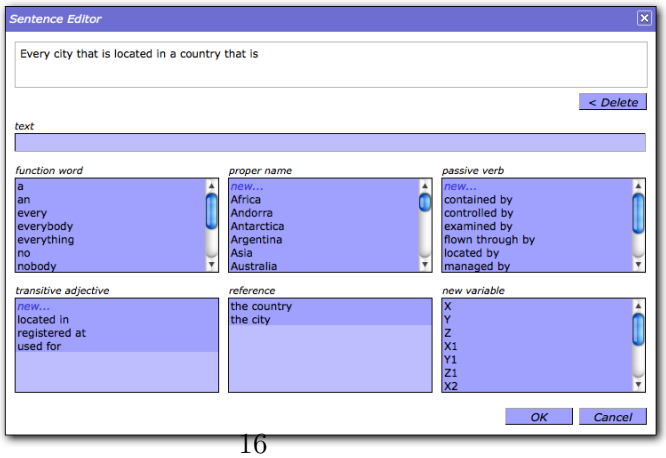
AceWiki

Website: attempto.ifi.uzh.ch/acewiki
Language: Java
License: LGPL
Version: 0.2.13 (alpha)

Tobias Kuhn
tkuhn@ifi.uzh.ch
Department of Informatics
University of Zurich

Usable

AceWiki makes it easy to add and modify ACE sentences. A predictive editor helps the users to create sentences that comply with the ACE syntax:

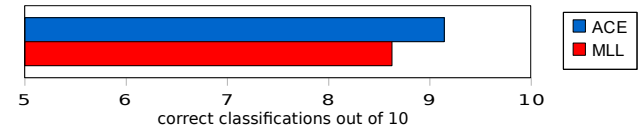


Two usability experiments showed that AceWiki is easy to learn and use. Untrained subjects were told to collaboratively create a knowledge base using AceWiki:

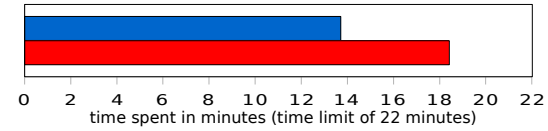
- 78%–81% of the sentences were correct and sensible
- 61%–70% of them were complex (containing negations, implications, disjunctions, or number restrictions)
- Creation of a correct sentence every 5–6 minutes
- Definition of a new word every 5–7 minutes

Understandable

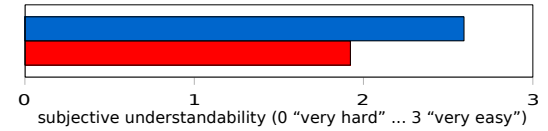
We performed an understandability experiment that compares the understandability of ACE to a comparable common formal language: MLL, a language that is heavily inspired by the Manchester OWL Syntax.
During the experiment, the subjects had to classify 10 statements in ACE/MLL as true or false according to a situation depicted by a diagram. Our results show that ACE was understood significantly better:



Furthermore, ACE required significantly less time to be learned:



After the experiment, the subjects stated that ACE was more understandable:



Information Extraction in Semantic Wikis

Pavel Smrž and Marek Schmidt

Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
E-mail: {smrz,ischmidt}@fit.vutbr.cz

Abstract. This paper deals with information extraction technologies supporting semantic annotation and logical organization of textual content in semantic wikis. We describe our work in the context of the KiWi project which aims at developing a new knowledge management system motivated by the wiki way of collaborative content creation that is enhanced by the semantic web technology. The specific characteristics of semantic wikis as advanced community knowledge-sharing platforms are discussed from the perspective of the functionality providing automatic suggestions of semantic tags. We focus on the innovative aspects of the implemented methods. The interfaces of the user-interaction tools as well as the back-end web services are also tackled. We conclude that though there are many challenges related to the integration of information extraction into semantic wikis, this fusion brings valuable results.

1 Introduction

A frequently mentioned shortcoming of wikis used in the context of knowledge management is the inconsistency of information that often appears when wikis are put to everyday use of more than a few knowledge workers. Semantic wikis, combining the easy-to-participate nature of wikis with semantic annotations, have a strong potential to help in this situation and to become the ultimate collaborative knowledge management system. However, adding metadata means additional work and requires user's attention and thinking. Since it is often difficult to give users immediate satisfaction in reward for this tedious work, annotations in internal wikis tend to be rather scarce. This situation has a negative impact on comprehension of the advantages of semantic wikis and discourages their extensive deployment.

Information extraction (IE) can be seen as a means of reducing user's annotation workload. It refers to the automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources [19]. The state-of-the-art IE technology can produce metadata from content, provide users with useful suggestions on potential annotations and ask questions relevant for the current context. The ultimate goal of IE in semantic wikis is to maximize the benefits of the rich annotation and, at the same time, minimize the necessity of manual tagging.

New application domains raise various challenges for large-scale deployments of IE models. Despite more than two decades of intensive research, the accuracy of the systems is still unsatisfactory for many tasks. Moreover, the results strongly depend on the domain of applications and the solutions are not easy to be ported to other domains. Language dependency is also an issue as the level of analysis required by some methods is available only for a few languages. Another difficulty, particularly significant for the use of IE technology in semantic wikis, lies in the limited character of examples that could be used to train extraction models. Indeed, the real use of semantic technologies calls for specialized annotations of complex relations rather than simple and frequent entities such as places, dates etc. Users are not willing to look for more than one or two other occurrences of a particular relation that should be automatically tagged.

The issues related to standard IE solutions also determine the work described in this paper. As almost all realistic IE-integration scenarios involve system suggestions and user interaction, the IE components that have been designed and are successively developed can be taken as a kind of semantic wiki recommendation system. We pay a special attention to the "cold-start problem" which appears in the beginning of the technology deployment when there are no data to provide high quality suggestions. Taking the semantic wiki as an open linking data platform (rather than a tool to enrich data with semantics for internal purposes only) helps in this respect as one can immediately take advantage of external data sources. We also deal with the incremental character of IE tasks running on gradually growing wiki pages. The implemented interfaces of the IE services facilitate the process of continuous updating of the annotations. They also help to interlink external resources that are modified independently of the controlled updates in the internal wiki.

The following sections tackle the mentioned challenges and show how IE can be used in real semantic wiki systems. As a use case, the next section briefly introduces the IE techniques and tasks applied in the KiWi project. Section 3 discusses specific features of the IE techniques required by the semantic wikis and the innovative aspects of the KiWi solutions. We conclude with future directions of our work.

2 IE Techniques and Tasks in the KiWi project

2.1 Conceptual Model

The main objective of the KiWi (Knowledge in a Wiki[1]) project is to facilitate *knowledge sharing* and *collaboration* among the users of the system to manage knowledge in a more efficient way [20]. Together with personalization, reasoning and reason maintenance, IE belongs to the key enabling technologies in KiWi.

There are two main use cases in this project. The first one is provided by Sun Microsystems, Prague, and is focused on knowledge management in software development, particularly in the NetBeans project. The second one addresses

[1] `http://www.kiwi-project.eu`

vital issues in process management in Logica. The examples given in this paper are taken from those use cases.

KiWi allows users to add meta-data to individual pages or their parts in the form of free or semantic tags. The role of IE is to support users in creating the semantic meta-data and making the knowledge explicit so that it can be further queried and reasoned in a semantic way. The conceptual model for IE in KiWi consists of three major components:

- content items;
- text fragments;
- annotations of content items.

Content item refers to any entity that can be identified. Text fragment is an arbitrary continuous piece of a content item. Text fragments are content items themselves. It enables adding metadata to individual text fragments. In a simple case of commenting a piece of information on a wiki page, the metadata can be of type "comment" and can contain the text of the comment. Tagging text fragments provides a bridge between structured and unstructured information. The fragments can be taken as generalizations of links representing any kind of related resources. In that sense, the fragments are independent of the formatting structure of the wiki page.

Figure 1 demonstrates the way in which information extracted from three different content items (wiki pages) is put together. All the pages mention the same resource – an issue identified by its number #1223 (corresponding to Sun's bug-reporting site IssueZilla). In the "Description of Issue #2536", one can read that the issue is actually a consequence of Issue #1223. The page on "Release Engineering" page says that issue #1223 is critical for the current release of the final product. Finally, "Meeting minutes" assign the task to fix the bug to Joe. The information extraction component is able to extract the mentioned pieces of information and save them (as a set of RDF triples) for further processing.

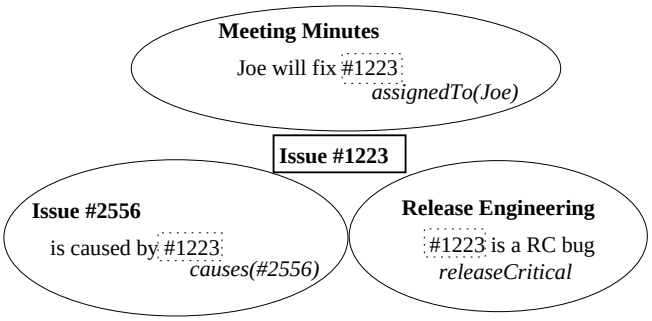


Fig. 1. Information about one entity may come from different pages

Note that the described concept of tagging text fragments that represent a resource (rather than to simply join the ascertained statement to the particu-

lar resource) enables identifying sources of extracted information, synchronizing text and tags, keeping track of the consequences of the changes and pointing out inconsistencies. What is even more important for the semi-automatic IE processes, it also makes the manual corrections easier and allows users to improve the IE accuracy by providing explicit feed-back to the system's decisions.

KiWi is designed as a modular system, in which modules provide additional functionality to the core system via *widgets* which a user may add to her custom layout. The main interaction between the IE system and the user is realized by the *annotation widget*. Figure 2 demonstrates the use of the widget for IE from meeting minutes. It shows information extracted from the text fragment appearing on the wiki page. The widget also offers actions enabled by the semantics of extracted information (such as inserting the event to the calendar, showing the map of the place, etc.). The annotation editor that has been also developed allows users to manually annotate fragments directly in the KiWi editor and to inspect associated semantic information in the annotation widget.

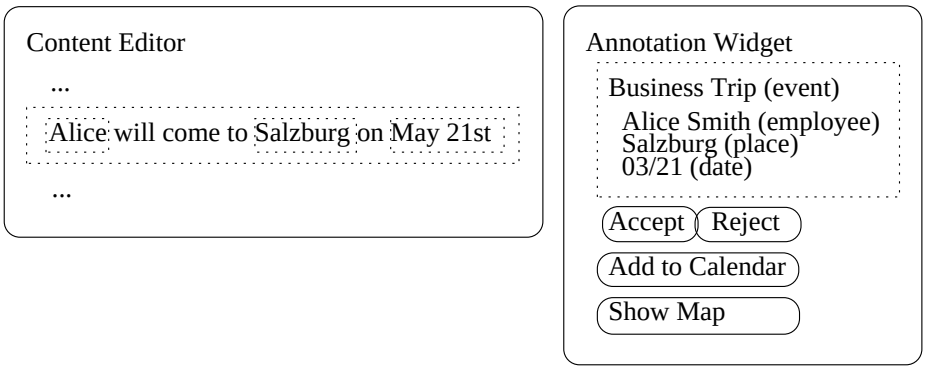


Fig. 2. An example of metadata and action suggestions provided by the KiWi annotation component

KiWi aims at an application of the state-of-the-art IE methods in the semantic wikis [8]. Various techniques and methods are employed to fulfil the key tasks identified in the project.

IE from wiki pages deals mostly with free text. In general, it can therefore benefit from a thorough language analysis of the input. In addition to tokenization and sentence splitting, the natural language processing may involve stop-list filtering, stemming or lemmatization, POS tagging, chunking and shallow or deep parsing. Many of these pre-processing steps are computationally expensive. Almost all of them are strongly language-dependent. Moreover, there is danger of cascading of errors from pre-processing in the IE system. That is why we follow the current trend and apply selective pre-processing in KiWi. The methods described below take into account the context in which they work and employ only those pre-processing processes that can bring significant value to the IE itself.

2.2 Implemented IE Techniques

The IE solutions implemented in KiWi rely on several state-of-the-art IE techniques. Before discussing the function of particular IE processes in KiWi, let us therefore mention the crucial techniques employed. The following technologies play a key role in the system:

- automatic term recognition combining domain-specific and general knowledge;
- computation of word relatedness to define similarity measures;
- text classification and sense disambiguation based on advanced machine-learning methods;
- dimensionality reduction for text feature vectors.

Any realistic approach to automatic term recognition (ATR) from wiki pages cannot ignore the fact that the source texts are usually rather short. Unfortunately, most of available ATR methods rely too much on high frequency counts of term occurrences and, therefore, cannot be utilized in the intended field.

To cope with the problem, we adopt a new ATR method proved to give the best r results suggestesults in our previous experiments (see [9]). It flexibly combines the frequency-based measure (a variant of the TF.IDF score) and the comparisons with a background corpus. The current implementation works with a general background data (such as American GigaWord [5] or Google TeraCorpus [1] for English) only. Our future work will focus on an automatic identification of supplementary in-domain texts that would be useful for the "focused background subtraction".

Various approaches to characterize the semantic distance between terms have been also explored in our research. For general terms, we make use of the wordnet-based similarity measures [16] that take into account the hierarchical structure of the resource. The same technique is employed when the closeness of concepts in a domain-specific thesaurus or ontology is to be computed (e.g., on Sun's Swordfish ontology [3]).

An implemented alternative method which does not require manually created resources (such as wordnet-like lexical databases or domain ontologies) determines the semantic similarity of terms by the relative frequency of their appearance in similar contexts. Of course, there are many ways to assess the similarity of contexts. The results of our preliminary experiments suggest that the best performer for the general case is the method taking into account the (dependency) syntactical structure of the contexts [7, 10] (terms are semantically close if they often appear in the same positions, e.g., as subjects of the same verb, modifiers of the same noun, etc.).

Many IE tasks can be formulated as classification problems. This finding is behind the immense popularity of machine learning techniques in the IE field today. In the rare case when there are enough data for training, KiWi follows this trend. Complex features computed on the dependency structures from the source text are gathered first.

The particular set of features applied depends on the task and the language in hand. For English named entity recognition, a gazetteer, word contexts, lexical and part of speech tags are used. For classification of the role an entity plays on a page (which can be interpreted as a semantic role labeling problem [13]), additional features provided by a dependency parser are employed. The classification is performed by CRF (Conditional Random Fields) and SVM (Support Vector Machine) models with tree kernels constructed from syntax trees of the sentences [21]. Depending on the context, the process can identify "soft categories" sorted by the descending probability of correctness. The resulting N-best options are presented to the user who chooses the correct one.

As opposed to the discussed situation, a typical classification task in the context of semantic wikis can be characterized by the limited character of the input text and the lack of data to train the classifier. The advanced methods that can deal with the latter issue are discussed in the next section. Let us therefore just note, that to overcome the former one (inherent to the wiki world), KiWi harnesses the other mentioned techniques and personal/organizational contexts to characterize the "ground" of the material provided by the user and to increase accuracy of the classification.

As exemplified by the Logica use-case in KiWi, the semantic wikis in the professional setting often need to integrate large sets of business documents (product specifications, customer requirements, etc.). Having such a document in hand, the user can ask the system to find similar documents in the given collection. As the terminology and the style of the documents can differ significantly, the straightforward computing of the similarity as a function of term co-occurrences is often insufficient. Standard approaches to overcome this (such as PLSA – Probabilistic Latent Semantic Analysis or LDA – Latent Dirichlet Allocation) transform the term vectors representing the documents to point out their semantic closeness.

Unfortunately, the computation of such transformations is prohibitively expensive. KiWi draws on the random indexing technique [6] that is several orders of magnitude faster than the mentioned approaches. As KiWi documents are indexed by means of Apache Lucene search library – we take advantage of Semantic Vectors [22] – a fast implementation of the random indexing concept based on the Lucene indices. This setting provides very efficient mechanism to evaluate similarity queries in KiWi.

2.3 IE Tasks in KiWi

The above-mentioned IE techniques find their application in various tasks and various contexts in the KiWi system. From a general point of view, the whole IE functionality can be seen as tag suggestion or automatic annotation (if the similarity is interpreted as a special kind of tagging). On the other hand, the user perspective distinguishes different kinds of tags for different purposes. The following tasks form the core of the KiWi IE module in the latter sense:

- suggestion of new free-text tags and thesaurus/ontology extensions;

– entity recognition and semi-automatic annotation of content items;
– relation extraction and structured tag suggestion;
– similarity search adapted according to the user's feedback.

Figure 3 (based on the KiWi use case defined by Logica) demonstrates the interplay of these tasks. It shows a situation when a project manager comes to the task to produce a project risk analysis report based on her notes from a preparatory meeting (as displayed in the KiWi annotation editor on the left side of the picture). Risk-related information needs to be formalized, the potential impact should be identified and the resolution strategies explicitly stated. Based on the user-specific setting, the IE component automatically identifies entities such as company products, development technologies, names of employees, dates, places, etc. and classifies the page as a (seed of) risk analysis report – a known type of document with an associated semantic form. The identified type narrows down the similarity search which focuses on those risk analysis reports that mention semantically related risks (it is realized as a simple word-based relatedness function on the "identified risks" sections in the current implementation).

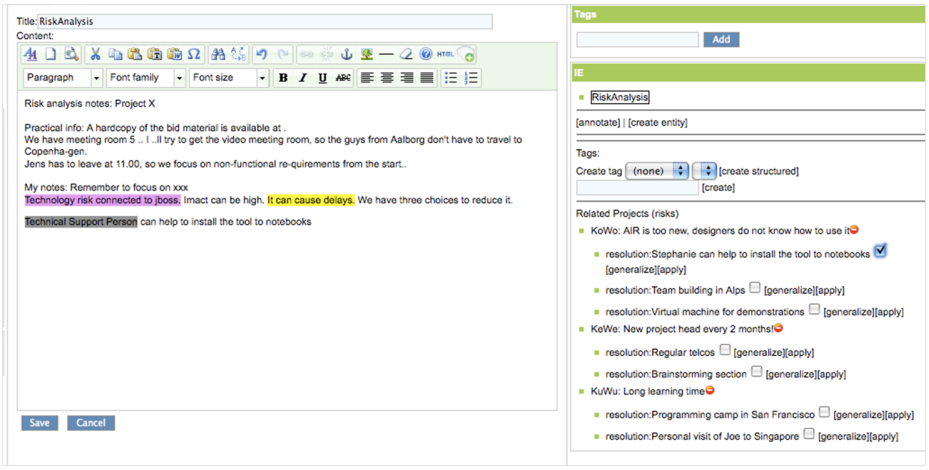


Fig. 3. KiWi annotation component classifying entities and relations and suggesting tags and projects related to a given page according to the identified risks

The annotation component also suggests terms found in the text as additional tags. The possibility to propose free-text tags is not particularly useful in the semantically-rich case discussed but it can be essential for "lightweight" semantic wiki environments. A more practical function in the actual context refers to semi-automatic extending the conceptual domain model. The most frequent form of this process regards thesaurus or ontology population by instances referred to in the analysed text. For example, finding new term "JBoss Seam" in the position where a development tool name is expected, the system can suggest adding the

term as an instance of class "development tools". The problem domain ontology can also be extended in higher levels, e.g., "video meeting room" can be suggested as a subclass of "meeting room".

Entity recognition employs the FSA (finite-state automaton) technology and implements a straightforward gazetteer strategy when it is tightly coupled with the annotation editor to identify types and instances of entities mentioned in the text and to suggest annotations linking the specific reference to the knowledge base. A limited set of rules is applied to identify compound expressions such as names, absolute temporal terms, monetary and other numeric expressions, etc. Apart from that, the functionality is completely based on lists of entities that should be identified in the texts. The lists are populated by terms referring to concepts in general ontologies (e.g., UMBEL[2] or GeoNames[3]) as well as domain-specific resources (such as Sun's Swordfish ontology or a list of team members and their roles). For the fast on-line processing, these extensive lists are compiled to a large FSA which is then used to identify matches in the text and to provide the type of the suggested tag.

Similarity search takes advantage of pre-computed matrices of term relatedness. This is crucial especially for comparing short text fragments such as the "identified risk" sections discussed above. Particular matrices correspond to various measures of the semantic distance between terms. Except for the batch document clustering, the similarity search is always intended for the on-line mode. The pre-computation of the term similarities in the form of the matrices helps to speed up the search significantly.

For the fast search on short text fragments (less than a paragraph), KiWi computes the information gain of the terms appearing in the text. The lines corresponding to the most informative terms are taken from the term-closeness matrices. This provides a simple query-expansion mechanism. The query combining the terms from the actual fragment and the semantically close terms (weighted by the informativeness and similarity, respectively) is evaluated on the all content items of the same type and the best matches are retrieved.

The whole wiki pages and full documents are indexed by the Lucene library. To perform similarity search on this kind of documents, SemanticVectors [22] are employed. It is often the case that the retrieved documents do not come up to user's expectations. The most informative terms can prove to be unimportant from the user's perspective. That is why it is very important to let KiWi users know why the particular documents are considered similar to that one in question and what terms played the key role in the system's decision. KiWi lists those terms for the entire set of the similar documents and for each individual document as well. The user can mark some of the terms as unimportant for the current context and the system re-computes the similarity with the new restricted set of terms.

The concept of tags in KiWi is rather general. It comprises the standard label-like tags, but also structured ones that encode relations of the concept

[2] `http://www.umbel.org`
[3] `http://www.geonames.org`

represented by the given term to other concepts. The corresponding IE task of relation extraction extracts facts from relations between entities in a wiki page (e.g., from statements like *Alice will travel to Salzburg on May 21st*). The relation can also be identified between an entity and the wiki page itself, since every page in the KiWi represents some entity.

The implementation of the relation extraction algorithm is similar to that of entity recognition. It employs advanced machine learning models (CRF mentioned above) and incorporates additional information provided by the user to improve the performance. For example, the user can specify features relevant for semi-structured documents as an XPath expression (e.g., to inform the automatic extraction method that the cost is always in the second column of a table). Unfortunately, the process is prone to the errors in the language analysis layer so that the results strongly depend on the quality of the language-dependent pre-processing phase.

Semantic wikis with annotations support evolution of knowledge from free-form to structured formalized knowledge. The role of IE is to support the user in the process of creating semantic annotations. If the structure of knowledge is well understood, the annotations can take a unified form of tags anchored in a domain ontology. However, the "wiki way" of knowledge structure that is only emerging in the collaborative work process calls for sharing of free-text tags as well. KiWi supports this by means of new tag suggestions based on the ATR (see above) from a particular document or a wiki page. Users can choose which extracted terms are appropriate to tag the resource and what their relations to other tags are. For ATR on short wiki pages, KiWi engages heuristics based on simple grammar patterns (such as "an adjective followed by a noun") to propose the candidate terms.

In addition to free-text tagging, ATR makes it also possible to suggest extensions to a domain ontology or thesaurus. KiWi checks whether the extracted terms correspond to existing concepts and if not, it proposes additions. If there are enough data for classification training, it can also find the most probable class to link the new concept to.

3 Innovative Aspects of IE in KiWi

As mentioned above, there are many challenges and open questions related to the use of IE in semantic wikis. The state-of-the-art IE systems [2, 12, 17] often make assumptions about the type of data, its size and availability, and the user interaction mode that are not acceptable in the given context. KiWi explores solutions that are able to cope with the problems and work the "wiki way" (provide sophisticated functionality but easy to understand and easy to use).

Machine learning plays a central role in the current IE paradigm [14]. From a conceptual point of view, statistical IE systems distinguish two phases: the *training phase* and the *deployment phase*. In the training phase the system acquires a model that covers a given set of annotation examples. In the deployment phase, the system identifies and classifies relevant semantic information in new

texts, i.e., texts that were not included in the training set. The predominant approach expects a large text corpus with annotated information to be extracted, and then uses a learning procedure to extract some characteristics from the annotated texts [23]. Unfortunately, an annotated training data set is available for a very limited number of cases. And it is unrealistic to expect that KiWi users will provide this kind of data to make the system "semantics-aware". This is especially true for the case of many application-specific relations in the explored domains.

To overcome the problem of training data scarcity, IE in KiWi explores a combination of the standard supervised algorithms with the methods that are able to learn from untagged texts. We take advantage of the concept of *bootstrapping*, which refers to the technique that starts from a small initial effort and gradually grows into something larger and more significant [14]. One of the currently employed methods relying on this principle is the *expansion*. An initial extraction model (learned from few examples) is applied to the unannotated text (wiki pages, linked documents or external resources) first. Newly discovered entities or relations that are considered sufficiently similar to other members of the training set are added and the process is iterated.

Another approach we apply is *active learning*. In active learning, the system itself decides what the best candidates for annotation are in order to maximize the speed of the learning process. A user is then asked to annotate these instances only. The idea of active learning perfectly fits the wiki philosophy that every user can annotate every page for which she has sufficient rights. All changes are naturally reported and there is no problem to come back to a previous version in case somebody made inappropriate annotations.

The combination of both methods lets the system exploit the knowledge as much as possible, but still allows users to have full control of the annotation process.

There is not much to do about the dependency of the IE methods on the result of the pre-processing phase. The trade-off between the quality of the language analysis and the general availability of the corresponding tools makes it impossible to provide the same grade of extraction in all languages. KiWi tries to mitigate the "curse of language-dependency" by means of using general resources that are available across languages. For example, our experiments with instances of Wikipedia in several languages used for the expansion proved that this functionality does not need to be limited to a particular language.

In addition to the lack of annotated data for training classifiers, there is also a specific problem of the unusual nature of some IE tasks in semantic wikis. The resources that are to be semantically annotated vary exhibit high diversity. The length ranges from a few words to entire pages and full documents that are uploaded to the system. Especially the lower side on this scale (very short texts) trouble the commonly used IE techniques – they often need more material to find similar contexts, to disambiguate a term, to classify a relation, etc.

One of the techniques that partially deals with the problem of short texts benefits from the PLSA and random projection algorithms discussed above. It

projects the dimensions given by the original set of terms to the space defined by a referential collection of resources. In the case of KiWi, pages from Wikipedia are taken as the dimensions. Thus, it is possible to present the results to the user in an intuitive form – pointing out the articles with the most significant contribution.

The concept of KiWi as the open linking data platform has been already mentioned. The IE technology tries to re-use as much as possible from existing semantic web knowledge resources. Dbpedia and Wikipedia find their place in the training of classifiers and sense disambiguators, the taxonomy based on WordNet and OpenCyc help to define the similarity measures etc. The external data sources are also linked to the user-interaction mode in KiWi. For example, a user defines new semantic tag "programming language" as `http://umbel.org/umbel/semset/en/wikipedia/Programming\_language`. The system fetches all relevant articles from Freebase and trains an initial classifier. The user can start to tag with it immediately and to provide feedback to improve the model.

4 Conclusions and Future Directions

Let us summarize the major point of the work reported in this paper. The application of IE methods on the specific set of problems (texts of varying size and character, complex relations, etc.) with this kind of user interface (semi-automatic, generic, ontology based) is novel. In addition to other results, KiWi brings valuable insights into the practical applicability of the best IE techniques in real conditions.

KiWi promises an advanced knowledge management system with state-of-the-art personalization, reasoning and information extraction features. As the project is still in its early phase (the first year finished in February 2009), only the first open source pre-release of the core system is available for real use. The IE components applicable in the context of the mentioned use-cases have been developed in parallel and are available in the experimental mode.

The accuracy of the IE methods significantly depends on the domain, task and data that can be used. For example, the reported figures for entity recognition range from 60 % to 90 % and generally correspond to the type of entities to be extracted [15]. The precision of the relation extraction task demonstrates even more significant variability (e.g., [18] reports results ranging from 7 % to 90 % on various relations from Wikipedia). It has been shown that the IE process can be useful even if the performance is imperfect [4]. However, to the best of our knowledge, no studies assessed the actual added value of the IE solutions for the highly interactive scenarios which is typical for the semantic wikis. This forms one of the key directions of our future work.

Another challenge we have to face in the next stage comes from the fact that the types of entities and relations to extract are not specified in advance. Users can apply the services to extract information from arbitrary complex texts. They can also specify an ontology and ask the system to identify any given relation. While, e.g., the use of ontologies to drive the IE process has been

already explored [11], it is not yet clear whether the performance of the general IE system, capable of extracting any type of entity or relation only by learning from the user annotations, will be acceptable for the end-users.

Acknowledgement

The work presented in this paper has been supported by European Commission, under the ICT program, contract No. 211932 and under the IST program, contract No. 27490.

References

1. Brants, T., and Franz, A. Web 1T 5-gram Version 1, 2006. Linguistic Data Consortium, Philadelphia.
2. Ciravegna, F., Chapman, S., Dingli, A., and Wilks, Y. Learning to harvest information for the semantic web. In *Proceedings of the 1st European Semantic Web Symposium, Heraklion, Greece* (2004).
3. Cone, S., and MacDougall, K. Case Study: The swoRDFish Metadata Initiative: Better, Faster, Smarter Web Content, 2007. `http://www.w3.org/2001/sw/sweo/public/UseCases/Sun/`.
4. Feldman, R., and Sanger, J. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, December 2006.
5. Graff, D. English Giga-word, 2003. Linguistic Data Consortium, Philadeplhia.
6. Kanerva, P., Kristoferson, J., and Holst, A. Random Indexing of Text Samples for Latent Semantic Analysis. In *22nd Annual Conference of the Cognitive Science Society* (2000), Erlbaum.
7. Kilgarriff, A., Rychly, P., Smrz, P., and Tugwell, D. The sketch engine. In *Practical Lexicography: A Reader*, T. Fontenelle, Ed. Oxford University Press, USA, 2008.
8. Knoth, P., Schmidt, M., and Smrž, P. KiWi deliverable d2.5: Information Extraction – State of the Art, 2008. `http://wiki.kiwi-project.eu/multimedia/kiwi-pub:KiWi_D2.5_final.pdf`.
9. Knoth, P., Schmidt, M., Smrž, P., and Zdráhal, Z. Towards a Framework for Automatic Term Recognition. In *Proceedings of Znalosti (Knowledge) 2009* (2009).
10. Lin, D. Automatic Retrieval and Clustering of Similar Words. In *COLING-ACL* (1998), pp. 768–774.
11. Maedche, E., Neumann, G., and Staab, S. Bootstrapping an ontology-based information extraction system. In *Studies in Fuzziness and Soft Computing, Intelligent Exploration of the Web* (2002), Springer.
12. McDowell, L. K., and Cafarella, M. Ontology-driven information extraction with ontosyphon. In *Proceedings of the International Semantic Web Conference* (2006), pp. 428–444.
13. Mitsumori, T., Murata, M., Fukuda, Y., Doi, K., and Doi, H. Semantic role labeling using support vector machines. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL)* (Ann Arbor, U.S.A., 2005), Association for Computational Linguistics, pp. 197–200. `http://www.lsi.upc.es/~srlconll/st05/papers/mitsumori.pdf`.

14. Moens, M.-F. *Information Extraction: Algorithms and Prospects in a Retrieval Context (The Information Retrieval Series)*. Springer, 2006.
15. Nadeau, D., and Sekine, S. A survey of named entity recognition and classification. *Journal of Linguisticae Investigationes* (2007).
16. Pederson, T., Patwardhan, S., and Michelizzi, J. WordNet::Similarity - Measuring the Relatedness of Concepts, 2004. `http://www.d.umn.edu/~tpederse/similarity.html`.
17. Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., and Kirilov, A. KIM - A semantic platform for information extraction and retrieval. *Natural Language Engineering 10*, 3-4 (2004), 375–392.
18. Ruiz-Casado, M., Alfonseca, E., and Castells, P. From wikipedia to semantic relationships: a semi-automated annotation approach. In *Proceedings of SemWiki06* (2006).
19. Sarawagi, S. Information Extraction. *Foundations and Trends in Databases 1*, 3 (2008), 261–377.
20. Schaffert, S. The KiWi Vision: Collaborative Knowledge Management, powered by the Semantic Web, 2008. `http://www.kiwi-project.eu/index.php/kiwi-vision`.
21. Settles, B. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the COLING 2004 International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA)* (Geneva, Switzerland, 2004). `http://pages.cs.wisc.edu/~bsettles/pub/bsettles-nlpba04.pdf`.
22. Widdows, D., and Ferraro, K. Semantic Vectors: A scalable Open Source package and online technology management application. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)* (Marrakech, Morocco, 2008), ELRA, Ed. `http://code.google.com/p/semanticvectors`.
23. Yangarber, R., and Grishman, R. Machine learning of extraction patterns from unannotated corpora: Position statement. In *Proceedings of Workshop on Machine Learning for Information Extraction* (2001), pp. 76–83.

Undo in Peer-to-peer Semantic Wikis

Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli

LORIA – INRIA Nancy-Grand Est, Nancy Université, France
{Charbel.Rahal, weiss, skaf, urso, molli}@loria.fr

Abstract. The undo mechanism is an essential feature in collaborative editing systems. Most popular semantic wikis support a revert feature, some provide an undo feature to remove any modification at any time. However this undo feature does not always succeed. Supporting the undo mechanism for P2P semantic wikis has never been tackled. In this paper, we present an undo approach for Swooki, the first P2P semantic wiki. We identify the problems to resolve in order to achieve such mechanism for P2P semantic wikis. We give the definition of the undo and the properties that must ensure. This approach allows both a revert and a permanent successful undo.

1 Introduction

Wiki systems are very popular collaborative editing systems. Thanks to a simple syntax, users can build complex text documents, including tables, pictures or videos. As collaborative editors, wiki systems provide an undo mechanism. This mechanism is integrated in two forms, either through a *revert* which allows to return to an old version, or through an *undo* allowing to remove any modification from the current version [24].

In spite of their success, wiki systems have some limitations such as:

Centralization Most of existing wikis are centralized: this implies a high cost to ensure scalability, censorship issues, data availability and durability issues especially in case of failure;

Low structuring Wiki systems are low-structured, they suffer in the navigation and the search, i.e. it is hard to navigate and to find relevant information in wikis [6]. Wiki content is only human readable and it is not accessible and readable by machines, hence, it cannot be reused in external applications.

To overcome these limitations, two orthogonal solutions are proposed: P2P wiki systems and semantic wiki systems.

P2P wiki systems [28, 12] are based on a decentralized architecture and optimistic replication[19] mechanism to improve scalability and data availability. As traditional wiki systems, they suffer from low structuring.

Semantic wiki systems [27, 20, 6] allow users to incorporate some computer readable information in wiki pages. Such information can be used to improve navigation and search. However, they suffer from centralization limitations.

Swooki [21] aims at conciliating both directions, it combines the advantages of P2P systems and semantic wikis. Swooki is a semantic wiki inspired from Semantic MediaWiki [27]. Moreover, Swooki supports massive collaboration, fault tolerance, off-line work mode and ad hoc collaboration thanks to its P2P structure and to the replication of semantic wiki pages on different sites. Unfortunately, this approach does not offer any undo mechanism.

In the literature, several collaborative editing systems offer an undo mechanism. Existing semantic wikis are centralized, therefore their undo mechanism is not adequate for the P2P environment. On the contrary, some undo frameworks were devised for distributed collaborative systems, however they do not support semantic wiki data type. The data type maintained in semantic wikis is the wiki pages and the semantic annotations storage. Our goal is to design an undo mechanism that is compatible with P2P constraints, that supports the semantic wiki data type and that ensures the consistency between the wiki pages and the semantic storage.

In this paper, we propose an undo mechanism for Swooki. We define the property that this mechanism must ensure. This mechanism supports both undo and revert features. We develop the undo component and the needed algorithms and extensions to instantiate this undo mechanism in Swooki.

The paper is organized as follows. In section 2, we motivate the need for the undo mechanism. Section 3 presents existing approaches for the integration of the undo mechanism in collaborative editing systems. Section 4 presents Swooki. An overview of the undo in Swooki approach is given in section 5. Section 6 describes the implementation of the approach. The last section concludes the paper.

2 Motivation

Similarly to classical collaborative editors, P2P semantic wikis require supporting the undo feature for many reasons:

- Undo is a user required feature. Indeed, users can use the undo feature as a powerful way to recover from their proper errors.
- In collaborative editors, when two or more users modify the same data, the system proposes a document based on all modifications. This merge algorithm is a best-effort and is not able to produce the result expected by users. The undo feature is useful to recover from unexpected results.
- We consider a P2P semantic wiki as an open system where anyone can join and leave. Since anyone can join, malicious users can also join. As a result, the undo feature can be used to remove the impact of vandalism acts.

In all these cases, the expected result matches the undo definition [23]:

"Undoing a modification makes the system return to the state it would have reached if this modification was never produced."

To achieve such a goal, the revert feature seems to be adequate: we can remove the whole content and add a previously created one. Unfortunately, the revert feature does not allow to undo any modification.

Another common idea is to undo changes by doing the inverse modification. Unfortunately, this does not achieve the undo definition.

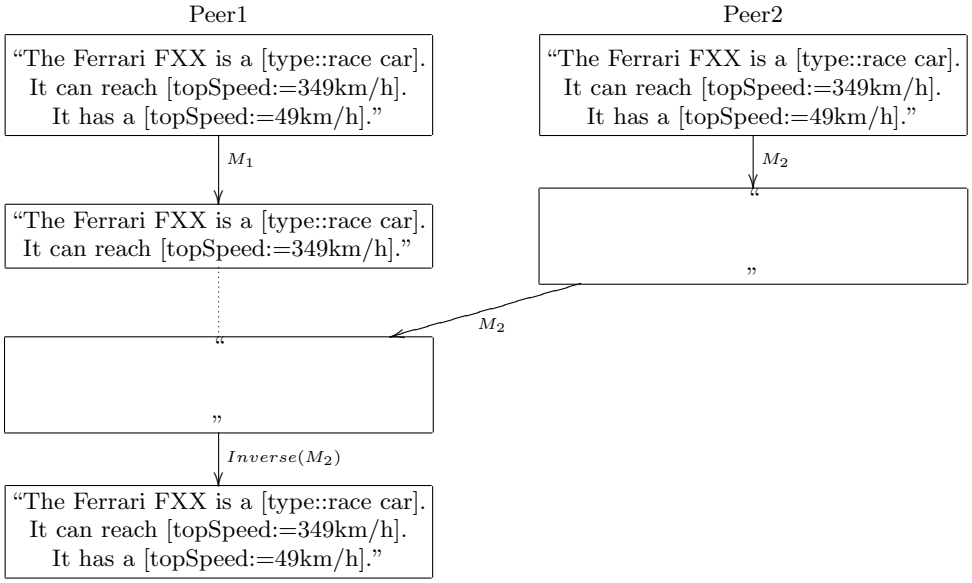


Fig. 1. Undo using the inverse modification

Starting from a version V_1 (Figure 1), a *user*2 on Peer2 generates a malicious modification M_2 by deleting the whole document. M_2 deletes all the document lines. Concurrently, *user*1 on Peer1 deletes the third line that contains an error. The inverse modification of M_2 inserts the three lines. As a consequence, when *user*1 on Peer1 undoes M_2, it reinserts the three lines and looses its proper modification M_1 which aims at deleting the third line. If M_2 was never produced, the document would be corrected to "The Ferrari FXX is a [type::race car]. It can reach [topSpeed:=349km/h].". A correct undo must return the document to this state. Our goal is to provide such an undo feature.

3 Related work

This section gives an overview about the undo mechanism in different collaborative editors.

3.1 Undo in semantic wikis

Semantic wikis such AceWiki [10], Rise [7] and WikiSar [4] do not support versioning for wiki pages, hence they do not support an undo mechanism.

Makna [8] is a wiki based tool for distributed knowledge engineering. It extends the JSPWiki wiki engine with generic, easy to use ontology-driven components for collaboratively authoring, querying and browsing Semantic Web information. Makna supports versioning for pages and metadata within the pages, thus the revert feature is provided. However, it does not support the undo feature.

IkeWiki [20] is a semantic wiki with features to support collaborative knowledge engineering, different levels of formalization ranging from informal texts to formal ontologies, and it has a sophisticated, interactive user interface. IkeWiki supports also a revert feature to restore an old version. IkeWiki does not support a feature to undo modifications applied on a page version. In addition, the annotations about the wiki pages are outside the content of these pages. Tracking the annotations changes is not provided, an insert or a delete of annotations can not be detected.

SweetWiki [6] is a semantic wiki based on the CORESE engine. It supports WYSIWYG edition of pages and annotations, and use the CORESE engine and the SeWeSe library for all semantic operations : navigation, search and others. Pages are annotated using tags which are outside the content of the pages. SweetWiki does not support a versioning support for the formalized content, i.e. changes in the tags on pages are not tracked. SweetWiki supports a revert feature without an undo one.

Rhizome [22] supports a modified version of WikiML (ZML) that uses special formatting conventions to make semantic properties directly explicit in the page content. Pages are saved in RDF and another editor can be used to edit the RDF directly. Rhizome provides a native versioning of content and metadata. It provides only a revert feature without an undo one.

Semantic MediaWiki (SMW) [27] is an extension of MediaWiki that helps to search, organize, tag, browse, evaluate, and share wiki content. SMW adds semantic annotations in wiki pages in order to bring the power of the Semantic Web into the wiki. SMW inherits all the features of MediaWiki including revert and undo. While the revert always succeeds in restoring an old version, in some cases the undo can fail [1]. For instance, the undo of a modification in a paragraph followed by other modifications in the same paragraph can not succeed.

OntoWiki [3] and Powl [2] are web based applications designed to collaboratively build ontologies and create instances. Every change on any element such as knowledge model, concept, property or instance is logged. So they enable users to track, review and selectively roll-back changes. Consequently, they can offer both the revert and the undo features. Unfortunately, their undo mechanism is designed only for ontological elements and not for text.

3.2 Undo in different collaborative editors

DBin [26] enables end users to create and experience the Semantic Web by exchanging RDF knowledge in P2P "topic" channels. DBin is not designed to

[1] `http://en.wikipedia.org/wiki/Wikipedia:Undo#Undo`

exchange and edit semantic wiki pages. However, it can be used as a complementary component for P2P semantic wikis separating the annotations from the wiki page content. There is no indication about the support of an undo mechanism.

Most undo approaches were devised in the Operational Transformation [9] (OT) framework.

In [15], the authors propose to select which operation to undo. They also add the notion of conflict. If a conflict occurs, the undo is aborted. Therefore, this framework does not allow undoing any operation.

In [18], the authors propose a solution to undo operations in the inverse chronological order, i.e. from the last operation to the first one without skipping one. Therefore this approach does not allow undoing any operation.

The GOTO-ANYUNDO approach [23] is the first approach that allows any user to undo any operation at any time. This approach is designed for real-time editing and uses state vectors [11]. Since state vectors size is proportional to the number of sites, this approach cannot be used in a P2P environment.

The COT approach [25] is an OT system designed for real-time editing which introduces the notion of "context vector". A context vector is associated to each operation and represents the operations executed before its generation. As state vectors, context vectors are not suitable in a P2P environment.

Distributed version control systems (DVCS) as Git [2] are P2P collaborative systems mainly used for source code editing. They compute a new patch to remove the effect of a previous one and treat it as a new patch. However, DVCS lack of a formal framework: there is no property to validate their correctness. For instance, in Git, the use of the undo feature may confuse further merges [3].

The UNO[29] framework proposed an undo for P2P collaborative editing based on the Operational Transformation approach. The main idea of this approach is to devise new operations for counterbalancing previously made operations. This framework cannot be used directly to provide an undo feature in Swooki. However, we propose an undo mechanism inspired by this framework capable of undoing any modification at any time, i.e. supporting both a revert and an undo features.

4 Swooki System

Swooki [16, 21, 17] is the first P2P semantic wiki, it combines the advantages of P2P wikis and semantic wikis. Swooki is a P2P network of a set of autonomous semantic wiki servers called also peers, that can dynamically join and leave the network.

Every peer hosts a copy of all wiki pages where these pages embed semantic data and an RDF store. Every peer can autonomously offer all the services of a semantic wiki server. Swooki supports massive collaboration, improves data availability and has a high performance thanks to its total replication of shared

[2] `http://git.or.cz/`

[3] `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#undoing-a-merge`

data. It allows to query and access data locally without any data transfer. In addition, it enables off-line works and transactional changes.

As in any wiki system, the basic element is a wiki page and every wiki page is assigned a unique identifier $PageID$, which is the name of the page. The name is set at the creation of the page. If several servers create concurrently pages with the same name, their content will be directly merged by the synchronization algorithm. An URI can be used to unambiguously identify the page. The URI is global and location independent.

A semantic wiki page $Page$ is an ordered sequence of semantic wiki lines. A semantic wiki line L is a four-tuple < LineID, content, degree, visibility > where $LineID$ is a unique line identifier in the whole network, *content* is a string representing text and the semantic data embedded in the line. *degree* is an integer used by the synchronization algorithm. *visibility* is a boolean representing whether the line is visible or not. Lines are not deleted physically, they are just invisible in the view of the semantic wiki page.

Changes in semantic wiki pages are detected as operations. An operation is either an insert operation $Op = Insert(PageID,\ line,\ l_P\ ,\ l_N)$ or a delete operation $Op = Delete(PageID,\ LineID)$ where l_P and l_N are the previous and the next lines of the inserted or the deleted line. An update of a line is considered as a delete of the old value followed by an insert of a new value.

A Swooki peer is composed of the following components (see figure 2):

User Interface. The Swooki UI component is basically a regular wiki editor. It allows users to edit a view of a page by getting the page from the Swooki manager. Users can disconnect their peer to work in an off-line mode and they can add new neighbors in their list to work with. In addition, the UI allows users to see the history of a page, to execute semantic queries and to export the semantic annotations of the wiki pages in an RDF format. The history of a page is the set of events concerning that page on a peer.

Swooki Manager. The Swooki manager is responsible for the generation and the integration of the editing patches. A *patch* is the set of delete and insert operations on the semantic wiki page. Patches are stored in a *patchGraph*. The SWooki manager implements the Swooki algorithm [17]. Requesting and modifying a page or resolving a semantic query in the RDF repository pass through this manager.

Sesame Engine. The RDF repository used in Swooki is Sesame 2.0 [5]. Sesame is controlled by the Swooki manager for storing and retrieving RDF triples. We used a facility of Sesame to represent RDF triples as multi-set. This component allows also generating dynamic content for wiki pages using queries embedded in the wiki pages. It provides also a feature to execute semantic queries and to export RDF graphs.

Diffusion Manager. The diffusion manager maintains the membership of the unstructured network and implements a reliable broadcast. This component is described in [21, 28].

The integration of the undo mechanism in Swooki requires the addition of the undo component, with slightly extensions of some existing elements. The undo

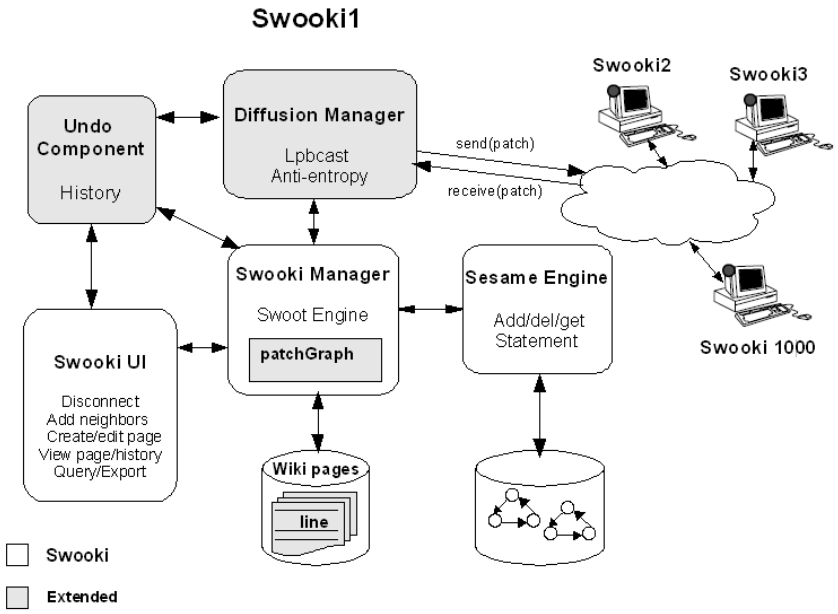


Fig. 2. Swooki architecture extended with the undo component

mechanism is designed to allow users to remove or reinsert the effect of some changes in the wiki pages and consequently to update their semantic annotations in the RDF repository. A detailed overview of the proposition is given in the following section.

5 Proposition

We developed the undo component for Swooki to provide users a feature to handle vandalism, to correct errors and to improve easily undesired result of an automatic merge done by Swooki.

5.1 Undo component

In this section, we describe the behavior of the undo component which is responsible of handling undo actions.

When a user wants to undo a modification, i.e. a patch, the document must return to a state in which the modification was never performed according to the undo definition see section 2. This definition implies two cases:

– the patch is already undone and the document must not be changed,

– the patch must be disabled and its effect must be removed.

Moreover, since the action of undoing a patch is also a modification of the document, users must be able to undo an undo modification, also called redo.

Similarly, according to the undo definition, if a patch is already redone, the action of "redo" has no effect, otherwise, we must re-apply its effect.

As a result, the system must know if a patch is enabled or not. Moreover, the system has to know how many times a patch has been undone or redone as illustrated in figure 3.

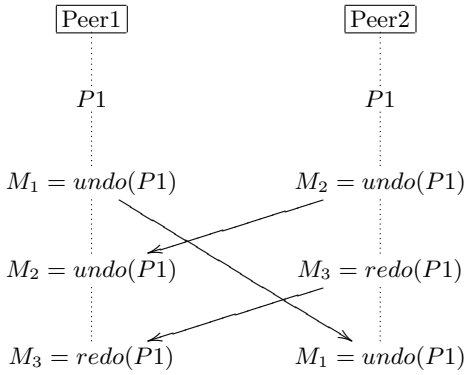


Fig. 3. Concurrent undo and redo messages

Assume that two sites, called Peer1 and Peer2, have received the same patch $P1$. Concurrently, both sites decide to undo this patch. Consequently, Peer1 generates a modification M_1 while Peer2 generates M_2. Then, Peer2 chooses to redo the patch P1. Finally, each peer receives each other modifications. Peer1 has received both "undo" modifications and then the "redo". If the system just knows that P1 is undone, the "redo" will reapply P1. Unfortunately, this example violates the definition. Indeed, if the modification M_2 was never produced, the only remaining modification is M_1, then the P1 must remain undone.

For instance, a patch with a *patchDegree* equals to three implies that the patch has been redone three times and that it has an effect in the semantic wiki page model. Actually, patches are never deleted from the *patchGraph*, however their effect is removed from the wiki page model as if they did not exist.

As a result, we propose to extend the *patchGraph* as defined in Swooki by associating a *patchDegree* to each *patch* in that graph. This *patchGraph* becomes a set of <patch, patchDegree> where the *patchDegree* indicates whether the patch has an effect or not in the current state of the wiki page. We arbitrary choose to affect a degree of 1 at the patch reception, to decrease by one the degree at the execution of an undo and increase it by one for a redo. Then, a patch is undone if its degree is strictly less than to 1.

Consequently, in Figure 3, Peer1 will compute a degree of 0 and will not restore P1.

Finally, we translate the description of the undo component above into the following algorithm:

```
integrateUndo(patchId):
    patch := getPatch(patchId)
    decreasePatchDegree(patch)
    if getPatchDegree(patch) = 0 then
       disable(patch)
```

```
integrateRedo(patchId):
    patch := getPatch(patchId)
    increasePatchDegree(patch)
    if getPatchDegree(patch) = 1 then
       enable(patch)
```

Now, the system can compute whether or not a patch has to be reapplied or undone. Next, we will explain how to remove or reapply a patch.

5.2 Removing and reapplying a patch

Since we keep deleted lines as tombstones in Swooki, we propose to reuse them instead of inserting new lines in case of redo. We call deletion the transformation of a line into a tombstone and, reinsertion the inverse of deletion.

Moreover, using the inverse operations as shown in Figure 1 is not sufficient for removing the effect of a patch. This is mainly due to concurrency: the line "It has a [topSpeed:=49km/h]." is deleted twice (by M_1 and M_2) and "reinserted" once by "Inverse(M_2)". Similarly to the previous example, we propose to count the number of deletions/reinsertions to overcome this issue.

In Swooki, a line in the wiki page model is never deleted, it is only set to invisible in the wiki page view. The visibility of a line is determined through a boolean visibility field. We change the line visibility as defined in Swooki into a visibility degree in order to let the system detect whether a line is visible or not after multiple undo and redo of patches. In our case, a line is visible if it is visibility degree is positive. A delete of a line turns its visibility degree to zero, however an undo (or a redo) action decreases (respectively increases) it by one.

Since we have changed the data model, we have to redefine the operations to modify it. In Swooki, the integration of an operation is processed in two steps: (1) text integration and (2) RDF statements integration. To integrate an insert operation $op = insert(PageID, line, l_P, l_N)$, the $line$ has to be placed among all the lines between l_P and l_N. Finding the right position where the line should be inserted is done through the Woot algorithm (for more details see [14]).

Once the right position is found, we insert the line in the page with a degree of 1 and insert the metadata into the RDF store:

```
insertLine( line ):
```

```
  insert (PageID, line,  NextIdentifier )
  integrateInsRDF(line)
```

Due to concurrency, a line can have a degree greater than 1. Therefore, the execution of a delete consists in decreasing the degree of that line. If the line becomes invisible, i.e. its degree is zero, we have to update the RDF store using the method "integrateInsRDF".

```
delLine(lineID ):
  line  := getLine(lineID)
  decreaseVisibilityDegreeOfLine( line )
  if  visibilityDegree ( line ) = 0 then
      integrateDelRDF(getContent(line))
```

Similarly, the reinsertion of a line increases its degree. If the line becomes visible, we update the RDF store.

```
reinsertLine (lineID ):
  line  := getLine(lineID)
  increaseVisibilityDegreeOfLine ( line )
  if  visibilityDegree ( line ) = 1 then
      integrateInsRDF(getContent(line))
```

Finally, we can now remove a patch effect:

```
disablePatch(patch):
  for op in patch do
       line  := getLine(op)
       switch(type(op))
            " insert ":  delLine( line )
            " delete ":  reinsertLine ( line )
```

or reapply its effect:

```
enablePatch(patch):
  for op in patch do
      line  := getLine(op)
          switch(type(op))
               " insert ":  reinsertLine ( line )
               " delete ":  delLine( line )
```

In Figure 1, since the third line is deleted twice, its degree is -1. As a consequence, when Peer1 undoes M_2, the line degree is increased by one and the line remains invisible.

5.3 Messages

As usual modifications, the undo/redo actions must be propagated to all other sites. Therefore, we need to extend the diffusion manager to take account of undo/redo messages. We define three kind of messages:

Do message: In case of a do message (i.e. containing only insert or delete operations), the patch is added in the *patchGraph* with a *patchDegree* equals to one. The operations of that patch are integrated depending on their type.

Undo message: In case of an undo message, the patch on which the undo message will be applied is extracted from the *patchGraph*;

Redo message: Similarly for a redo message, the patch is extracted from the *patchGraph*.

When a message is received, it is added into a waiting queue. Then each message in that waiting queue is tested if it is executable or not. A *do* message is executable if its operations satisfy their preconditions as defined in [13], however an *undo* or a *redo* message is executable if the patch on which it is applied exists in the *patchGraph*. In case of an executable message, its message information $mInfo$ is added into the page history. Then the message is integrated depending on its type. The algorithm stops when the waiting queue does not contain any executable message.

```
uponReceive(message):
  addMsgWaitingQueue(message)
  stop := false
  while(stop = false) do
      stop := true
      for msg in MsgWaitingQueue do
          if isExecutable(msg)= true then
              stop := false
              writeHistoryEvent(mInfo)
              switch(type(msg))
                  " Do":
                      addInPatchGraph(getPatch(msg))
                      for op in patch do
                          if type(op) = insert
                              integrateIns (op)
                          else
                              integrateDel(op)
                      done
                  " Undo":
                      disablePatch(getPatch(msg))
                  " Redo":
                      enablePatch(getPatch(msg))
          endif
  endif
```

6 Implementation

The undo or redo of changes can take place either by visiting the history or the patch graph of a page. The figure 4 shows the history where different messages are integrated on the wiki page. A line in the history is equivalent to the message

information. It indicates the identifier of the patch, the peer that generated the patch, the type of the message and a user comment when it exists. The undo action of a patch has a grey background. In order to undo a set of patches, a user can click the checkbox that precedes each one of them and then press the undo preview link. The result of this preview is a temporary version of the page which undoes these patches. If he is satisfied, he can apply these changes by saving the undo preview.

Another option provided in the history is to revert the current version of the wiki page. Users can choose to return to a state of a page undoing all the changes integrated after the chosen patch. This is can be achieved by selecting the patch and clicking on the revert preview. Similarly, if he is satisfied the user can save the revert preview and his changes will be applied. The undo of each patch inserts a new line in the history. The history allows also providing more information about each patch by a click on show details link at the end of each line. Each undo action in the history generates a new message of type undo. This message is sent through Swooki diffusion manager to the other peers in order to be integrated. The integration of that message locally or on the other peers is done as defined in section 5.

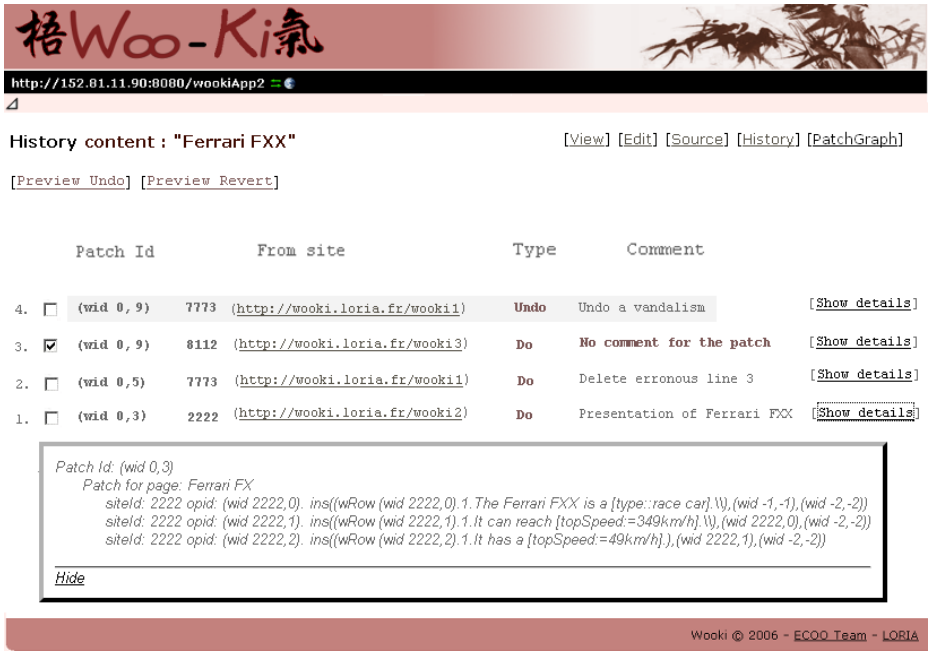


Fig. 4. Undo from the history

Another way to undo or redo a patch is through the patch graph (see Figure 5). The patch graph is viewed as an oriented graph of patches. Each patch

is a node in the graph and the arrows represent the dependence between these patches. A node relating one or more nodes implies that this patch was generated on a state integrating these patches. Each node is labeled with the patch identifier and the patch degree. A black node is a disabled patch that has no effect in the wiki page. A right click on a node allows to show the information about the patch, to undo or redo a patch or to preview a version of the wiki page. The patch graph is visualized through an applet built using JGraphT java libraries and JGraph is used to render the graph layout. It is based on a recursive algorithm browsing the patch graph [1]. The patch graph provides information about the patches dependency, hence the concurrency between them.

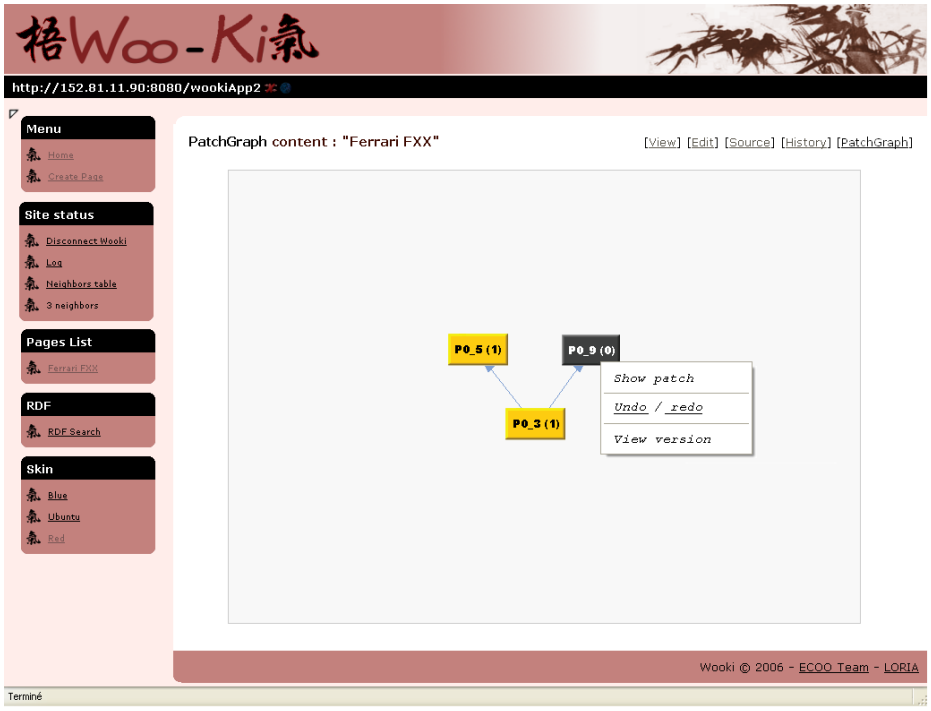


Fig. 5. Undo from the patch graph

7 Conclusion

In this paper, we propose an undo mechanism for Swooki. This mechanism allows users to undo any modification at any time, i.e. to remove any modification as if it was never produced. It provides both an undo and a revert features. We developed the undo component, the appropriate algorithms and extended some parts of Swooki to provide it with an undo mechanism. The undo component is

responsible for the generation and the integration of undo and redo actions. The propagation of these actions lies on Swooki diffusion manager. Swooki extension ensures the convergence of the wiki pages and the RDF stores on all peers. This convergence is independent from concurrent modifications, the order of integration of the undo or redo actions and the fact that users can edit in an off-line mode i.e. join or leave at anytime.

We identified the problems to resolve in order to achieve such mechanism for P2P semantic wikis. While the revert feature may be sufficient for centralized semantic wikis, this is not the case for P2P semantic wikis aiming at removing any modification at any time. Our solution is general, it is based: (1) on enabling/disabling patches in the patchGraph and (2) on the generation and the integration of undo/redo actions. It can be adopted in any P2P semantic wiki and any P2P wiki.

As future work, we intend to carry out user studies to evaluate: (1) the quality improvement of wiki pages and the knowledge in the RDF stores using our approach and (2) how this mechanism facilitates the task of the users compared to Swooki without the undo mechanism.

References

1. S. Alshattnawi, G. Canals, and P. Molli. Concurrency awareness in a p2p wiki system. In *Proceedings of CTS 2008, The 2008 International Symposium on Collaborative Technologies and Systems, Irvine, California, USA*, May 2008.
2. S. Auer. Powl. In *Proceedings of the 1st Workshop on Scripting for the Semantic Web (SFSW'05), Hersonissos, Greece*, 2005.
3. S. Auer, S. Dietzold, and T. Riechert. Ontowiki - A tool for social, semantic collaboration. In *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.
4. D. Aumueller and S. Auer. Towards a semantic wiki experience - desktop integration and interactivity in wiksar. In *Proceedings of the Workshop on Semantic Desktop, Galway, Ireland*, 2005.
5. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *ISWC 2002: First International Semantic Web Conference*, 2002.
6. M. Buffa, F. L. Gandon, G. Ereteo, P. Sander, and C. Faron. Sweetwiki: A semantic wiki. *Journal of Web Semantic*, 6(1):84–97, 2008.
7. B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht. Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis. *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE), November 6th-10th*, 2005.
8. K. Dello, E. P. B. Simperl, and R. Tolksdorf. Creating and using semantic web information with makna. In *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics*. ESWC2006, June 2006.
9. C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In J. Clifford, B. G. Lindsay, and D. Maier, editors, *SIGMOD Conference*, pages 399–407. ACM Press, 1989.

10. T. Kuhn. Acewiki: Collaborative ontology management in controlled natural language. *In Proceedings of the 3rd Semantic Wiki Workshop, CEUR Workshop Proceedings, 2008*, Jul 2008.
11. F. Mattern. Virtual time and global states of distributed systems. In M. C. et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Château de Bonas, France, october 1989. Elsevier Science Publishers.
12. J. C. Morris. Distriwiki: : a distributed peer-to-peer wiki network. In *Int. Sym. Wikis*, pages 69–74, 2007.
13. G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for P2P collaborative editing. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, Banff, Alberta, Canada, November 2006.
14. G. Oster, P. Urso, P. Molli, and A. Imine. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *The Second International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*, Atlanta, Georgia, USA, November 2006. IEEE Press.
15. A. Prakash and M. J. Knister. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(4):295–330, 1994.
16. C. Rahhal, H. Skaf-Molli, and P. Molli. Swooki: A peer-to-peer semantic wiki. In *The 3rd Workshop: 'The Wiki Way of Semantics'-SemWiki2008, co-located with the 5th Annual European Semantic Web Conference (ESWC), Tenerife, Spain*, June 2008.
17. C. Rahhal, H. Skaf-Molli, and P. Molli. Swooki: A peer-to-peer semantic wiki. Research Report 6468, INRIA, Mars 2008.
18. M. Ressel and R. Gunzenhäuser. Reducing the problems of group undo. In *GROUP*, pages 131–139, 1999.
19. Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, 2005.
20. S. Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *WETICE*, pages 388–396. IEEE Computer Society, 2006.
21. H. Skaf-Molli, C. Rahhal, and P. Molli. Peer-to-peer semantic wikis. Research report, INRIA, 2008.
22. A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5):87–91, 2005.
23. C. Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(4):309–361, December 2002.
24. C. Sun and C. A. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Proceedings of CSCW*, pages 59–68, New York, New York, États-Unis, Novembre 1998. ACM Press.
25. D. Sun and C. Sun. Operation Context and Context-based Operational Transformation. In *Proceedings of CSCW*, pages 279–288, Banff, Alberta, Canada, November 2006. ACM Press.
26. G. Tummarello, C. Morbidoni, and M. Nucci. Enabling semantic web communities with dbin: An overview. *The Semantic Web - ISWC 2006*, pages 943–950, 2006.
27. M. Vôlkel, M. Krtôzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. *Journal of Web Semantics*, 5(4), 2007.
28. S. Weiss, P. Urso, and P. Molli. Wooki: a p2p wiki-based collaborative writing tool. In *Web Information Systems Engineering*, Nancy, France, December 2007. Springer.
29. S. Weiss, P. Urso, and P. Molli. An undo framework for p2p collaborative editing. In *CollaborateCom*, Orlando, USA, November 2008.

Enabling cross-wikis integration by extending the SIOC ontology

Fabrizio Orlandi[1] and Alexandre Passant[2]

[1] Università degli studi di Modena e Reggio Emilia,
Modena, Italy
orlandi.fabrizio.31985@unimore.it
[2] Digital Enterprise Research Institute,
National University of Ireland, Galway
alexandre.passant@deri.org

Abstract. This paper describes how we extended the SIOC ontology to take into account particular aspects of wikis in order to enable integration capabilities between various wiki systems. In particular, we will overview the proposed extensions and detail a webservice providing SIOC data from any MediaWiki instance, as well as related query examples that show how different wikis, designed as independant data silos, can be uniformally queried and interlinked.

Key words: Semantic Web, SIOC, wikis, MediaWiki, Social Semantic Web, Linked Data, DBpedia

1 Introduction

The SIOC Ontology – Semantically-Interlinked Online Communities [4] – is now considered as one of the building blocks of the *"Social Semantic Web"*. More than 50 applications are currently using SIOC[3], either as a common vocabulary to expose their data in RDF, alongside with FOAF for instance, as well as using existing SIOC data, as for instance Yahoo! SearchMonkey. Moreover, the use of SIOC goes further than mainstream Web 2.0 services, from Enterprise 2.0 information integration[4] to Health Care and Life Sciences discourse representation[5].

However, only a few work have been done so far regarding wikis, semantic wikis and the SIOC ontology. While the SIOC Types[6] module already provides the `Wiki` and `WikiArticle` classes that can be used to represent the basic objects manipulated by wikis, some particular features of wikis such as pages versioning and backlinks are not taken into account, neither in the SIOC core nor in its modules. Yet, providing wikis information using SIOC would have several advantages in terms of integration with existing and constantly dynamically-created

[3] http://sioc-project.org/applications
[4] http://www.w3.org/2001/sw/sweo/public/UseCases/EDF/
[5] http://esw.w3.org/topic/HCLSIG/SWANSIOC
[6] http://rdfs.org/sioc/types

SIOC data, as well as interlinking with other RDF data for advanced querying purposes. For instance, one will be able to run the same SPARQL query to find latest created items on a MediaWiki instance or on a WordPress weblog. Hence, we recently worked on extending the SIOC ontology for this purposes, as well as providing a SIOC exporter for MediaWiki, potentially creating millions of SIOC-based RDF documents from various popular wiki services.

This paper is organized as follows. First, we will go through an overview of wiki features that are important to consider in such a modeling approach and explain how we took them in consideration in regards of the SIOC ontology and how we extended it based on this analysis. Second, since some wikis already expose their data in a machine-readable form thanks to Semantic Web technologies, we will focus on a state of the art of existing models that achieve the same goal. Then we will detail how we built a webservice that translates any MediaWiki wiki page to RDF using the newly-extended SIOC ontology. We will particularly focus on how this service produces RDF data compliant with the Linked Data principles and how it relates to initiatives such as DBpedia. Then we will show some relevant query examples, from advanced queries in a single wiki to cross-querying capabilities. Finally, we will conclude the paper with an overview of future works on the domain.

2 Using and extending the SIOC ontology for advanced modeling of wiki structure

In this section we spot and explore what we consider being the typical and fundamental features of wikis in terms of structure and social interactions. Typically wikis allow editing of documents and, by definition, allow multiple users to simultaneously contribute to the content; they track history of changes so that pages can be restored to previous modified versions; they include comments or discussion areas; they link to other external sources or within the wiki; they describe categories into hierarchical structures. For each identified feature, we give a brief overview of its goal, and detail how we extended, and generally use, the SIOC Core ontology[7] and its Types module[8] taking each feature into account in our modeling approach.

2.1 Modeling relevant wiki features

Multi-authoring. A fundamental feature of wikis is that multiple users are allowed to modify the same content, enabling some kind of collective intelligence process. In this regard the semantic infrastructure should provide a model to identify users and theirs modifications, marking events with a corresponding timestamp.

This feature can be modeled using the class `sioc:User` as object of the property `sioc:has_creator` that describes a user account in an online community

[7] `http://rdfs.org/sioc/ns`, prefix `sioc` in this document
[8] `http://rdfs.org/sioc/types`, prefix `sioct` in this document

site, and which is a subclass of `foaf:OnlineAccount`. In this way a `foaf:Person` could be linked to several `sioc:User` belonging to different wiki sites. Another way to model the relationships between pages and their authors is to reuse properties from the Dublin Core ontology, i.e. `dc:contributor` (or `dc:creator`) and `dcterms:created`. Yet, these properties do not link to a user URI but to a simple text string, which can be an issue when querying information, especially for cross-querying as we will detail in section 5.

Categories. Wiki pages are generally related to categories, that allow readers to find sets of articles on related topics. Categories can also be organized in a tree-like structure and their semantic model should mantain the original taxonomical structure. In this regard an appropriate solution is provided by the SKOS[9] vocabulary [11], as it offers a way to model hierarchical structures between various categories, represented as instances of `skos:Concept`.

As regards the SIOC ontology, a `sioct:Category` class was already present into the SIOC Types module, allowing only the modeling of a flat set of category names. Hence, we decided to declare this class as a subclass of a `skos:Concept`, giving it the ability to use the wide SKOS ontology capabilities to organize categories into advanced taxonomies. Moreover, thanks to the `sioc:topic` property, one can link any wiki page to such category.

Social tagging. While not all wiki engines support that feature, we believe this is particularly relevant, especially as it offers an open and user-driven classification scheme for wiki pages. The use of tags lead to a non-organised but dynamic organisation process, known as a "folksonomy", rather than the more widely used hierarchical structures.

The properties `sioc:topic` and `dc:subject` can be used to represent tags related to a particular wiki page, either using URIs for these tags (with `sioc:topic`) or simple keywords (`dc:subject`). In addition, vocabularies such as the Tag ontology [12], SCOT [8] or MOAT [13] allow to model tagging as tripartite actions (between a wiki page, a user and a tag) as well as organize tags together or link them to ontology concepts, in order to solve common tagging issues such as ambiguity between tags.

Discussions. Several wikis associate a discussion page to every wiki page, so that each user is able to comment and argue his point-of-view on the topic. On a discussion page, people can discuss about the article subject, or about the way that subject is presented (see the Wikipedia's approach[10]). A first modeling solution could be to simply keep the native wiki text format of the wiki and just semantically link the discussion page to the related article page.

The SIOC's main class responsible for the modeling of a discussion is the `sioc:Forum` class, but there could be other specific classes that are more suitable

[9] `http://www.w3.org/2004/02/skos/`
[10] `http://en.wikipedia.org/wiki/Wikipedia:Talk_page_guidelines`

for these discussion purposes, as defined in the Types module. The appropriate class to choose depends also on the type and style of the discussion page. So it has been necessary to identify a proper attribute to link a wiki page to its discussion page. In this regard we decided to add a `sioc:has_discussion` property to the SIOC Core ontology, with domain `sioc:Item` and open range. This choice has been done in order to make this property reusable also in other contexts, for instance linking a simple webpage to a discussion forum. The discussions happening within the related `sioc:Forum` can then be modeled either as wiki-style discussions or threaded ones, and that feature also allows us to re-use advanced SIOC-based argumentative discussion modeling as defined in [10].

Backlinks. Backlinks are an important feature of wikis, as they allow to visualize instantaneously all the incoming links to a website or web page. More precisely they are wiki internal links pointing to a wiki article. It is a very common wiki feature and they may be of significant interest: they indicate who is paying attention to the linked page or topic.

We modeled this feature using the already existing `sioc:links_to` property. This property identifies links extracted from hyperlinks within a SIOC concept and is a subproperty of `dcterms:references`. It is important to remember that this property has to be defined into the RDF description of the original wiki article which links back to the wiki article. Hence, to model for instance that the Wikipedia page about "DERI" features a backlink from the page about "RDF", the following statement would be added into the RDF description of DERI's page.

```
<http://en.wikipedia.org/wiki/
   Resource_Description_Framework> sioc:links_to
 <http://en.wikipedia.org/wiki/
    Digital_Enterprise_Research_Institute> .
```

Listing 1.1. Representing backlinks

Pages versioning. Usually all editable pages on wikis have an associated page history. This history consists of the old versions of the wikitext, as well as a record of the date and time of every edit, the username or IP address of the user who wrote it, and their edit summary. All this is usually accessible through a special *"history"* page which shows time-ordered links to all the revisions. Commonly the latest revision of a wiki page has always the same URL (alias name), meanwhile older versions have further parameters appended to the URL.

The versioning of pages could be modeled in several ways. Taking into account other semantic wikis, that we describe in Section 3, we took inspiration from other existing approaches. Then we defined our own different model because we wanted to keep the pros of each model and we did not find one capable to satisfy completely our needs. An important requirement we take into account is the fast and simple browsing capability that the model should have. For this

reason we chose to use transitive properties to express the temporal relation between revisions of a wiki page. The model is displayed in Fig. 1 and all the used properties are now defined in the SIOC Core ontology.

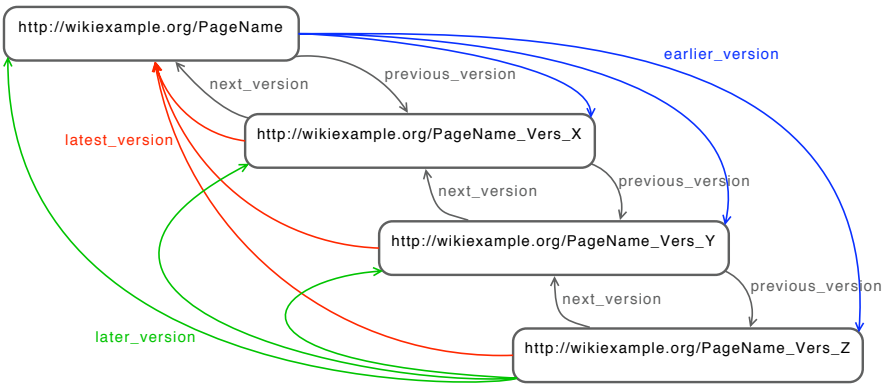


Fig. 1. Pages versioning model with SIOC properties. Please note that, for more clarity, transitive properties `earlier_version` and `later_version` are only displayed for two wiki articles: the latest one and the first one.

The `next/previous_version` properties link only the next/previous revision of a generic `sioc:Item`. Meanwhile `earlier/later_version` are defined as transitive properties and as super-properties respectively of `next/previous_version`. The main advantage of the definition of transitivity and the declaration of super-properties, is that they can be inferred automatically by a reasoner. Hence, using an OWL level reasoning engine, when modeling a `WikiArticle` (or a `sioc:Item` in general), it is only necessary to describe its previous and next revision and the transitive super-properties will be automatically inferred by the system. This can also be convenient during the querying process (described in Section 5): if it is necessary to get all the earlier versions of a wiki page, with transitivity it is sufficient to use the `sioc:earlier_version` transitive property, while in the other case, it has to be implemented a query that recursively "jumps" on each `sioc:previous_version` of the latest wiki article. Another introduced property is the `sioc:latest_version` which points always to the newest revision. Usually it is used in combination with an alias name of the latest version so that it is not necessary to change the referred URI in all the earlier versions as soon as a modification happens. All the wikis we analyzed adopt this solution as it addresses scalability.

2.2 Changes summary

All the changes we made to the SIOC ontology are summarized as follows:

– Defined the `sioct:Category` class as a subclass of `skos:Concept`.

- Added a `sioc:has_discussion` property, with domain `sioc:Item` and open range.
- Added a `sioc:latest_version` property, with `sioc:Item` as domain and range.
- Added two transitive properties: `sioc:earlier_version` and `sioc:later_version`, with `sioc:Item` as domain and range.
- Defined `sioc:later_version` as inverse property of `sioc:earlier_version`.
- Defined `sioc:next_version` as a subproperty of `sioc:later_version`.
- Defined `sioc:previous_version` as a subproperty of `sioc:earlier_version`.

3 Related work

3.1 Existing models to represent structure of wikis

While our aim is to model wiki features by extending SIOC, several vocabularies have been already proposed to achieve this goal. We will overview some of them in this section, by distinguishing models created with a general purpose and models created for a particular wiki engine but nevertheless available on the Web. In addition, it is worth noticing that we focus here on models and tools defined to represent the structure of wikis and not on the ones that allow domain ontology modeling and population in RDF(S)/OWL from the wiki itself, while these two levels of semantics can be obviously combined.

As regards generic models **WikiOnt**[11] [7] is an ontology for describing and exchanging wiki articles and it aims at integrating Wikipedia (and by extension other MediaWiki-based sites) into the Semantic Web framework, making Wikipedia machine-processable. This OWL ontology uses DublinCore to identify multiple authors of wiki pages as well as the editing date, and provides `Article` and `Category` classes. Yet all the other features are currently not modeled by the ontology. **Wiki Interchange Format**[12] (WIF) [16] is a project that allows data exchange between wikis and related tools. Different from other approaches, it also tackles the problem of page content and annotations. WIF defines a subset of XHTML as an over-the-wire format for wiki content exchange. It defines the classes of `WikiUser` (subclassing `foaf:Person`) and `WikiPage` to model pages and authors. It also privides a versioning system thanks to the `hasPreviousVersion` and `hasChangeDate` properties. Categories, social tagging, discussions and backlinks are features currently not modeled by the ontology.

In addition to the previous models we considered some particular wiki engines that expose their data in RDF providing an RDF vocabulary for such export. **IkeWiki**[13] [15] aims at creating instance data based on an existing ontology but also at being a tool for creating and editing ontologies. In addition, IkeWiki provides a complete export of the wiki structure using a dedicated OWL ontology. It introduces a `User` class, subclass of `foaf:Person`, and uses a `hasAuthor`

[11] `http://sw.deri.org/2005/04/wikipedia/wikiont.html`
[12] `http://wif.ontoware.org/2005/04/`
[13] `http://ikewiki.salzburgresearch.at`

property, subclass of `foaf:maker`, to associate an author/User with a resource in the wiki. It is also worth noticing that IkeWiki uses SIOC to model discussions, using a `hasDiscussion` property that links any wiki page to an instance of `sioc:Forum`. A wiki article is defined by a `Page` class. Social tagging, backlinks and pages versioning are features currently not modeled by the ontology.
SweetWiki[14] [6] is a semantic wiki based on the CORESE engine. It relies on Web standards for the semantic annotations (RDFa, RDF) and for the ontologies it manipulates (OWL Lite). The SweetWiki ontology manages versions of wiki pages using a `Version` class, defined as a subclass of the main `WikiPage` class. A `pageHasVersion` property links each old version with the latest page represented by the `WikiPage` class. The page version number is declared as an integer number with the `isTheVersionNumber` attribute. To note that SweetWiki offers advanced social tagging features, which are not modeled by other Semantic wikis. Keywords can be collaboratively structured through a lightweight ontology editor and related either to pages, categories (defined using a particular `Category` class) or embedded media content. Sweetwiki also supports backlinks, but not discussion pages at the moment. It also defines its own classes to model authors and wiki pages, respectively using `Person` and `WikiPage`.
Semantic MediaWiki[15] (SMW) [9] uses a particular ontology to represent the semantic data exported from a page by a user, named *SWIVT* – Semantic Wiki Vocabulary and Terminology. This ontology provides a basis for interpreting the semantic data exported by SMW, and it incorporates various elements that are closely related to SMW's metadata model. Yet, while some features such as backlinks or categories are provided by SMW, they are not exposed in the SWIVT model, exporting only a `Wikipage` class.

Finally, regarding the use of SIOC for wikis, we can mention UfoWiki [14], that have been deployed in complement of other SIOC-related data in an Enterprise 2.0 platform, while it does not support versioning in its RDF representation.

3.2 Comparison and positioning of our approach

Based on the previous analysis, we produced a comparison matrix, to underline the pros and cons of each approach. We may conclude that multi-authoring is a feature supported by the ontologies of all the wiki models, and this is because it is an inescapable characteristic of a semantic wiki. On the other hand backlinks and versioning are not modeled by most of the considered wikis. These two features are addressed only by SweetWiki, with the exception of WIF developing a very simple versioning solution. Moreover, some features that are not modeled by these vocabularies could be added by external vocabularies, as for instance social tagging. Finally the most complete model to take into account is SweetWiki, being able to accomplish to every requirement but the discussions.

[14] `http://sweetwiki.inria.fr/sweetwiki`
[15] `http://semantic-mediawiki.org`

×	IkeWiki	SweetWiki	SWIVT	WikiOnt	WIF	SIOC
Multi-authoring	√	√	√	√	√	√
Categories	√	√	√	√	×	√
Social Tagging	×	√	×	×	×	√
Discussions	√	×	×	×	×	√
Backlinks	×	√	×	×	×	√
Versioning	×	√	×	×	√	√

Table 1. Comparing various ontologies to represent wikis structure

4 Exporting SIOC data from MediaWiki

In order to see implications of our extension, our aim was to build an exporter from a popular wiki platform, so that it can expose its data in RDF using our proposed model. Hence we decided to create a web-service application to export any MediaWiki instance. MediaWiki is one of the most popular wiki platforms, hosting all the Wikimedia Fundation wikis (i.e. Wikipedia, Wiktionary, etc.) and propulsing more that 25 millions of wiki articles from different wiki sites[16].

4.1 Principles of the SIOC-MediaWiki webservice

In order to export SIOC data from MediaWiki's wikis we implemented a web-service, written in PHP, that exports a wiki article in RDF with the structure we explained in the previous sections. The webservice is publicly available at `http://ws.sioc-project.org/mediawiki/`.

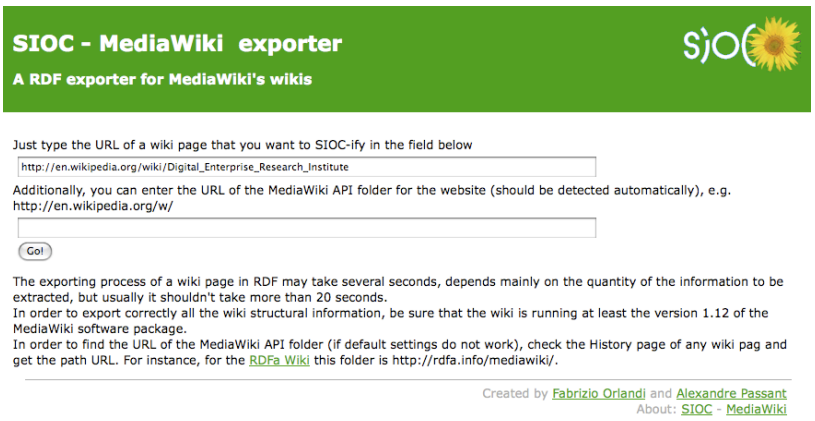


Fig. 2. Interface of the SIOC-MediaWiki webservice

[16] `http://s23.org/wikistats/largest_html.php`

The MediaWiki exporter is relatively lightweight and built thanks to two PHP classes: the SIOC-Mediawiki exporter itself and the already existing SIOC API[17]. Our approach combines the use of the MediaWiki API as well as the SIOC PHP API – that has been extended based on the previously-detailed ontology changes – to create SIOC data. The exporter class is the part responsible for querying the MediaWiki API and parsing the results, and the SIOC API is responsible for exporting the content in RDF. The script indeed uses the MediaWiki API to get all the information about the article inserted in the form, with the following process (as represented in Fig. 3):

- it automatically discovers the API location (if not detected it is possible to manually specify the API path in the proper text field);
- it connects to the API sending HTTP requests as queries;
- it parses the results of the queries and fills in the proper variables;
- it calls the SIOC API to export in RDF the fetched structural information and outputs the results in RDF/XML serialization.

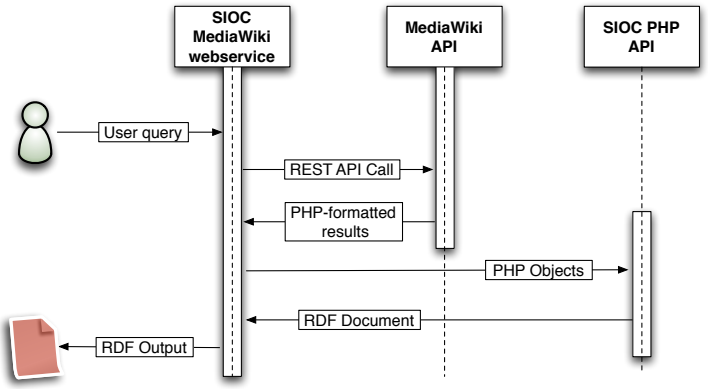


Fig. 3. Activity diagram of the SIOC-MediaWiki webservice

4.2 Following the Linked Data principles

One of our goals with this exporter was not only to create RDF data from any MediaWiki page, but also to easily allow interlinking between various wikis, as well as between wiki data and other RDF data, whatever it is social data modeled with FOAF or SIOC or any other kind of RDF data. Hence, we followed the Linked Data best practices defined in [1], [2] and [3].

Particularly, to offer a better browsing experience and ease the process of crawling SIOC exports of MediaWiki instances, our webservice automatically

[17] `http://wiki.sioc-project.org/index.php/PHPExportAPI`

produces `rdfs:seeAlso` links between wiki pages. Actually, more than a simple link to the wiki page, the exporter provides a link to the related RDF document, as we can see in the following Listing 1.2 related to a particular `sioc:User`. As we can also notice in that example, we distinguish the concept itself (i.e. `User:StefanDecker`) and the related RDF page. These `seeAlso` links are very useful not only to provide link to other related RDF documents, that can be used for instance when browsing data with Tabulator, but also in a crawling perspective. A RDF crawler could easily follow all the `seeAlso` links found on every document and continue to crawl. In this regard, for example, we crawled and exported entire wiki sites just following these links.

```
<sioc:User rdf:about="http://en.wikipedia.org/wiki/User:
    StefanDecker">
 <rdfs:seeAlso rdf:resource="http://ws.sioc-project.org/
     mediawiki/mediawiki.php?wiki=http://en.wikipedia.org
     /wiki/User:StefanDecker"/>
</sioc:User>
```

Listing 1.2. Modeling a user in the MediaWiki exporter

Another interesting feature is the linkage to the corresponding DBpedia[18] resource, if the article belongs to the english Wikipedia. Since DBpedia semantically models the content of a Wikipedia page, this connection is very useful to link semantic data about the content and the structure of a wiki article. DBpedia resource URIs are used in rage of the `foaf:primaryTopic` property, this because it relates a document to the main thing that the document is about.

5 Cross-wikis integration and advanced querying process

In order to evaluate our proposal, we exported and crawled different MediaWiki instances. Four different wikis have been crawled – using for each crawl a single entry point thanks to the use of the `rdfs:seeAlso` links – each one belonging to the same area of interest in order to have a high probability of shared topics and users: *Semanticweb.org*[19], *Protégé Wiki*[20], *RDFa Wiki*[21] and the *ONTOLORE Karlsruhe* wiki[22]. In total, we collected about 1GB of RDF data and loaded it in Sesame [5]. As we needed an higher degree of inference (because of the OWL transitive properties) we installed and configured the reasoning engine OWLIM[23] on the top of it.

[18] `http://www.dbpedia.org`
[19] `http://www.semanticweb.org`
[20] `http://protegewiki.stanford.edu`
[21] `http://rdfa.info/wiki/RDFa_Wiki`
[22] `http://logic.aifb.uni-karlsruhe.de/wiki/ONTOLORE`
[23] `http://www.ontotext.com/owlim/`

5.1 Advanced querying for a single wiki

A first example of advanced querying for a particular wiki is the ability to answer to the following question: "what are the collaborating users that worked alternatively on the same wiki article?". In Listing 1.3 we provide the SPARQL implementation of this query.

```
SELECT DISTINCT ?wikiArt ?Contrib_a ?Contrib_b
WHERE {
 ?x sioc:latest_version ?wikiArt.
 ?wikiArt sioc:earlier_version ?VersA .
 ?VersA sioc:earlier_version ?VersB ;
  dc:contributor ?Contrib_a .
 ?VersB sioc:earlier_version ?VersC ;
  dc:contributor ?Contrib_b .
 ?VersC dc:contributor ?Contrib_a .
 FILTER (?Contrib_a != ?Contrib_b) .
}
```

Listing 1.3. Identifying collaborating users

In Fig. 4 we display a diagram that summarizes the above query, meanwhile in Fig. 5 we show the results we got querying on our SESAME triplestore. As we can see, this query takes advantage of the transitivity of the newly created property `sioc:earlier_version`, since we identify users that worked on earlier versions, and not only immediately on the previous one.

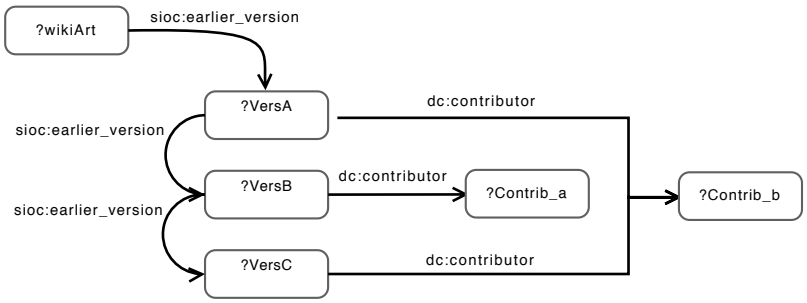
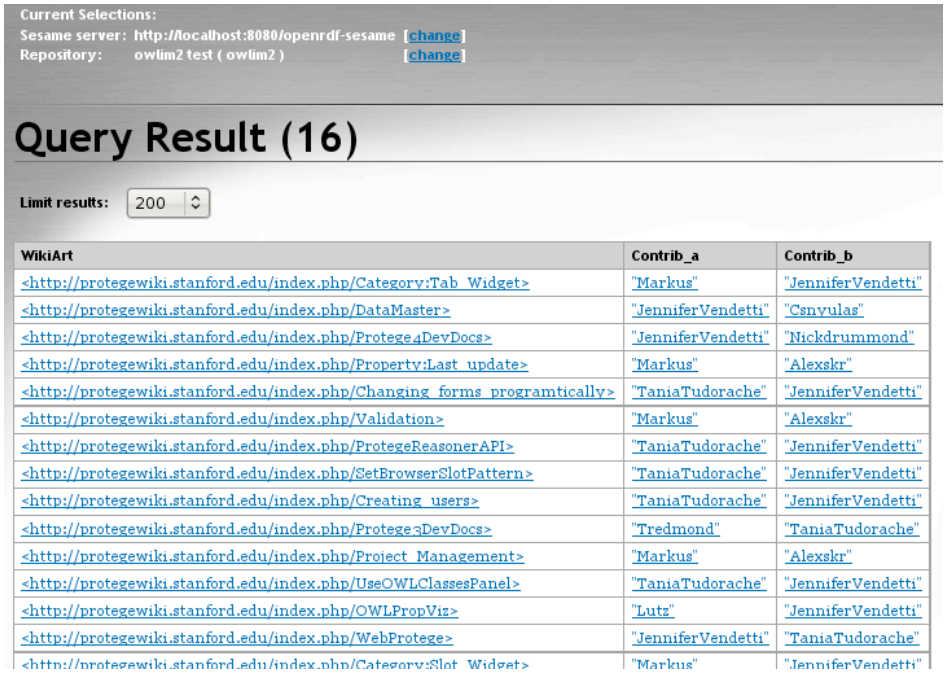


Fig. 4. Identifying collaborating users

The query results provide the article URI and the two usernames in case the first user (`?Contrib_a`) re-edited the article after a modification made by the second user (`?Contrib_b`). It enables you to look for users sharing the same interests and knowledge areas. The query is important expecially in a social semantic context.


Current Selections:
Sesame server: http://localhost:8080/openrdf-sesame [change]
Repository: owlim2 test (owlim2) [change]

Query Result (16)

Limit results: 200

WikiArt	Contrib_a	Contrib_b
<http://protegewiki.stanford.edu/index.php/Category:Tab_Widget>	"Markus"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/DataMaster>	"JenniferVendetti"	"Csnyulas"
<http://protegewiki.stanford.edu/index.php/Protege4DevDocs>	"JenniferVendetti"	"Nickdrummond"
<http://protegewiki.stanford.edu/index.php/Property:Last_update>	"Markus"	"Alexskr"
<http://protegewiki.stanford.edu/index.php/Changing_forms_programtically>	"TaniaTudorache"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/Validation>	"Markus"	"Alexskr"
<http://protegewiki.stanford.edu/index.php/ProtegeReasonerAPI>	"TaniaTudorache"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/SetBrowserSlotPattern>	"TaniaTudorache"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/Creating_users>	"TaniaTudorache"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/Protege3DevDocs>	"Tredmond"	"TaniaTudorache"
<http://protegewiki.stanford.edu/index.php/Project_Management>	"Markus"	"Alexskr"
<http://protegewiki.stanford.edu/index.php/UseOWLClassesPanel>	"TaniaTudorache"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/OWLPropViz>	"Lutz"	"JenniferVendetti"
<http://protegewiki.stanford.edu/index.php/WebProtege>	"JenniferVendetti"	"TaniaTudorache"
<http://protegewiki.stanford.edu/index.php/Category:Slot_Widget>	"Markus"	"JenniferVendetti"

Fig. 5. Results of the query for collaborating users on SESAME

5.2 Cross-wiki integration and querying

Another interesting feature of our approach is the ability to do cross-wikis querying, since they are based on the same model. Obviously, one can argue that since all the exported wikis are based on MediaWiki, the same approach could have been used simply with the MediaWiki API. Yet, our proposal has many advantages as it relies on SPARQL instead of a particular API and it provides advanced inference capabilities that the original API does not offer. The following query identifies users involved in different wikis, looking for the same usernames.

```
SELECT DISTINCT ?creator1 ?page1 ?page2 ?wiki1 ?wiki2
WHERE {
 ?page1 sioc:has_container ?wiki1 ;
  dc:contributor ?creator1 .
 ?page2 sioc:has_container ?wiki2 ;
  dc:contributor ?creator2 .
 FILTER (str(?creator1)==str(?creator2)) .
 FILTER (str(?wiki1)!=str(?wiki2)) .
}
```

Listing 1.4. Identifying pages created by a single user in different wikis

While this is a very simple query it requires high computation capabilities when ran through a large number of different wikis. Hence, in Fig. 6 we display

a screenshot of the results we get after running the same query between the *Semanticweb.org* wiki and the *Protégé* wiki. Instead of also displaying all the other details, such as the related wiki pages and the two wiki containers, we show only the distinct usernames of the found users.

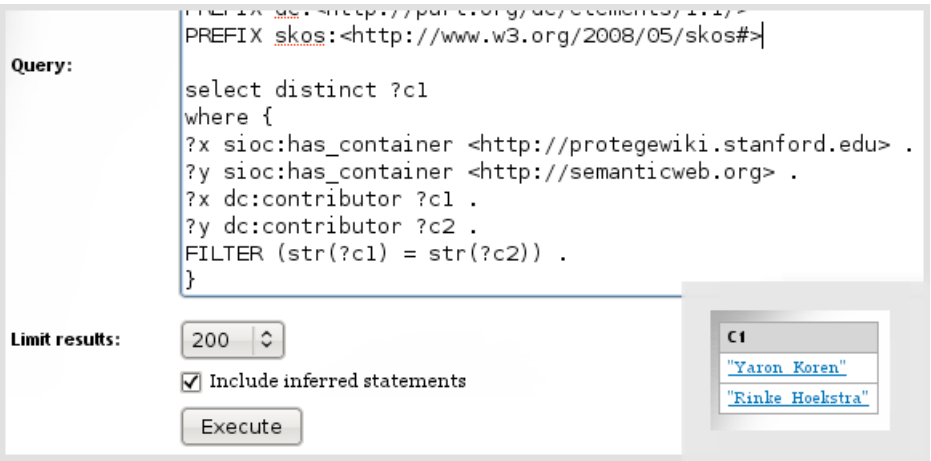


Fig. 6. Query results. Shared users between two wikis.

Yet, as this query relies on a FILTER clause, it will identify common users only if they use the same account name on two different wikis. Moreover, we can imagine that some common account names will be used by different people on different wikis, e.g. JohnSmith. To that extend, we can benefit from the strong ties that exist between FOAF and SIOC and the fact we are modeling a wiki user using the `sioc:User` class. One person can indeed define in his FOAF profile the various wiki accounts he owns, using simple `foaf:holdsAccount` properties. Then, the previous query can be adapted to deal not only with text strings to identify the user, but with their related accounts from the FOAF URI, so that a single query can be used to retrieve all the contributions of a user whatever the wiki used was. Moreover, since the wiki model is based on SIOC, the same query can be used to retrieve wiki pages, blog posts, etc. as follows.

```
SELECT DISTINCT ?content
WHERE {
 <http://example.org/js#me> foaf:holdsAccount ?account .
 ?account rdf:type sioc:User .
 ?content sioc:has_creator ?account .
}
```

Listing 1.5. Cross-sites querying by combining FOAF and SIOC

6 Conclusion

In this paper we presented how the SIOC ontology and lightweight semantics can be used and extended to represent the structure of wikis in an unified way. We first explained our motivations regarding some properties of wikis that we focused on in our modeling process, particularly focusing on a versioning process, and how we can benefit in this case of OWL reasoning capabilities. Then, we described how we designed a webservice to translate any MediaWiki page into SIOC data, following the Linked Data best principles to provide not only isolated RDF, but interlinked data. Finally, we gave some examples regarding how this data could be efficiently used for querying purposes.

While the work done here have been only applied to MediaWiki, further developments may include exporters and plug-in for other platforms to enable better cross-wikis integration. We also consider extending the versioning system defined in SIOC regarding wiki pages to other user-generated content. Moreover the semantic modeling of a wiki article might be improved adding more details about the content of the article itself. One of our goals is also to run cross-queries between our Wikipedia export and DBpedia, for instance to identify which people where the most active on a particular wiki page or topic. Finally, we also believe that this article gives a complete and nice overview regarding how to extend an ontology such as SIOC for particular purposes.

7 Acknowledgements

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

1. Danny Ayers and Max Völkel. Cool URIs for the Semantic Web. W3C Interest Group Note 03 December 2008, World Wide Web Consortium, 2008. `http://www.w3.org/TR/cooluris/`.
2. Tim Berners-Lee. Linked Data. Design issues for the world wide web, World Wide Web Consortium, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.
3. Chris Bizer, Richard Cyganiak, and Tom Heath. How to Publish Linked Data on the Web. Technical report, 2007. `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`.
4. John G. Breslin, Andreas Harth, Uldis Bojārs, and Stefan Decker. Towards Semantically-Interlinked Online Communities. In *Proceedings of the 2nd European Semantic Web Conference (ESWC2005)*, volume 3532 of *Lecture Notes in Computer Science*, pages 500–514. Springer, 2005.
5. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *The Semantic Web - ISWC 2002. First International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

6. Michel Buffa, Fabien L. Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A semantic wiki. *Journal of Web Semantics*, 6(1):84–97, 2008.
7. Andreas Harth, Hannes Gassert, Ina O'Murchu, John G. Breslin, and Stefan Decker. WikiOnt: An Ontology for Describing and Exchanging Wikipedia Articles. In *Proceedings of Wikimania 2005 – The First International Wikimedia Conference*, 2005.
8. Hak Lae Kim, Sung-Kwon Yang, John G. Breslin, and Hong-Gee Kim. Simple algorithms for representing tag frequencies in the scot exporter. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 536–539. IEEE Computer Society, 2007.
9. Markus Krötzsch, Denny Vrandecic, and Max Völkel. Semantic MediaWiki. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, volume 4273 of *Lecture Notes in Computer Science*, pages 935–942. Springer, 2006.
10. Christoph Lange, Uldis Bojārs, Tudor Groza, John G. Breslin, and Siegfried Handschuh. Expressing Argumentative Discussions in Social Media Sites. In *Proceedings of the ISWC2008 Workshop on Social Data on the Web (SDoW2008)*, volume 405 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
11. Alistair Miles and Sean Bechhofer. SKOS Simple Knowledge Organization System Reference. W3C Working Draft 29 August 2008, World Wide Web Consortium, 2008. `http://www.w3.org/TR/2008/WD-skos-reference-20080829/`.
12. Richard Newman, Danny Ayers, and Seth Russell. Tag ontology, December 2005.
13. Alexandre Passant and Philippe Laublet. Meaning Of A Tag: A collaborative approach to bridge the gap between tagging and Linked Data. In *Proceedings of the WWW2008 Workshop Linked Data on the Web (LDOW2008)*, volume 369 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
14. Alexandre Passant and Philippe Laublet. Towards an Interlinked Semantic Wiki Farm. In *Third Semantic Wiki Workshop – The Wiki Way of Semantics*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
15. Sebastian Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *First International Workshop on Semantic Technologies in Collaborative Applications (STICA 06)*, 2006.
16. Max Völkel and Eyal Oren. Towards a Wiki Interchange Format (WIF) - Opening Semantic Wiki Content and Metadata. In *Proceedings of the First Workshop on Semantic Wikis - From Wiki to Semantics (SemWiki-2006)*, volume 206 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

What The User Interacts With: Reflections On Conceptual Models For Semantic Wikis

François Bry, Michael Eckert, Jakub Kotowski and Klara Weiand

Institute for Informatics, University of Munich
Oettingenstr. 67, 80538 München, Germany
http://pms.ifi.lmu.de

Abstract. Traditional wikis excel in collaborative work on emerging content and structure. Semantic Wikis go further by allowing users to expose knowledge in ways suitable for machine processing, e.g. using Semantic Web technologies. The combination of ease of use, support for work in progress and Semantic Web technologies makes Semantic Wikis particularly interesting for knowledge-intensive work areas such as project management and software development. While several Semantic Wikis have been put to practical use, the concepts their users interact with have been little discussed. This position paper explores this issue, showing that the design of a conceptual model is not trivial and showing the repercussions of each design choice. The issue is explored stressing the social aspect of Semantic Wikis.

Key words: Semantic Wiki, Social Software, Semantic Web, Reasoning, Querying

1 Introduction

"Semantic Wiki" can refer either to Wikis enhanced with semantic technologies (or, after [8] "Semantic data for Wikis") or Wikis for ontology engineers (after [8] "Wiki for semantic data"). This paper uses the term in the first sense even though its contribution may be useful in both contexts.

Traditional Wikis are popular for managing personal and professional knowledge, primarily in the form of relatively simple hypertext, that is, Wiki pages and the links between them. Several features of traditional Wikis have been the key to their success: simplicity, openness and their thorough support for emerging and changing content in collaborative environments. Typically, Wikis are easy to use and allow anyone to make changes to the content. All Wiki content is version-controlled, meaning that previous versions of Wiki pages are never lost and changes can be tracked and reverted. Though many Wikis support advanced concepts that are relevant to their administration such as access rights and page and user groups, the basic concepts a regular users interacts with are limited to pages, links, and possibly text structuring and formatting.

Content in traditional Wikis consists of natural language text (and possibly multimedia files) and is not directly accessible to automated semantic

processing. Therefore, knowledge in Wikis can be located only through simple user-generated structures (tables of contents, inter-page links) and simple full text keyword search. More advanced functionalities that are highly desirable in knowledge-intensive professional contexts such as querying, reasoning and semantic browsing are not possible. Semantic Wikis introduce capabilities into Wikis for specifying knowledge not just in natural language but also in more formal, machine-processable ways.

Most of the advanced technologies that Semantic Wikis employ were developed for use in a static environment with annotations and rules being crafted by knowledge representation experts. This is in contraposition to the ever-changing, dynamic character of Wikis where content and annotations are, for the most part, created by regular users. In such an environment, inconsistencies and ambiguities can easily arise and the system should therefore be able to cope with them and support users in their work.

While several Semantic Wikis have been put to practical use [1,7,17,16,5,12,21] [14], each using their own conceptual model, there has been little explicit theoretical exploration on the possible choices for conceptual models and their consequences [20]. By conceptual model, we here understand the basic concepts or buildings blocks that a user interacts with as well as how these building blocks relate to each other[1]. In a traditional Wiki, there are typically only few such building blocks, the most basic ones being "page" and "link." Semantic Wikis, however, add new building blocks such as typed links, tags and RDF or OWL annotations. The basic building blocks of a Semantic Wiki and how they relate to each other has rarely been discussed in the literature, and one can assume that many decisions in this regard have been without full consideration of the design space.

In this article, we seek to draw attention to this issue, showing that the design of a concept model for a Semantic Wiki is a non-trivial issue and design choices greatly influence how the user sees the system and what functionalities the system can offer. We will show that there are several possibilities for approaching certain issues in a Semantic Wiki and that these have advantages and disadvantages, as well as important consequences on how other issues can be approached.

2 Content

This section outlines the representation of content in the Wiki. "Content" here refers to text and multimedia which is used for sharing information, most frequently through the use of natural language, between the users of the Wiki, and whose meaning is not directly accessible for automatic processing. Information Extraction techniques can be used to extract structured data from text or speech, which enables computerised processing, but this introduces another level of representation which is not considered "content" in this sense.

[1] Since the term "concept" is overloaded, we refer instead to "building blocks" of a conceptual model.

While regular Wikis are restricted to content as data, Semantic Wikis add further layers, namely data that can be used for human as well as automatic processing or data that is intended only for computers and not easily understandable for humans. These two other types of data are discussed in the following two sections.

Content Items. Content items constitute the primary unit of information in the Wiki; a simple textual content item can be thought of as being similar to a paragraph or section in a formatted text. Content items give structure to Wiki content. A content item can directly contain only one type of content, for example text or video. However, content items are also compositional to enable the representation of complex composite content structure. Therefore, content items can be nested, yielding complex content items. Content items do not overlap and every content item has a URI and can be addressed and accessed individually. As a consequence, there is no inherent distinction between Wiki pages and content items, or rather, by default, all content items are Wiki pages. If every (simple or complex) content item can only be embedded in one other content item, the Wiki content consists of a set of finite trees. Root nodes, that is, content items that do not have a parent node, then have a special status in that they encompass all content that forms a cohesive unit. In this, they can be seen as being alike to a Wiki page in a regular Wiki.

Having an explicit concept of content structure in a Wiki is desirable both with respect to the semantic as well as the social nature of a Semantic Wiki as the structural semantics of the content can be immediately used for querying and reasoning as well as for facilitating collaboration and planning of content. For example, queries could be used to automatically generate tables of contents and the modular nature of content items facilitates collaboration and planning. In addition, content items constitute a natural unit for assigning annotations for content (see Section 3).

Allowing one content item to have several parents, that is, to be directly contained in multiple other content items through transclusion [9], is a design decision that adds functionality but also has side-effects, some of which may be unwanted. Allowing transclusion means that content items can be easily reused and shared, which is useful for example for schedules or contact data. If a copy of the content item's content is embedded, multiple occurrences of the content item in the Wiki can not be traced as naturally or easily. On the other hand, updating the content item or reverting it to an earlier version can lead to unintuitive and undesired effects as the content item changes in all contexts it is embedded in. The user editing the content item then needs to be aware of all the contexts in which the content item is used and has to ensure that the change to the content item is appropriate in all contexts. To facilitate this, information about embedding locations should be readily available to the users, but even then, the user is burdened with deciding whether the change can be made, something which he might not be willing or feel knowledgeable enough to do. When the transclusion of content items is enabled, loops, which arise when a content item contains itself as a descendant, pose another problem. This is due to the fact the resulting

infinite recursion is problematic with respect to rendering the content item[2] as well as reasoning or querying over it. Since loops additionally appear to have no straightforward meaningful interpretation in the Wiki context, transclusions which would cause loops should generally be forbidden. In summary, allowing both content items that can be multiply embedded as well as content items that can only exist in one context combines the advantages of both strategies and gives the users maximum flexibility.

Fragments. Fragments are small, continuous portions of text (or, potentially, multimedia) that can be annotated with tags (see Section 3). While content items allow the authors to create and organise their documents in a modular and structured way, the idea behind fragments is that they constitute a means for users to annotate and use them separately from the original structure as they see fit and find useful. If content items are like chapters and sections in a book, then fragments can be seen as passages that readers mark; they are linear and in that transcend the structure of the document, spanning across paragraphs or sections and different sections of the book might be marked depending on which aspect or topics a reader is interested in.

Fragments should be maximally flexible in their placement, size and behaviour to allow for different groupings. Towards this goal, it is generally desirable that –unlike content items– fragments can overlap. The intersection between two overlapping fragments then can be further processed or it can be ignored. When two overlapping fragments f_1 and f_2 are tagged with "a" and "b" respectively, a third fragment that spans over the overlapped region and is tagged "a, b" could be derived automatically. Similarly, automatically taking the union of identically tagged overlapping or bordering fragments might be intuitive and expected by the user. However, this automatic treatment of fragments is a complex issue which might not always be appropriate or wanted.

On the other hand, fragments could be seen as co-existing but not interacting, meaning that the relationships between fragments are not automatically computed and no tags are added. This view has the advantage of being simpler and more flexible in that control of the fragments and their tags stays with the user. It is also in tune with the philosophy that, unlike content items that always only realise one structuring, fragments are individual in that different users can group a text in many different ways and under many different aspects. Fragments can either be restricted to the content directly contained in one content item, or it can span across content items. In the latter case, a rearrangement of content items can lead to fragments that span over multiple content items which no longer occur in successive order in the Wiki and, similarly, transclusion means that content items may contain only part of a fragment with the other part being absent (but present in some other content in which the content item is used).

Fragments could be deleted when the structure of content items no longer supports them, this means that a user might find a fragment she created destroyed as a consequence of another user's rearrangement of content items.

[2] At least if we assume that all of the content item is to be rendered at once.

Two possibilities of realising fragments are the insertion of markers in the text to label the beginning and end of an fragments ("intrusive"), or external referencing of certain parts of a content items, using for example XQuery, XPath, XPointer, or line numbers ("non-intrusive"). The latter has the advantage of allowing to define and annotate fragments on external content items, while the former means that fragments are less volatile and updates to the text do not affect fragments as easily, for example when text is added before the fragment.

External Content Items. Linked websites that are located outside of the Wiki are considered to be external content items. That means, they can be tagged and they can contain non-intrusive fragments, but they are not considered to be complex, that is, nested.

3 Semi-formal Annotations

One problem that frequently arises in the context of Semantic web applications is that it is hard to motivate users to annotate content since they find the process complicated and laborious. One solution is to provide means for creating less formal annotations which are easier to use. As work progresses, these annotations can be made increasingly more precise and can eventually be transformed into formal knowledge. Tagging is one such kind of semi-formal annotation. Tags normally consist only of keywords users associate with resources. Despite their simplicity, there are many possibilities as to how exactly the tags should work and be used [19]. Further, traditional keyword tagging can be extended in a number of ways [2,23,18] such as structured tags, negative tagging, and rules for tags [6]. Semantic links are another kind of semi-formal annotation. They are anchored in content items or fragments and can point to content items or fragments. Tags can be used on content items, fragments and possibly links as a way to assign a type to a link.

The semiotic triangle. One question to ask when designing a system that includes annotations is "What is annotated?" This question may have a quick, superficial answer: "Any resource that the system allows to be annotated." But what does that mean precisely? Let us say that the resource is a Wiki page about an elephant. Does a tag added to the page state a fact about the page itself (a representation of an elephant in the Wiki system) or does it refer to the actual elephant? This leads to a concept known as semiotic triangle [10], Peirce's triad [13] or de Saussure's distinction between the signifier and the signified [15]. This distinction is important because it has consequences on how the annotations are interpreted. In [11], the authors let the users decide what exactly they want to express by providing them with a syntax that allows the users to distinguish between these two cases.

Although it may not be important for the user, for the system design, it is essential to differentiate tags from tag associations (or "taggings"). Users connect tags to resources which is reflected in the system by the creation of a

tag association, which, apart from the user, the tag and the tagged resource, may involve additional information such as the time of the tagging event.

Structured tags. Ordinary flat tags are limited in their expressiveness. To overcome this limitation, different extensions of tagging are currently being proposed: machine tags[3], sub-tags [2] as used in the website http://www.rawsugar.com/, structured tags [2], etc. Most of the proposals are a variation of keyword:value pairs, some extend it to full RDF triples [23]. Note that keyword:value pairs can be seen as triples, too - the resource being annotated is the subject, the keyword is the predicate and the value is the object of the triple. More complex schemes which involve nesting of elements might be practical in some cases, e.g. "hotel(stars(3))" could express that the tagged resource is a three-star hotel. These extensions develop the structure of the tag itself and a set of tags is interpreted as a conjunction. It is conceivable to allow users to tag resources with a disjunction of tags or even with arbitrary formulae. This may be practical for some applications but it has two drawbacks: 1) reasoning with disjunctive information is difficult, 2) simplicity and intuitiveness would suffer.

Negative tags. So far, we have only addressed expressing positive information. In a collaborative context, we may be interested in tracking disagreements which presupposes some way to express negative information, such as negative tagging. If the user is allowed to tag a resource with tag "t" he or she may want to tag it with "not t" as a way to express disagreement or to simply state that the resource is not "t" or does not have the property "t". An example may be a medical doctor tagging a patient's card as "not lupus" to state that the patient definitely does not have "lupus". There are two ways to interpret negative tagging. It might be seen as classical negation or it may be seen as a kind of voting to express agreements and disagreements (see Section 5). Although a tag "not t" could be seen as introducing classical negation into the system, it may in fact be only a very weak form of negation because we can allow negating only pure tags, not general formulae (or sets of tags), and the only way to interpret this kind of negation would be by introducing a rule which says that from tag "t" and tag "not t" a contradiction symbol shall be derived (for more about reasoning see Section 6).

Tags as concepts. A hindrance in the transition from tags to more formal knowledge (e.g., RDF triples) is that tags are just keywords (i.e., strings). Often different keywords can be used to express the same abstract concept (e.g., keywords in different languages, synonyms). Similarly, the same keyword might be used to express different concepts (e.g., homonyms like "bank"). A possibility that fits well in the Wiki context, is to separate concrete keywords and the abstract concepts by using content items (which represent the abstract concepts) instead of keywords for tagging. Keywords still play an important role, as they are what is entered by the user, but the system will automatically resolve them to corresponding content items, possibly interactively asking for clarifications in the

[3] http://tech.groups.yahoo.com/group/yws-flickr/message/2736

case of ambiguities. In systems supporting semantic browsing over tags, there is also a natural need to have (partly automatically generated) content items for tags, e.g., to provide a list of all content items being tagged with a particular tag.

Using content items as tags also solves some further issues beyond synonyms and homonyms. Unlike keywords, content items have a URI that can be used when transforming information of semi-formal tags into formal RDF models (e.g., by the use of rules). Even more importantly, content items also offer a place for describing tags further. This encompasses both natural-language explanations for humans on the meaning and intended use of the tag as well as machine-readable descriptions, e.g., by means of tagging a tag's content item (see also tag hierarchies further down).

Links. Links are primarily used for navigation but can also be considered a kind of annotation. With respect to annotation, links can be seen as a way to specify some kind of relation between the two linked resources. For an untyped link, this relation may default to the "is related to" relation. Typed links express a specific relation between two resources. Link type is a new concept in the usage model which could be unified with the rest of the system by letting the user specify the type by tagging the link. Advantages of this approach are the intuitiveness of tagging and the social, work-in-progress aspect of the environment which allows the users to converge on a precise meaning of the link only as their work progresses (e.g. by discussing the link type on the page of the tag). Disadvantages are unclear meaning of a link with multiple tags and possible user interface issues. A question that arises with links with multiple tags is how they are interpreted for reasoning, querying, and translation to formal knowledge (e.g. RDF). For this consider a link between resources R_1 and R_2 with tags A and B. Can it be distinguished from two links between resources R_1 and R_2, one with tag A, the other one with tag B? Treating multiple tags as multiple links, i.e. not distinguishing the two situations, is simpler for translation to RDF because then each link maps directly to one triple where tags correspond to properties. If on the other hand they are to be distinguished, a new property has to be introduced to express that the link is tagged with both A and B.

Tags vs links. When tags as concepts are supported, simple flat tags on content items can be seen as a kind of link between the tagged resource and the concept of a tag represented by the content item describing its meaning (see above). Similarly, structured tags, such as keyword-value pairs, can be seen as expressing a relation (or a link), with its type given by the keyword, between the tagged resource and another resource, given by the value. In a Wiki supporting semantic browsing over such tags, the question may then arise of what differentiates a link from tags and structured tags. From a technical point of view there may not be a strict differentiation after all, flat tags can be seen as specialised links between a taggable resource and the content item describing the concept of the tag, as a link is then a general way of expressing a relation. The difference usually is in the way they are presented and used. Tags are usually represented separately

from a content item, e.g. in a special area of the page, while links are represented with anchors inside the content item. Further, tags make a statement about a single content item, e.g. give it a type, whereas purpose of links is to express an association between two content items. Finally, while links can be tagged, one cannot link to or from another link.

Tag Hierarchies. Tag hierarchies constitute a step in the transition from informal to formal annotation. They are useful for example for reasoning and querying since they enable the processing of tag relationships. Tag hierarchies could be created through "tagging tags", that is, tagging a tag's content item to indicate an "is-a" relation.

Semi-formal annotations described in this section provide a means to transform knowledge from human-only content described in Section 2 to machine-processable information described in the next section. Semi-formal annotations seem to be an important feature of social software because they provide a low-barrier entry point for user participation on enrichment of content with metadata which are machine processable. Users can use gradually more expressive and formal methods of annotation as they become familiar with the system. First, they only create and edit content. Then they can begin using flat tags to annotate content and later perhaps start using structured tags. Advanced users or system administrators can further enhance the metadata enrichment efforts by specifying rules for semi-formal annotations, see Section 6.

4 Formal Knowledge Representation

Currently, the most common format for semantic data is RDF. RDF data are easily processable by machines but not easily interpretable by Wiki users. They can use semi-formal annotations which can then be represented directly in RDF or be later transformed to formal annotations that use vocabularies with well-defined meanings. This transformation can be supported by rules or by methods that automatically extract folksonomies from sets of tags. In this approach, ontologies could naturally emerge based on on-demand basis as a formalisation of semi-formal annotations of Wiki content. Semi-formal annotations such as flat tags, structured tags and links can be easily translated into RDF triples, meaning that RDF is a suitable choice for the representation of all annotations. It may be the case for the low-level implementation of the system but it is not desirable to let the user write or read raw RDF data. It is usually obvious how to use a tag but it is not obvious how to write an RDF triple. Therefore the user should be rather exposed to higher-level, intuitive concepts such as tag, structured tag and link which can then be automatically translated into RDF. For practical applications, support of RDF is important also because of interoperability with current semantic web applications and linked data [4,3]. A social Semantic Wiki should therefore support at least import and export of RDF data.

5 Social Content Management

To facilitate social collaboration and leverage the social aspects of the Semantic Wiki, several options and aspects have to be considered.

Groups. User groups can be used among other things for personalisation of wiki content, for querying and reasoning and to attribute wiki data to a group. Tags are an easy way to group things which is used in the wiki, so it would be an obvious choice to form user groups by tagging users' content items. Every tag that is used on at least one (or possibly two) user's content item then constitutes a group. One possible drawback of this approach is that this proliferation of groups might be demanding in processing when special mechanisms for treating groups are established.

The social weight of tags. If several users tag one item with the same tag, it is natural to aggregate these tag assignments to give a clearer view of all the tags assigned. Tags then can be seen to have weights depending on how often they were assigned. On the other hand, other users might not agree with the assignment of a certain tag to a content item, adding a negative component to the tags' weight. The overall social weight of a tag can then be calculated for example by weighting the number of positive assignments versus the sum of both positive and disagreeing assignment, or by assigning a value to both actions and calculating the total. The social weight of a tag then summarizes the users' views on the appropriateness of a specific tag assignment and thus provides a valuable measure that can be used in reasoning and querying. Moreover, reinforcing or disagreeing about tag assignments constitutes a low-barrier activity in the wiki, making it easy for beginning users to participate.

Agreement or disagreement could also be expressed with respect to a content item itself, for example through a specific set of tags which are reserved for this use.

Access Rights. Users, user groups and rules for reasoning could be used to handle access rights in the wiki, but this is a complex issue which requires further investigation. Questions that arise include who owns the rules and what are the access rights on rules and who can assign the tags that restrict the access. Static rules would not be suited for rights managements in all environments. For them to function well, the organization and roles in the wiki have to be relatively stable, which may be the case in professional applications. In other areas, such as the development of open source software, such rules may not be desired or the social organization might not be static enough for rules to be adequate.

6 Reasoning

Reasoning is enabled by the formal and semi-formal annotations in Semantic Wikis. Wiki content can change frequently, disagreements are common and inconsistencies and ambiguities can easily arise. This is not only unavoidable but

even desirable in a creative environment and reasoning should be able to cope with it and support users in their work-in-progress. Reasoning that has these properties was sketched previously in [6]. As indicated in previous sections, it is also desirable that social software supports a rule language for annotations that would help users to further annotate content. A rule might for example express that a tag "elephant" induces an implicit "mammal" tag. Another example might be a rule expressing that a tag "bug report" without a corresponding tag "processed" induces an implicit "todo" tag.

The second example of a rule presupposes that the rule language includes negation as failure. This choice of negation seems to be appropriate for a Wiki. A Wiki is, in a way, a world of its own. For example, if there is a page describing a meeting with a list of tasks to be done then it is safe to assume that the list is complete, i.e. there are no other tasks for this meeting other than the ones listed on the page. Also, if a meeting is not tagged as "all-hands" then it is safe to assume that it is not an all-hands meeting. Therefore negation as failure seems to be the negation of choice for a Semantic Wiki. On the other hand, negative tagging, as discussed in Section 3, expresses negative information explicitly. Therefore the user could express not only positive and negative information but also refer to missing positive and missing negative information. The problem of combining classical negation with negation as failure is a field of its own and it cannot be expected that a regular user would understand it. On the other hand, recall that, in our approach, negative tags can be just positive tags with a negative marker that is interpreted only using a rule "from tag t and tag not t derive a contradiction *symbol*". This results only in a very weak form of negation that should not be too difficult to combine with negation as failure in an intuitive way. The derivation of a contradiction symbol rather than a contradiction enables paraconsistent reasoning, see [6] for details. The reasoning approach sketched in [6] has also a social aspect in that it allows to track different inconsistencies to their origin and thus can provide users with useful information about the cause of each inconsistency which may be a result of a disagreement within a group of people or simply a mistake.

It may be interesting and beneficial for users to see how a specific group of users tagged a resource or to compute the ratio of the number of people agreeing and disagreeing with an item. Therefore the rule language should include aggregation and be sufficiently expressive to allow referring to tags by different users associated to different content items.

7 Querying

A query language for a Semantic Wiki should enable users to select, access and reuse data while leveraging the Wiki's properties to improve retrieval, ranking and the "mashing-up"/embedding of existing data. Traditional Wikis frequently use full text search, while several query languages have been proposed for semantic web data, specifically XML and RDF [22]. Full text search is simple but not powerful. Semantic web query languages on the other hand, are too compli-

cated for casual users, although some efforts to enable keyword-based querying of RDF and XML. Furthermore, relatively little research has been made in the area of combining querying of textual data and annotations.

However, in the Semantic Wiki, all conceptual building blocks, for example content items and their structure, tags and tag hierarchies should be amenable to querying and it should be possible to combine selection criteria for several data sources in one query, for example expressed as label:keyword terms where the label specifies a datasource (text, tags) or property (bold, author, time added) and the keyword is a string.

Further, the query language needs to be versatile and user-friendly, meaning that users should be able to enter just a keyword and get meaningful results, while more experienced users should be able to specify complex queries. The transition between the two needs to be smooth and flexible. A suitable query language for a Semantic Wiki should allow for aggregation and construction of results to create views in the form of content items composed of Wiki data. Similarly, queries embedded in content items may be used to always display up-to-date query results that change as the Wiki content does.

Finally, the ranking of results could utilise properties specific to the Semantic Wiki like tag weights, edit frequency, the number of hits or the author's expertise or equity value to improve ranking results.

8 Conclusion

In this article, we explored the conceptual bulding blocks of Semantic Wikis and outlined choices that have to be made when designing a Wiki with advanced functionalities. There are many options and details we could not discuss here for space reasons. One vital question we had to omit is how the two kinds of data – content and annotations– are to be handled with respect to versioning, such as whether content and annotations should be versioned together or separately. We also did not discuss a complex method of measuring social weight of tags and content items called community equity. Community equity is employed by Sun Microsystems, a partner in the KiWi project, in their internal portal SunSpace and is used to encourage user participation by showing them the importance of their contributions to the community (i.e. community equity). This paper focused on two of the advanced wiki functionalities, namely reasoning and querying. There are other advanced functionalities such as personalisation and information extraction which affect the design decisions as well. For example fragments are an important concept for annotation by means of information extraction and rules and groups play an important role in personalisation. Also, many Semantic Wikis have already been implemented and it would be worthwhile survey the design decisions that were made in these existing systems; to the best of the authors' knowledge no such survey exists yet[4].

[4] Although recently, there has been a related effort to create a "Semantic Wiki Feature Matrix", see http://semanticweb.org/wiki/Semantic_Wiki_State_Of_The_Art

Acknowledgements The research leading to these results is part of the project "KiWi - Knowledge in a Wiki" and has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211932.

References

1. S. Auer, S. Dietzold, and T. Riechert. Ontowiki-a tool for social, semantic collaboration. *International Semantic Web Conference*, 4273:736–749, 2006.
2. Judit Bar-Ilan, Snunith Shoham, Asher Idan, Yitzchak Miller, and Aviv Shachak. Structured vs. unstructured tagging a case study. 2006.
3. T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, volume 2006, 2006.
4. Tim Berners-Lee. Linked data. *W3C Design Issues*, 2006.
5. Philip Richard Boulain. Swiki: A semantic wiki wiki web. Master's thesis, University of Southampton, 2005.
6. François Bry and Jakub Kotowski. Towards reasoning and explanations for social tagging. *Proc. of ExaCt2008 - ECAI2008 Workshop on Explanation-aware Computing. Patras, Greece*, http://www.pms.ifi.lmu.de/publikationen#PMS-FB-2008-2, 2008.
7. Kensaku Kawamoto, Yasuhiko Kitamura, and Yuri Tijerino. Kawawiki: A template-based semantic wiki where end and expert users collaborate. In *Proceedings of 5th International Semantic Web Conference (ISWC2006)*, 2006.
8. Markus Krtzsch, Sebastian Schaffert, and Denny Vrandecic. Reasoning in semantic wikis. In *Reasoning Web Summer School 2007*, pages 310–329, 2007.
9. T.H. Nelson. *Literary Machines*. Mindful Press, 1993.
10. C.K. Ogden, I.A. Richards, B. Malinowski, and F.G. Crookshank. *The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Harcourt, Brace & Company, 1938.
11. E. Oren. Semantic Wikis for Knowledge Workers. 2005.
12. Eyal Oren. Semperwiki: a semantic personal wiki. In *Proceedings of 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure*, Galway, Ireland, 2005.
13. C.S. Peirce. On a new list of categories. In *Proceedings of the American Academy of Arts and Sciences*, volume 7, pages 287–298, 1868.
14. Niko Popitsch, Bernhard Schandl, Arash Amiri, Stefan Leitich, and Wolfgang Jochum. Ylvi - multimedia-izing the semantic wiki. In *Proceedings of the 1st Workshop "SemWiki2006 - From Wiki to Semantics"*, Budva, Montenegro, 2006.
15. F. Saussure. Course in general linguistics (W. Baskin, Trans.). *New York: Philosophical Library*, 1916.
16. Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic wikis. *IEEE*, page 7, 2008.
17. Sebastian Schaffert, Rupert Westenthaler, and Andreas Gruber. Ikewiki: A user-friendly semantic wiki. In *3rd European Semantic Web Conference (ESWC06)*, Budva, Montenegro, 2006.
18. B. Sereno, B. Shum, and E. Motta. Formalization, User Strategy and Interaction Design: Users' Behaviour with Discourse Tagging Semantics. 2007.

19. Gene Smith. *Tagging: People-powered Metadata for the Social Web (Voices That Matter)*. New Riders Press, December 2007.
20. R. Tolksdorf and E.P.B. Simperl. Towards wikis as semantic hypermedia. In *Proceedings of the 2006 international symposium on Wikis*. ACM New York, NY, USA, 2006.
21. Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic wikipedia. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 585–594, New York, NY, USA, 2006. ACM.
22. K. Weiand, T. Furche, and F. Bry. Quo Vadis, Web Queries? In *Proc. Int'l. Workshop on Semantic Web Technology (Web4Web) 2008*, 2008.
23. Jie Yang, Yutaka Matsuo, and Mitsuru Ishizuka. Triple tagging: Toward bridging folksonomy and semantic web. *ISWC07*, page 14, 2007.

Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis

Rolf Sint[1], Sebastian Schaffert[1], Stephanie Stroka[1] and Roland Ferstl[2]

[1] {firstname.surname}@salzburgresearch.at
Salzburg Research
Jakob Haringer Str. 5/3
5020 Salzburg
Austria
[2] roland.ferstl@siemens.com
Siemens AG (Siemens IT Solutions and Services)
Werner von Siemens-Platz 1
5020 Salzburg
Austria

Abstract. The growing impact of Semantic Wikis deduces the importance of finding a strategy to store textual articles, semantic metadata and management data. Due to their different characteristics, each data type requires a specialized storing system, as inappropriate storing reduces performance, robustness, flexibility and scalability. Hence, it is important to identify a sophisticated strategy for storing and synchronizing different types of data structures in a way they provide the best mix of the previously mentioned properties.
In this paper we compare fully structured, semi-structured and unstructured data and present their typical appliance. Moreover, we discuss how all data structures can be combined and stored for one application and consider three synchronization design alternatives to keep the distributed data storages consistent. Furthermore, we present the semantic wiki *KiWi*, which uses an RDF triplestore in combination with a relational database as basis for the persistence of data, and discuss its concrete implementation and design decisions.

1 Introduction

Although the promise of effective knowledge management has had the industry abuzz for well over a decade, the reality of available systems fails to meet the expectations. The EU-funded project *KiWi* - Knowledge in a Wiki project sets out to combine the wiki method of collaborative content creation with the technologies of the Semantic Web to bring knowledge management to the next level. Combining a Wiki with Semantic Web technologies results in three types of content:

Wiki Articles, which are basically unstructured textual content,
Management Data, like authors, creation dates and revisions, and

Semantic Metadata, which provide flexibility and spreading of the data about *KiWi*'s contents.

In this paper we explain the differences of data storage for these data types. We describe our choice of design and illustrate its usefulness for Semantic Social Software Applications. Furthermore, we explain how these three different approaches can be integrated in a single application, which is build with the Java Enterprise Edition (Java EE)[1] platform.

We present and discuss three different kinds of data: structured, unstructured and semi-structured. We discuss the weaknesses and strengths of each of them and describe that for a semantic social software application a combination of them brings advantages in form of an improved flexibility and performance. Furthermore, we describe several ways and designs how an application can implement the different ways of persisting data. The main challenge by developing an application which uses different data storages is to define a common interface for the access of data and to guarantee the synchronization of the different data storages.

Chapter 2 discusses the benefits, techniques and differences of structured, unstructured and semi-structured data. For this discussion examples for each paradigm are compared: An Apache Lucene full-text index for unstructured data, a relational database for fully structured data and an RDF triplestore for semi-structured data.

Chapter 3 describes several design patterns, which were used within *KiWi* to combine the three different approaches, focusing on the combination of relational databases and RDF triplestores. This chapter tries to answer the question which data should be stored where and discusses the decisions taken by the *KiWi* project.

Chapter 4 gives an overview of related work and chapter 5 summarizes the practical relevance of this approach.

2 Structured, Unstructured and Semi-Structured Data

In the semantic wiki *KiWi* we need all three kinds of data: structured, unstructured and semi-structured. This chapter presents and compares the different forms of data and gives examples and state-of-the-art techniques. Finally, a tabular overview of the different kinds of data structures is given.

2.1 Unstructured Data

According to[1], the term *unstructured* refers to the fact that no identifiable structure within this kind of data is available. Unstrucured data is also described as data, that cannot be stored in rows and columns in a relational database.

Storing data in an unstructured form without any defined data schema is a common way of filing information. An example for unstructured data is a document that is archived in a file folder. Other examples are videos and images.

[1] `http://java.sun.com/javaee/`

The advantage of unstructured data is, that no additional effort on its classification is necessary. A limitation of this kind of data is, that no controlled navigation within unstructured content is possible.

A common technology to search in unstructured text documents is full-text search. The advantage of full-text search is, that it is completely decoupled from the data. This makes it very flexible, because it can be used on every kind of textual data, even if no schema or structure is defined. One limitation of full-text search is that it cannot be used to search for pictures or videos.

Full-Text Search can be optimized by generating a full-text index, that increases the performance of a full-text search query. A famous full-text search engine library is *Apache Lucene*[2] . Other examples are MySql[3] and Postgres indixes[4].

2.2 Fully Structured Data

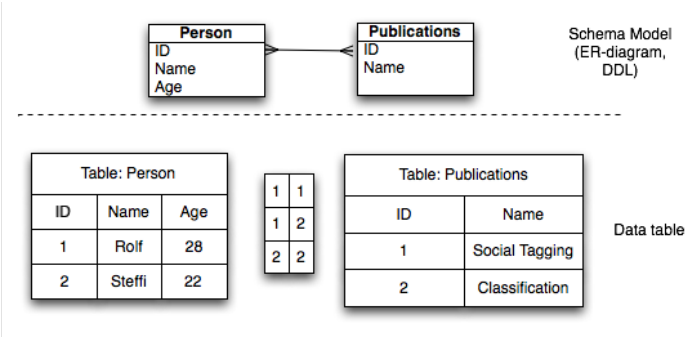


Fig. 1: Sample Table in a Relational Database System

Fully structured data follows a predefined schema. *"An instance of such a schema is some data that conforms to this specification,"*[2]. A typical example for fully structured data is a *relational database system*. Designing a database schema is an elaborate process, because a schema has to be defined before the content is created and the database is populated. The schema defines the type and structure of data and its relations. Figure 1 illustrates an Entity Relationship diagram (ER-diagram) and its concrete tables within a RDBMS (relational database management system).

"The well-defined schema of fully structured data enables efficient data processing and an improved storage and navigation of content,"[2, page 122]. The

[2] `http://lucene.apache.org`
[3] `http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html`
[4] `http://www.postgresql.org/docs/7.4/static/indexes.html`

cost for high performance and navigation is flexibility and scalability. It is difficult to subsequently extend a previously defined database schema that already contains content. For example, it is not possible to extend a single table row with a new attribute without creating another table column. This is unprofitable for tables that contain thousands of other rows that do not need another attribute.

An advantage of relational database applications are the existing tools and web frameworks, which support the development of database-focused applications. For instance, *Hibernate*[5] and *Oracle TopLink*[6] are *Object/Relational* (O/R) *Mapping* frameworks, which map classes and objects to relational database tables and rows. Moreover, there exist several practical tools for maintenance, management and administration of relational database systems.

2.3 Semi-Structured Data

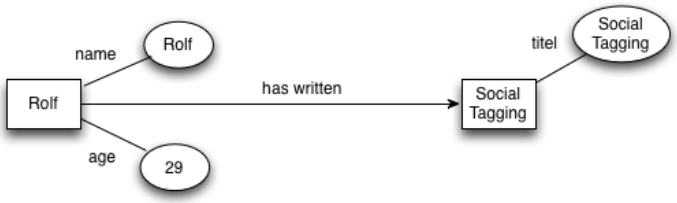


Fig. 2: Sample RDF Graph

Semi-structured data is often explained as *"...schemaless or self-describing, terms that indicate that there is no separate description of the type or structure of the data"*[2, page 11]. Semi-structured data does not require a schema definition. This does not mean that the definition of a schema is not possible, it is rather optional. The instances do also exist in the case that the schema changes. Furthermore, a schema can also be defined according to already existing instances (posteriori). The types of semi-structured data instances may be defined for a part of the data and it is also possible that a data instance has more than one type[2].

One of the strengths of semi-structured data is *"... the ability to accommodate variations in structure"*[2, page 12]. This means that data may be created according to a specification or *close* to a type. For instance, fields can be duplicated, data can be lacking or there may exist minor changes[2]. Figure 2 illustrates a graph representation of semistructured data. Figure 4 illustrates the same schema as in Figure 3, with the difference that the instance model has an additional property, which is not defined in the schema model.

[5] `http://www.hibernate.org/`
[6] `http://www.oracle.com/technology/products/ias/toplink/index.html`

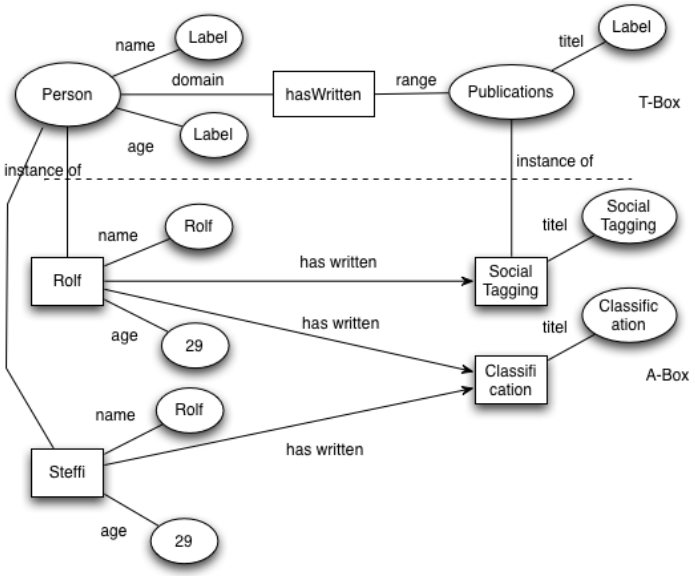


Fig. 3: RDF Schema (RDFS) and two instances

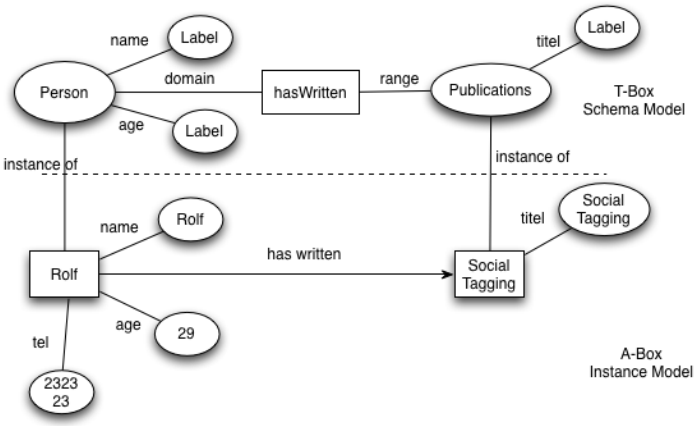


Fig. 4: RDFS and a flexible instance

A typical example of semi-structured data is XML, which is a language for data representation and exchange on the web. In XML data can be directly encoded and a *Document Type Definition* (DTD) or *XML Schema* (XMLS) may define the structure of the XML document[2].

In the research fields of the Semantic Web, knowledge is encoded in Resource Description Framework (RDF) triples[3], which store data in the form of subject, predicate and object nodes. The RDF Schema (RDFS)[4] vocabulary definition language allows the definition of classes and properties. In the World Wide Web RDF is used as a language that provides metadata to web resources.

2.4 Transformation of Data

In *KiWi*, data sometimes needs to be transformed from one structure into another. For instance, fully structured data is converted into unstructured data when a user generates a PDF out of a wiki article and its management data like author, creation date and so forth. It is also possible to convert data from a database into semi-structured data, like an RDF graph. Several modern web applications use RSS feeds , which are generated by reading data of a relational database and provide it in RDF format.

On the contrary, it is more complex to transform unstructured information into semi- or fully structured information. *KiWi* structures textual content with techniques of information extraction and natural language processing. Tags, which describe the content of a text, are automatically extracted out of a wiki article. In this way the unstructured data can be converted into semi-structured data.

2.5 Comparison and relevance for an application

It can be summarized, that the high degree of typing enables a better performance and less flexibility.

Serge Abiteboul, Peter Buneman and Dan Suciu define several reasons why defining a structure is good for[2]:

- to optimize query evaluation,
- to improve storage,
- to construct indexes,
- to describe the database content to the user and facilitate query formulation,
- to proscribe certain updates, and
- to support strongly typed languages.

Table 1 gives an overview over the strengths and weaknesses of the different storing structures in technology fields that may be important in practice.

2.6 Conceptual Federation of Relational Databases and Triplestores

To know how to combine a relational database and a triplestore we have to consider what data is stored where. Therefore, we review the strengths and

	Unstructured	Fully Structured	Semi-Structured
Technology	Character and binary data	Relational database tables	XML/RDF
Transaction Management	No transaction management, no concurrency	Matured transaction management, various concurrency techniques	Transaction management adapted from RDBMS, not matured
Version Management	Versioned as a whole	Versioning over tuples, rows, tables, etc.	Not very common, versioning over triples or graphs is possible
Flexibility	Very flexible, absence of schema	Schema-dependent, rigorous schema	Flexible, tolerant schema
Scalability	Very scalable	Scaling DB schema is difficult	Schema scaling is simple
Robustness	-	Very robust, enhancements since 30 years	New technology, not widely spread
Query-Performance	Only textual queries possible	Structured Query allows complex joins	Queries over anonymous nodes are possible

Table 1: Comparison of unstructured, fully structured and semi-structured content

weaknesses of different data structures and discuss the demand of structure characteristics for specific data sets. A relational database stores fully structured data, which necessarily have a predefined schema. Relational databases provide the application with a high query-performance and fast joins. Vulnerabilities are rare since more than 30 years of research, development and improvement eliminated most of them and increased the robustness.

Semi-structured data like RDF data does not have to predefine a schema and is very scalable and flexible. Furthermore, RDF and OWL[7] allow the definition of logical rules and many applications implement an inference layer that infers new triples by reasoning over the existing data set.

Thus, data that has a predefined schema, that is sensitive and that is often queried should be stored in a relational database. Data that is added to the application lately (e.g. data for extensions or plug-ins) and data that might be important for reasoning should be stored in the triplestore. Figure 5 provides a quick overview over the division into relational database data and triplestore data. As one can see, the data sets are partially overlapping.

[7] `http://www.w3.org/TR/owl-features/`

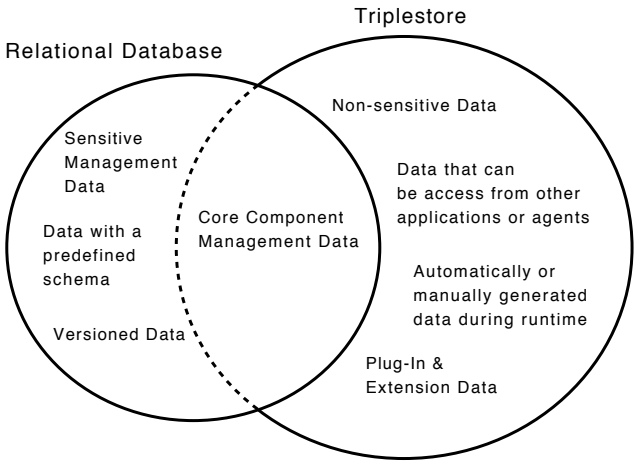


Fig. 5: Overlapping data sets stored in the triplestore and in the relational database

3 Data representation in *KiWi*

Combining structured and unstructured data is an often applied strategy in web applications to achieve the advantages of both persistence types. The employment of all three alternatives, however, is uncommon.

KiWi is a platform for Semantic Social Software applications, implemented with Java EE technologies. We decided to store data in a semi-structured form, because we wanted to attain a better flexibility and scalability than provided by the structured form. We also wanted to store data in a robust database with good query and join performance. We have to control a big amount of textual content, which needs to be queried for keywords.

Hence, we decided to combine unstructured, structured and semi-structured data storage and segmented the data into long textual content (*unstructured*), core component data (*fully structured*) and flexible data (*semi-structured*). For a better clarity, Table 2 visualizes the segmentation. The sets of fully structured and semi-structured data are overlapping, because we represent the non-sensitive core data additionally in the triplestore to get a complete data set that can be provided to other Semantic Web Applications (e.g. Linked Data[8]).

3.1 Three possible Levels of Synchronization

Applications that store data in a triplestore as well as in a relational database have to implement a synchronization mechanism to keep information consistent. Such a synchronization mechanism can be implemented on different layers of an application.

[8] http://linkeddata.org

	Content Type	Example
Unstructured	Textual Content	Wiki Articles, Blog Pages
Fully Structured	Sensitive Content & System Maintenance Data	ContentItem,
	Core Component Data	User data
Semi-Structured	Non-sensitive Core Component Data, Flexible Content & Individual Data	ContentItem-extending Data, Use Case Data

Table 2: Persistence alternatives and apportioned content

Database Layer Synchronization on the database layer is implemented by forcing a data storage (e.g. database) to update another data storage (e.g. triplestore) when a data item changed. For instance, every time an application writes on a database, the according operation could be executed on the triplestore, which might be hold in the database. This could be implemented using database triggers or Java EE persistence interceptors. Another possibility is that the triplestore is generated automatically from the entries within the database. Hence, the triplestore could be updated regularly. In both variants the database is defined as master and the triplestore is defined as slave. This design is illustrated in Figure 6.

This design benefits from high performance and good integration of relational databases into existing software technology stacks (e.g. Java EE). Furthermore, functions provided by a triplestore, like reasoning, are possible, because the data also exists in a semi-structured form. The disadvantage is that this design does not offer the flexibility of semi-structured data, and that the application has read only access to one data storage.

An alternative design is a bi-directional trigger synchronisation between relational database and triplestore. The triplestore, as well as the relational database can update each other with database triggers. The advantage of this design is that it allows writing access to both data storages. This design is illustrated in Figure 7. The limitation is, that some updates on the triplestore cannot be processed on the database and must be forbidden to keep consistency. Therefore, this design does not support the full flexibility of semi-structured data, too.

O/R Mapping Tool O/R mapping tools provide another layer for synchronisation. This design is illustrated in Figure 8. For instance, the Java Persistence API (JPA)[9] could be extended to persist Java objects in the database as well as in the triplestore. This encloses the translation of JpaQL (JavaPersistenceApiQueryLanguage)[10] queries into triplestore queries. This approach decouples the

[9] http://java.sun.com/developer/technicalArticles/J2EE/jpa/
[10] http://java.sun.com/javaee/5/docs/tutorial/doc/bnbtg.html

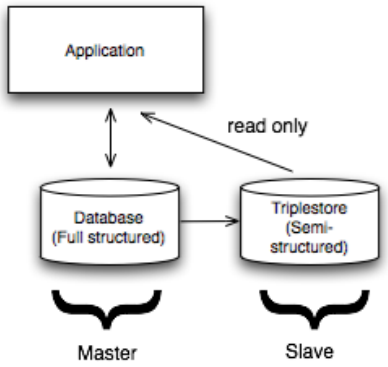


Fig. 6: Database defined as master and triplestore defined as slave

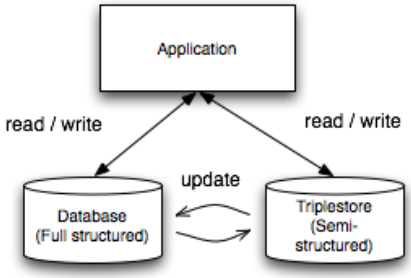


Fig. 7: Triplestore and database update each other

persistence layer from the application layer, and, therefore, provides the flexibility of semi-structured data. Thus, additional attributes of an object may be defined during the runtime of an application and persisted in a triplestore. This may be realized using Aspect Oriented Programming (AOP)[11] techniques or dynamic languages like Groovy[12]. With this approach, distributed queries over several datasources could be realized.

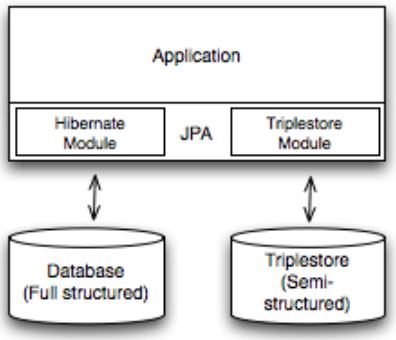


Fig. 8: Extension of the JPA with a triplestore module to guarantee consistency

[11] http://www.eclipse.org/aspectj/
[12] http://groovy.codehaus.org/

Application Layer / Middleware Layer Another alternative to guarantee the synchronisation of data is to implement it in the middleware or application layer. This layer could use normal JpaQL queries for the database as well as SPARQL commands to query the triplestore. This design is illustrated in Figure 9. A different alternative is to provide a general purpose query language for both data stores. In this way, distributed reasoning over the triplestore, as well as over the data in the relational database system could be enabled. This design is illustrated in Figure 10.

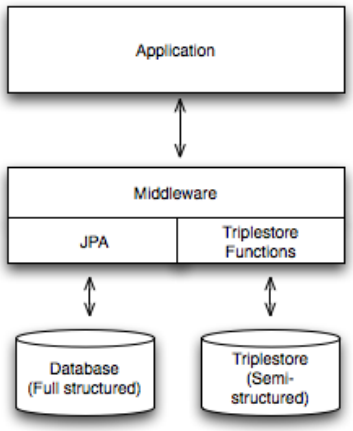


Fig. 9: Middleware layer which handles the persistence of data

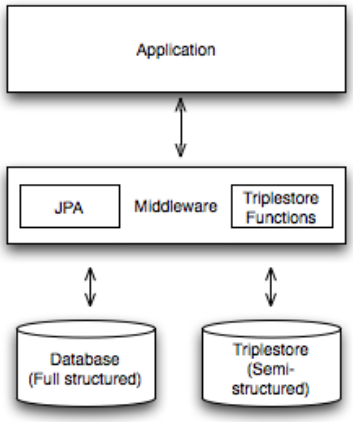


Fig. 10: General purpose query language

3.2 Integration of a triplestore in the Java EE stack

We decided to choose the Application Layer for synchronization, because it grants us flexibility to improve weaknesses and to enforce the strengths of each data structure type. First, we will give you an overview over the triplestore position inside of *KiWi*.

Figure 11 illustrates the overall structure of *KiWi*. The combination of triplestore and relational database can be found in the *Persistence* and *Data Model* layers. As an RDF triplestore *KiWi* currently uses *Sesame2*[13]. The relational database connection is enabled through Hibernate with JPA. Storage configurations for relational database and triplestore can be applied with Java annotations.

[13] http://www.openrdf.org/

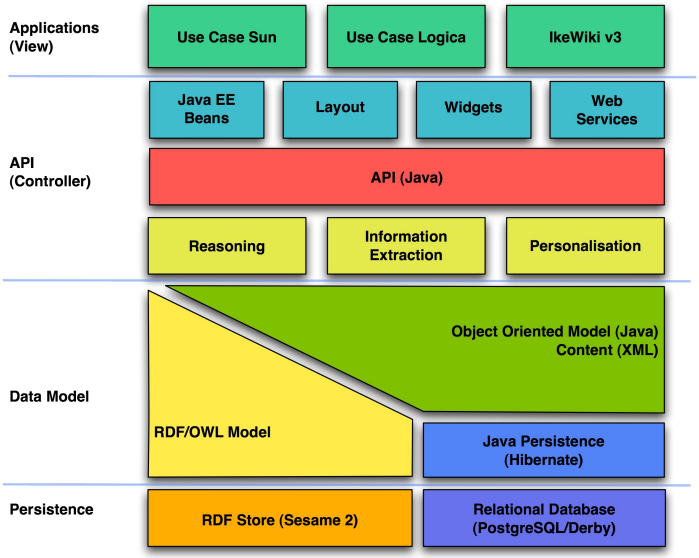


Fig. 11: KiWi's overall structure, adapted from[5]

Transactional synchronization As Table 1 illustrated, transaction management for unstructured and semi-structured data is not very common or matured. Though, storing data in those federated, heterogeneous databases needs to be controlled to avoid states of inconsistency. A global transaction management is the easiest way to administer all three data structure types in terms of their transactions. *JBoss Seam*[6], *Hibernate/JPA*[7], and *Enterprise Java Beans* (EJB)[8] provide us with diverse techniques to control transactions programmatically and declaratively, for example:

Java Transaction API, also called JTA[14] specifies standard Java interfaces for Java Enterprise Applications implemented by the application server[9].

Seam Transactions extend JTA UserTransactions with useful functionality, for example the registration of a synchronization implementation[6].

EntityManager Transactions are provided by Hibernate/JPA for programmatic transaction management to start and stop transactions explicitly[10].

Programmatic transaction processing requires the definition of a start and end time for the transaction. It allows flexible pre- and post-treatment of the application when the transaction ends. Declarative transaction processing, on the other hand, is simpler than programmatic transaction management, because the transaction start and end time is managed by the container[11]. To control the behaviour before and after a transaction ends in applications using declarative

[14] http://java.sun.com/javaee/technologies/jta/index.jsp

transaction processing, a synchronization implementation can be registered[9]. In *KiWi* we use the *before-completion phase* to synchronize the relational database state with the triplestore state. Thus, updates to both databases will be executed simultaneously at the end of a transaction. Figure 12 illustrates the process. If an update fails, the whole transaction including changes on both databases will be rolled back.

A more detailed description of the transaction models in Java Enterprise Applications, the concurrency problems that triplestores must consider and the database synchronization is given in [12].

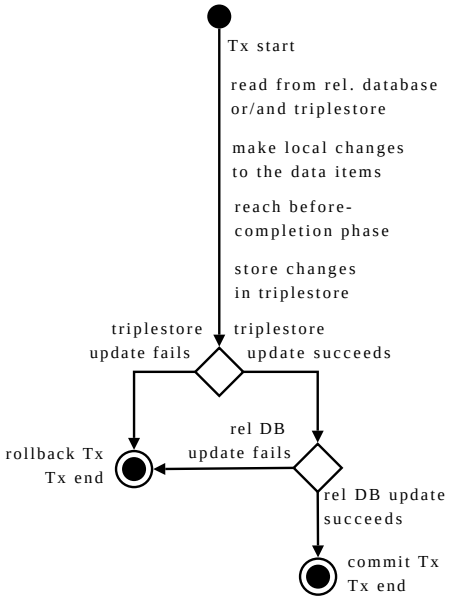


Fig. 12: Transactional synchronization process

Data Versioning Versioning of unstructured, semi-structured and fully structured data is an important core functionality of *KiWi*. RDF triple versioning is uncommon and few well-established RDF repositories allow versioning. Sesame2 puts RDF triples internally under version control, but it does not enable undo or redo functions.

With the chosen transaction strategy we can easily implement version-control of unstructured, semi-structured and fully structured data. At the end of a transaction, updates for all kinds of data are creates and stored as revisioning and update tables in the relational database. This design was chosen to collect all versioning data in a robust database, to enable easy querying, and, consequently,

to allow fast undo and redo functionality for all kinds of data. Versioning data has a pre-defined schema that will not be changed in the future.

Query & Reasoning With the chosen level of synchronization it is possible to create a query language for all kinds of data. *KiWi* enables this global querying that interprets to *SQL* and *SPARQL*[15] queries. Furthermore, reasoning is not limited to the RDF repository anymore. The interested reader is referred to [13] for a more detailed discussion about this issue.

4 Related Work

In the following we provide an overview over implementations of semi-structured data into existing application stacks. *Elmo*[14] is a Java library for Semantic Web applications that maps Java classes to RDFS/OWL classes. Another implementation of a server which offers access to different representations of data is *Virtuoso*, *"... which is a database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, Web Application Server and File Server functionality in a single server product"*[15].

5 Conclusion

The main advantage of fully structured data is the strong typing which enables high performance and efficiency. On the other hand, unstructured and semi-structured data allow a higher degree of flexibility. In this paper we compared unstructured, semi-structured and fully structured information and discussed an application design which combines all three types of data, based on a relational database system combined with an RDF triplestore. We illustrated this design on the concrete implementation of the semantic wiki *KiWi*. We saw that a challenge for such an application is to avoid states of inconsistency and present three different layers where a synchronisation of data within an application could be implemented:

1 On a low level database layer,
2 On the he O/R mapping layer, and
3 On the application layer.

In *KiWi* the synchronisation of data is implemented on the application layer because it offers database independence and enables the implementation of a common query language for all different data stores.

[15] http://www.w3.org/TR/rdf-sparql-query/

References

1. Blumberg, R., Atre, S.: The Problem with Unstructured Data. http://www.dmreview.com/issues/20030201/6287-1.html (19.02.2009) (2003)
2. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (1999)
3. Manola, F., Miller, E.: Resource Description Framework (RDF):Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (19.02.2009) (2004)
4. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (19.02.2009) (2004)
5. Schaffert, S., Sint, R., Grünwald, S., Stroka, S.: The KiWi Architecture. (2008)
6. Allen, D.: Seam in Action. Manning Publications Co. Greenwich, CT, USA (2008)
7. Bauer, C.: Java Persistence with Hibernate. Manning Publications Co. Greenwich, CT, USA (2006)
8. DeMichiel, L., Keith, M.: JSR 220: Enterprise JavaBeansTM,Version 3.0. http://java.sun.com/products/ejb/docs.html (20.02.2009) (2006)
9. Cheung, S., Matena, V.: Java Transaction API (JTA). http://java.sun.com/javaee/technologies/jta/index.jsphttp://java.sun.com/javaee/technologies/jta/index.jsp (19.02.2009) (2002)
10. : javax.persistence.EntityTransaction Interface JavaDoc. http://java.sun.com/javaee/5/docs/api/javax/persistence/EntityTransaction.html (11.02.2009) (unknown)
11. Connolly, T., Begg, C.: Database Systems: A Practical Approach to Design, Implementation, and Management. Addison Wesley Publishing Company (2005)
12. Stroka, S.: Transaction Management in Federated, Heterogeneous Database Systems for Semantic Social Software Applications. (2009)
13. Francois Bry, Michael Eckert, J.K., Weiand, K.: What the User interacts with: Reflections On Conceptual Models For Semantic Wikis. (2009)
14. Leigh, J.: Elmo User Guide. http://www.openrdf.org/doc/elmo/1.4/user-guide/index.html (19.02.2009) (2008)
15. Virtuoso: Virtuoso Universal Server. http://virtuoso.openlinksw.com (2009)

WIKITAAABLE: A semantic wiki as a blackboard for a textual case-based reasoning system

Amélie Cordier[1], Jean Lieber[2], Pascal Molli[2],
Emmanuel Nauer[2], Hala Skaf-Molli[2], Yannick Toussaint[2]

[1] Université de Lyon, CNRS,
Université Lyon 1, LIRIS, UMR5205, F-69622, France.
{firstname.lastname}@liris.cnrs.fr
[2] INRIA Nancy -Grand Est
Nancy Université
LORIA – UMR 7503. B.P. 239, F-54506 Vandœuvre-lès-Nancy cedex, France.
{firstname.lastname}@loria.fr

Abstract. Semantic wikis enable a community of users to produce formalized knowledge readable and usable by machines. To take one step further, one can use a semantic wiki as a blackboard allowing humans and machines to interact in order to build knowledge that is useful for both humans and machines. In this paper, we present a case study of the use of a semantic wiki (Semantic Media Wiki) as a blackboard to manage culinary data and knowledge. This case study is performed within the context of the TAAABLE application, a case-based reasoning web system aiming at solving cooking problems on the basis of existing recipes. With WIKITAAABLE, an evolution of TAAABLE based on a semantic wiki, we show how a semantic wiki assists users in their knowledge management tasks by taking into account user feedback. The issues related to the integration of several knowledge management mechanisms in a single application are discussed at the end of the paper.

1 Introduction

Wikis have demonstrated how it is possible to transform a community of strangers into a community of collaborators. By integrating Semantic Web technologies, semantic wikis [1–3] allow this new community of contributors to produce formalized knowledge readable by machines. Maybe the next step is to use semantic wikis as a blackboard where humans and machines can interact together for producing knowledge that is updatable by humans and machines.

We investigate in this paper how this can be achieved using the TAAABLE system [4] as a case study. The TAAABLE system[3] allows users to query a cooking recipe base to solve cooking problems. For example, a user can submit the following request: "I want a dessert with rhubarb but without chocolate". If no recipe exists with the specified characteristics, an existing recipe is adapted

[3] The TAAABLE system is accessible online at http://taaable.fr/

in order to answer the request. The system relies on a case-based reasoning (CBR [5]) engine to perform adaptation. The CBR engine uses different data and knowledge sources: a set of indexed recipes, an ontology of ingredients, types of dishes, geographical origins of dishes and types of diets (vegetarian, nut-free, no alcohol) and a set of adaptation rules. The indexed recipes are computed from recipes written in natural language. An indexing tool uses the different ontologies to index the recipes.

If the system is working fine, the maintenance of recipe base and knowledge is cumbersome. The indexing tool performs natural language processing which is error prone. In addition, the system is not able to capture the users feedback to improve its internal adaptation capabilities.

In this paper, we study how a semantic wiki can be helpful in the TAAABLE system for managing data (e.g. cooking recipes, terminological information in the domain of cooking) and knowledge (e.g. ontology and adaptation rules) feeding the CBR engine.The semantic wiki is used as a blackboard where humans and computers interact to produce knowledge. Humans can add new recipes, correct indexing of existing ones and give feedback about the results of adaptation. Machines can perform adaptation of users' queries and indexing of recipes. Machines can also take into account user feedback in order to improve adaptation and indexing.

The paper is organized as follows: section 2 describes the current application. Section 3 describes how we are building the WIKITAAABLE system based on Semantic Media Wiki. The last section concludes the paper and points out future works.

2 The TAAABLE project

The TAAABLE project was initially designed to participate to the Computer Cooking Contest[4] (CCC) challenge, an international scientific challenge that aims the confrontation among systems able to solve cooking problems. Candidate systems must be able to answer queries expressed in natural language by retrieving or adapting existing recipes given the user constraints. The recipe book, common to all participated systems, is a set of textual recipes described by a title, an ingredient list, and a set of preparation instructions. User requests may include constraints on ingredients, dish types and dish origins.

The TAAABLE project involves researchers interested in various knowledge-based systems fields such as case-based reasoning (CBR), ontology engineering, data and text mining, text indexing and hierarchical classification. TAAABLE entered the CCC in 2008 and won the vice-champion award. The ongoing researches on this project aim at improving the efficiency and the possibilities of evolution of the TAAABLE application.

In the next section, we detail how the TAAABLE application addresses the retrieval and the adaptation of recipes.

[4] http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc

2.1 The TAAABLE application

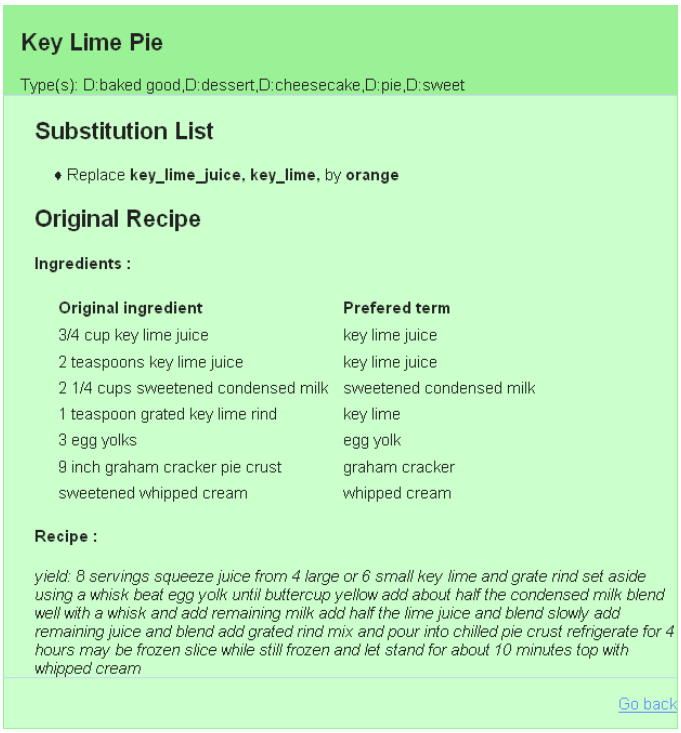
Fig. 1. TAAABLECBR web user interface.

The web user interface allows the user to enter her query. The system answers a query by returning a set of recipes satisfying the user's query. If adaptations are needed, they rare displayed to the user. Figure 1 shows an example of response for a "pie with orange".

Adapting a recipe consists in replacing some ingredients by some others. By clicking on a given recipe, the user will reach the recipe, including the list of ingredients that have to be substituted. In figure 2, the *Key Lime Pie* recipe is adapted by replacing *key lime juice* and *key lime* by *orange*.

The architecture of the TAAABLE system is composed of two distinct parts. The offline part of the system focuses on the management of the knowledge base and the indexing of the CCC recipes. Specific tools have been developed for that purpose. The result of the offline part is a set of data that will be used for bootstrapping the CBR engine plugged behind the web user interface (in the online part).

The design of these two parts is detailed in the next section.



Key Lime Pie

Type(s): D:baked good,D:dessert,D:cheesecake,D:pie,D:sweet

Substitution List

♦ Replace **key_lime_juice, key_lime,** by **orange**

Original Recipe

Ingredients :

Original ingredient	Prefered term
3/4 cup key lime juice	key lime juice
2 teaspoons key lime juice	key lime juice
2 1/4 cups sweetened condensed milk	sweetened condensed milk
1 teaspoon grated key lime rind	key lime
3 egg yolks	egg yolk
9 inch graham cracker pie crust	graham cracker
sweetened whipped cream	whipped cream

Recipe :

yield: 8 servings squeeze juice from 4 large or 6 small key lime and grate rind set aside using a whisk beat egg yolk until buttercup yellow add about half the condensed milk blend well with a whisk and add remaining milk add half the lime juice and blend slowly add remaining juice and blend add grated rind mix and pour into chilled pie crust refrigerate for 4 hours may be frozen slice while still frozen and let stand for about 10 minutes top with whipped cream

Go back

Fig. 2. An example of recipe adaptation.

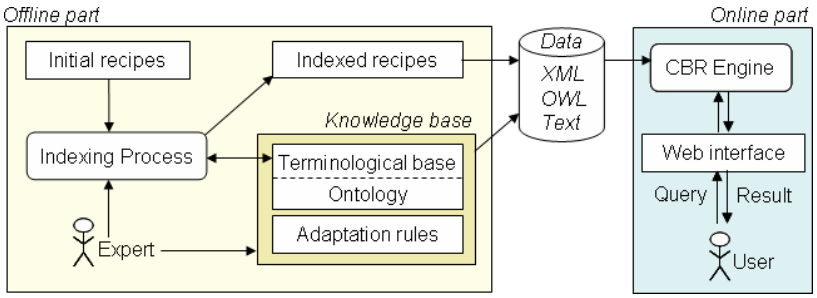


Fig. 3. Architecture of the TAAABLE system.

2.2 Knowledge Organization

A hierarchy of ingredients. The CBR engine adapts a recipe by substituting some ingredients by other ones. However, for sake of simplicity, each ingredient involved in the adaptation process of a recipe is replaced by one ingredient (1 to 1 substitution). The underlying idea is that a recipe can be adapted by substituting an ingredient by another "close" ingredient. Therefore, ingredients

are organized in an ontology. This ontology is used by the CBR engine to calculate the cost of a substitution: the closer the ingredients, the lower the cost. For instance, *orange* is closer to *lemon* than *apple*.

Hierarchies of recipe types. In order to type recipes, three other hierarchies are defined. *Dish moments* such as appetizer, dessert, . . . , *dish types* such as cake, pizza, . . . and finally *dish origins* such as Mediterranean, Chinese, ...

We call **ontology** in the following, the four above hierarchies. The ontology describes concepts of the domain. The link between the conceptual structure (the ontology) and the linguistic level (recipes) is performed thanks to the **terminological base**: each concept of the ontology is associated with its linguistic forms, *i.e.* a set of term variations. For instance, the concept *litchi* is associated to the terms *litchi*, *lichi*, *lychee*, *leechee*, and *laichee*.

The ontology and the terminological base have been built jointly by the experts. Depending on the results of the indexing process (see below) which may highlight lacks in the terminology, the expert manually decides which terms and concepts (associated to terms, if needed) should be added and the place of the concept in the hierarchy.

In addition to the ontology, the CBR engine may use **adaptation rules** to adapt a recipe. An adaptation rule is an ordered pairs of ingredients sets (s_1, s_2) with a cost given by the expert. (s_1, s_2) stating that the set of ingredients s_1 can be replaced by the set of ingredients s_2.

Adaptation rules and the ontology form the **knowledge base**.

2.3 Indexing the recipes according to the terminological base

The **indexing process** is an automatic process which creates the indexing of the recipes according to the terminological base. It takes as entry recipes in their initial textual form. For each recipe, the output of the indexing process is (1) tagged recipes: an XML (textual) form of the recipe where terms from the terminological base are tagged, (2) a list of concepts indexing recipes written in propositional logic (3) Error reports: a set of ingredient lines in the initial recipe where no term from the terminological base has been identified (possible lack in the base). We detail below these points.

Tagged recipes. The CCC recipes are given in a loosely structured XML format: tags are used for identifying the recipe title (`TI` element), the ingredients (`IN` elements), and the preparation (`PR` element). The indexing process adds tags to the ingredient part of recipes. It searches into the ingredient lines (tagged `IN`) the presence of terms of the terminological base, and introduces for each term the concept involved. The ingredient line `<IN>300 g mashed bananas</IN>` will be tagged as `<IN><ING>banana</ING><QT>300</QT><U>g</U><QL>mashed</QL><R/></IN>` where `<ING>banana</ING>` is the concept associated with the term bananas, `<QT>300</QT>` is the quantity, `<U>g</U>` is the unit, `<QL>mashed</QL>` is a "qualifier", `<R/>`is the rest of the ingredient line not recognised by the parser.

Here it is empty. The tagged line is used by the expert to control the correctness of the parsing.

Types (dish types, origins, and moments) of a recipe are not explicitly mentionned in the initial form of a recipe. It is automatically computed by three steps process. First, it searches if a recipe with the same title exists in the *recipesource.com* web database with some type information. If it fails, it searches if the title of the recipe contains terms that represents a type of dish or an origine (e.g. *Banana Butterfinger* **cake**). Finally, if step 2 fails, the process uses a set of association rules *ingredient(s)* $\rightarrow$ *type or origin* (e.g. *mascarpone* $\wedge$ *coffee* $\rightarrow$ *tiramisu*) to type the recipe. Association rules have been previously extracted from the complete *recipesource.com* web database.

Type indexation is quite noisy. 30% of recipes are not assigned to any type, some types (moment ...) are missing, and there are some errors (e.g., "pizza sauce" is not a *pizza*). The experts have to check manually these tags for each recipe.

Indexing recipe in a propositional form. The knowledge representation language used by the CBR engine is a fragment of propositional logic. The ontology is encoded as a set of axioms $a \Rightarrow b$. For example, the axiom $\texttt{apple} \Rightarrow \texttt{fruit}$ of O states that any recipe with apples is a recipe with fruits.

All recipes of the recipe book are indexed by a conjunction of literals. For example, the recipe of the apple pie (denoted by R) is indexed by:

$$Idx(R) = \texttt{apple} \wedge \texttt{pie} \wedge \texttt{sugar} \wedge \texttt{pastery}$$

The set of indexed recipes constitutes the case base of the system.

The indexed recipes resulting from the indexing process and the ontology are exploited by the CBR engine to answer user queries.

Error reports. We deal here with two main types of errors. One is coming from bad writing of recipes. The other one is due to concept missing in the ontology. All these errors should be corrected for the CBR engine to run properly.

Bad writing of recipes. Parts of recipes need corrections because of different typographic mistakes:

- `<IN>4 ts Baking power</IN>` should be corrected into `<IN>4 ts Baking powder</IN>`,
- the two consecutive ingredient elements `<IN>1 lb Boneless pork, cut in 3/4</IN><IN>Inch cubes</IN>` should be merge in one ingredient line `<IN>1 lb Boneless pork, cut in 3/4 inch cubes</IN>`,
- `<IN>Salt; pepper, Worcestershire and lemon juice</IN>` should be split into `<IN>Salt</IN><IN>pepper</IN><IN>Worcestershire</IN><IN>lemon juice</IN>`).

Most of these errors have been detected by the experts while checking the tagged recipes.

Missing concepts in the ontology. Some ingredient lines were not indexed by any terms of the terminological base. This means that no ingredient concept was recognised in this line. The expert has to check first that the line is correctly written. If it is correct, the error comes from missing concepts in the ontology or from missing terms in the terminological base. For example, if the `spam` concept [5] does exist in the ingredient hierarchy then the ingredient line `<IN>1/2 cn Spam, in 3/4" cubes</IN>` cannot be indexed. The expert has to add the new concept `spam` in the ontology, defines its position in the hierarchy, and associates this concept to a list of terms (*spam* ...) in the terminological base. Otherwise, if the concept exists in the ontology but the term is not recognised. Then, only the terminological base has to be updated. This error will be corrected at the next run of the indexing process. The same problem exists for the hierarchies of types, moments and origins.

2.4 Case-based reasoning

Querying the system. A request expressed in natural language is processed through the web interface and is formalized in the system by a conjunction of literals. For example, in TAAABLE, the request "I want a dessert with rhubarb but without chocolate" is transformed in a query, denoted Q:

$$Q = \texttt{dessert} \wedge \texttt{rhubarb} \wedge \neg\texttt{chocolate}$$

Retrieval and adaptation mechanism. The first step of the CBR process consists in retrieving among the available recipes, a recipe "similar" to the query. The retrieval is performed by classification on the basis on the query and the recipes indexes. First, a strong classification is applied: the system searches for a recipe whose index matches exactly the index of the query. If strong classification fails, a smooth one is applied: the query is generalized until a satisfactory solution is found [6]. Smooth classification leads to an approximate matching of the results and implies an adaptation of the retrieved recipe for answering the query.

The adaptation of a recipe uses adaptation knowledge. In the first version of TAAABLE, adaptation knowledge is only given by ingredient substitutions. For example, an apple pie recipe can be adapted in a rhubarb pie recipe by substituting apples by rhubarb in the recipe. However, one might like to perform a more complex adaptation of this recipe. This will be possible in a future version of the application. For example, the adaptation of the apple pie recipe will be performed by substituting rhubarb to apples and by adding sugar to the recipe in order to make the recipe less sharp.

2.5 Synthesis

Obviously, the knowledge base used by the CBR engine is neither correct nor complete and needs to be updated and improved. However, due to the independence of the two parts of the TAAABLE system, it is impossible to take into

[5] *Spam* stands for *spiced ham*, a kind of precooked canned meat.

account interactively the user feedback to make evolve the knowledge base of the system. This is a significant limitation of the current architecture. From a more practical point of view, it has also been a limitation during the development of the first version of TAAABLE: we were not able to improve the knowledge base online and any modification was time-consuming.

Hence, for the next version of the application, the goal is to link the two parts, i.e., the CBR engine and the knowledge base management tools, in order to be more efficient and to be able to take into account user feedback in the application.

3 WIKITAAABLE: A semantic wiki for TAAABLE

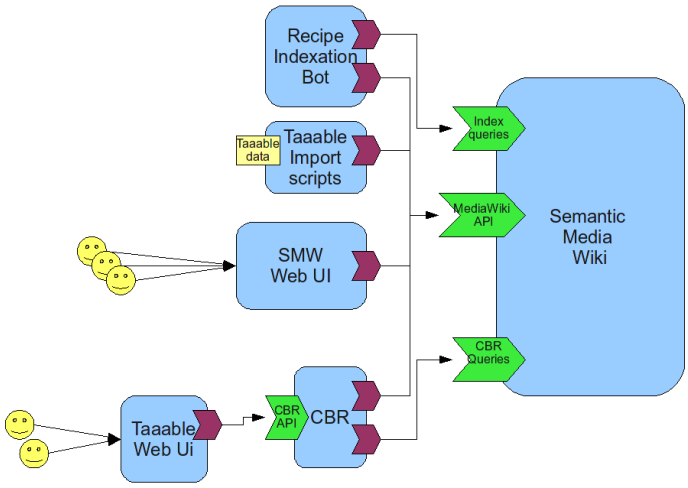


Fig. 4. WIKITAAABLE components.

In this section, we present the new generation of the TAAABLE system, called WIKITAAABLE.[6] In WIKITAAABLE, we address many problems of the TAAABLE application by using Semantic Media Wiki (SMW) [1] as a blackboard. The semantic wiki allows users to browse, query, edit, and validate the knowledge base as pointed out in [7]. In addition, the knowledge base can be updated by results produced by the CBR engine and by the automatic indexing tool. The architectural view of the system is presented in figure 4.

[6] WIKITAAABLE is not yet available for public because of the Computer Cooking Contest.

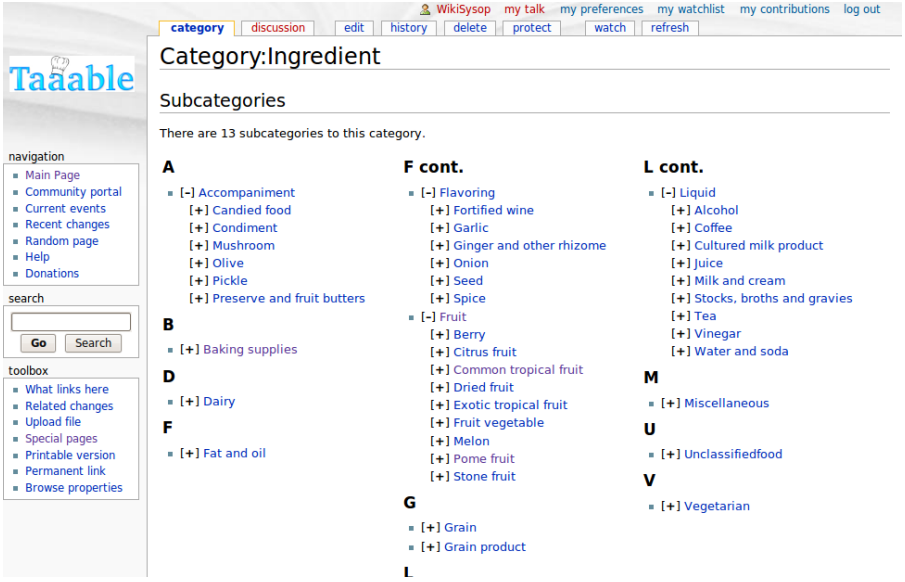


Fig. 5. WIKITAAABLE ingredient ontology.

Semantic Media Wiki. Semantic Media Wiki is used as a blackboard by users, the CBR engine and the recipe indexing bot. The CBR engine and the recipe indexing bot rely on a set of predefined semantic queries to gather their inputs. The knowledge base is represented by a graph of semantic wiki pages. We have represented as semantic wiki pages the indexed recipes, the ingredient ontology and the ontology of dish types (see figure 5).

Mediawiki Web User Interface. This is the regular user interface of semantic media. Through this interface, users can add new recipes and modify the ingredients, the types of dishes and the origins of dishes.

Recipe Indexing Bot. The recipe indexing bot crawls the recipe pages, extracts ingredient information, and updates recipe pages with semantic indexings and categorization of recipes. The crawling and updates of recipes is done using the mediawiki API, accessing the knowledge base is done using predefined semantic queries. The following example illustrates the input of the Recipe Indexing Bot:

```
== Ingredients ==
*   1 c rice
*   2 c water
*   1/2 c sugar
*   1 ts salt
*   2 c evaporated milk
*   1 c raisins
*   3 eggs separated
*   3/4 ts vanilla
```

Fig. 6. indexed recipe of "ARROZ DULCE".

```
*   1/4 ts cinnamon
*   1/4 ts nutmeg

== Preparation ==

* combine the rice water sugar and salt in a large saucepan bring the
  water to a boil and cover the saucepan reduce the heat to low and
  continue to cook for 12 - 15 minutes or until the water is absorbed
  combine the milk and egg yolk add them to the rice then mix in the
  raisin vanilla and cinnamon simmer for five minutes remove from the
  heat beat the egg white until stiff fold them into the rice chill and
  garnish with nutmeg before serving also taste good warm
```

The above recipe is indexed in the semantic wiki as presented in figure 6. We used the n-ary relationship of Semantic Media Wiki to represent an ingredient line.

CBR. The CBR engine retrieves its knowledge base through predefined semantic queries. Next, it is able to answer requests issued using the CBR web interface

Fig. 7. TAAABLE user feedback.

(cf. figure 1). Our idea is that the CBR user interface proposes recipes based on adaptation of existing recipes. Users are invited to give feedback about recipes. The figure 7 illustrates how we can capture the feedback of the user. This interface is currently not functional.

We imagine that the WIKITAAABLE will work like this: Suppose a user requests "pie" and "orange" thinking about an "orange pie". The system presents results as in figure 7. The user chooses "I don't like it" for the first proposal because replacing lemon juice by orange in an "apple crumble pie" does not transform an "apple crumble pie" into an "orange Pie". However, the proposal 2 and 3 are acceptable. So the user select "I like it" for these two proposals.

If the feedback is positive, then the new recipe is added to the knowledge base and a new semantic wiki page is created for it. The new recipe is marked as "generated". If the user feedback is negative, then the computed recipe is also added to the knowledge base and a new page is created. However,this new page belongs to the category "refused". These recipes are kept for future

reuse. For instance, experts can analysis refused recipes and use them as a basis for a failure driven knowledge acquisition [8].

TAAABLE Import Scripts. We have written scripts to import the current knowledge base of TAAABLE into Semantic Media Wiki. The recipe base contains about 888 recipes, the ingredient ontology contains 8506 different classes of ingredients. We used *RAP - RDF API for PHP* to parse the ontology.

4 Discussion and future work

The strength of the semantic wiki comes from the facility for the community to update and enrich a set of annotated recipes as well as the ontology. The WIKITAAABLE system has many advantages compared to the original one:

- Users can add new recipes.
- Users can correct indexation of recipes.
- Users can browse and navigate through the ontology.
- Ontology maintainers can quickly modify the ingredient ontology or the dish type ontology and test the effects on the CBR engine.
- The feedback about adaptation of recipes can now be captured and represented in the semantic wikis. The knowledge base of the system increases just by using it.

One of the reasons of the success of CBR systems is that they are theoretically able to "learn from experience" by acquiring additional knowledge with each problem solving session. In TAAABLE, however, learning from experience is difficult because of the lack of an embedded mechanism allowing to use feedback for improving existing knowledge bases. A strength of the use of Semantic Media Wiki in WIKITAAABLE is that it facilitates this process by enabling the enrichment and the update of data and knowledge by a community of users. The management of the ontology and the annotated recipes have many advantages compared to this of the previous version of the application. Indeed, users can add new recipes, correct indexing of recipes, and browse and navigate through ontologies. Ontology maintainers can as well easily modify the ontologies and test the impact of the modifications on the results of the CBR engine. Besides, a major advantage is that the feedback on the adaptations made by the CBR engine can be captured and represented in the wiki.

However, the development of WIKITAAABLE raises several issues that are mainly related to the coherence of the system. How can we guarantee the coherence of the systems while several users, often having different viewpoints, are allowed to modify the knowledge coded in the system? How can we efficiently combine a semi-automatic procedure and a manual enrichment process to build an ontology? These issues are of major importance because the ontology plays a central role in the TAAABLE system and is mainly used at two different levels. It guides the indexing process by identifying concepts involved in each recipe, and it is used by the CBR system to adapt recipes. If any user can freely modify the ontology, then the CBR engine and the recipe indexing bot might

produce unpredictable results. Several strategies can be envisioned to tackle this problem:

- Restrict the update of ontologies to "administrators". This is a limitation to the collaborative work. Moreover, the improvement of the ontology strongly depends on the availability of the administrators.
- Validation of changes before they are integrated in the running system. This strategy has also several limitations: poor process support by semantic wikis, synchronization problems between several versions of a single system, and time consuming for "administrators". Moreover, this does not solve the problem of conflicts between several concurrent changes.
- Adaptation of continuous integration approaches used in software engineering in the context of the WIKITAAABLE system. For example, if an adaptation of a recipe has been validated by several users, then the ontology should be modified in order to preserve this adaptation (in this case, user feedback collected by WIKITAAABLE should generate non-regression tests).
- Toward a peer-to-peer WIKITAAABLE? In such an approach, each user would have his/her own version of the application, relying on a common knowledge base, and would be able to perform personal adaptation of the knowledge. Adaptation would be shared between users on the basis on the confidence they have in their peers.

The next step of the development of WIKITAAABLE is to fully integrate the CBR engine to the Semantic Wiki and to set up interaction possibilities at several levels. One particular focus will be put on the ability of the semantic wiki to represent and manage complex adaptation rules, including Boolean constraints.

References

1. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic wikipedia. Journal of Web Semantics **5**(4) (2007)
2. Schaffert, S.: IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. 1st International Workshop on Semantic Technologies in Collaborative Applications (STICA06), Manchester, UK (2006)
3. Buffa, M., Gandon, F.L., Ereteo, G., Sander, P., Faron, C.: Sweetwiki: A semantic wiki. Journal of Web Semantics **6**(1) (2008) 84–97
4. Badra, F., Bendaoud, R., Bentebitel, R., Champin, P.A., Cojan, J., Cordier, A., Desprès, S., Jean-Daubias, S., Lieber, J., Meilender, T., Mille, A., Nauer, E., Napoli, A., Toussaint, Y.: TAAABLE: Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In Schaaf, M., ed.: ECCBR 2008, The 9th European Conference on Case-Based Reasoning, Trier, Germany, September 1-4, 2008, Workshop Proceedings. (2008) 219–228
5. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey (1989)
6. Lieber, J.: Strong, Fuzzy and Smooth Hierarchical Classification for Case-Based Problem Solving. In van Harmelen, F., ed.: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02), Lyon, France, IOS Press, Amsterdam (2002) 81–85

7. Krötzsch, M., Schaffert, S., Vrandecic, D.: Reasoning in semantic wikis. In Antoniou, G., Aßmann, U., Baroglio, C., Decker, S., Henze, N., Patranjan, P.L., Tolksdorf, R., eds.: Reasoning Web. Volume 4636 of Lecture Notes in Computer Science., Springer (2007) 310–329
8. Cordier, A.: Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning. PhD Thesis, Université Claude Bernard Lyon 1 (2008)

Engineering on the Knowledge Formalization Continuum

Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe

University of Würzburg, Am Hubland
97074 Würzburg, Germany

Abstract. Usually, domain knowledge is available at different levels of formality, for example such as documents, data bases, and (business) rules. We argue, that today's systems limit the knowledge engineering process to a fixed level of formality and expressiveness, respectively, and that these limitations hinder effective knowledge acquisition and use. In consequence, we introduce the *knowledge formalization continuum* as a metaphor, that embraces the fact that knowledge is available in different formalities. We motivate that a semantic wiki is a suitable tool to work on the knowledge formalization continuum, and we introduce KnowWE as an example wiki implementation.

1 Introduction

In today's enterprises we see that knowledge management systems and knowledge-based solutions are already implemented with reasonable success. There still exists a great deal of interest to build knowledge-based solutions. Typically, "knowledge" is already available in different representations ranging from technical documents, construction plans, sheets and experiences of human experts. However, the knowledge acquisition bottleneck, i.e., the problem of formalizing existing knowledge into a machine processable model, is still present and often prevents successful developments. Experiences in many projects over the last years showed that the implementation often faces differently favorable but conflicting options, thus creating the following dilemmas:

1. *The Single Expert Dilemma.* The motivation and sophistication of *single domain specialists* are often the driving forces of successful knowledge acquisition and evolution. Although, high-quality experts can guarantee the construction of a high-quality knowledge base, these persons are often short in time and motivation. A distribution of the workload would decrease this problem, but will open the risk of also reducing the overall quality of the formalized knowledge. Furthermore, the collaboration among a group of specialists is not supported sufficiently in many industrial systems concerning the distributed development of a knowledge base. Here, the dilemma exists of favoring a distributed over a monolithic development process.
2. *The Flexibility Dilemma.* Current state-of-the-art tools are often constrained to a specific knowledge representation and acquisition interface for developing the knowledge base. In consequence, the tools are commonly not flexible enough to map the mental model of the domain specialists that are responsible to formalize

the knowledge in the given application project. Additionally, "knowledge" can appear in diverse forms, such as textual and tabular data but also explicit rules. On the one hand, the mapping of the particular mental model of the specialists to the provided knowledge representation and interfaces, respectively, often turned out to be difficult, complex and time-consuming. On the other hand, a tool having the maximal flexibility regarding the user interfaces and provided knowledge representations, typically would increase the complexity of its use and therefore decreases the effectivity of the developers [1, p. 86]. In consequence, we face the dilemma of demanding for a tool with maximal flexibility vs. a tool with maximal productivity.

This paper introduces approaches to weaken the two dilemmas by first introducing the metaphor of a *knowledge formalization continuum* in order to give domain specialists a flexible mental model of the knowledge that is planned to be formalized. It frees the developers to commit to a particular knowledge formalization at an early stage, but offers a versatile understanding of the formalization process. Second, we propose the use of a *semantic wiki* as a suitable development tool for knowledge bases, that is able to scale from one single domain specialist to the collaboration of multiple domain specialists without changing the existing working process.

2 The Knowledge Formalization Continuum

A *continuum* can be seen as "a nonspatial whole in which no part or portion is distinct or distinguishable from adjacent parts"; alternatively a continuum can be understood as "anything that goes through a gradual transition from one condition, to a different condition, without any abrupt changes"[1].

We use these definitions of a continuum to explain the idea of a *knowledge formalization continuum*, where gradual transitions on formalization degrees of the same knowledge are possible, but where the knowledge to be modelled experiences no abrupt changes or "discontinuities".

The idea is quite simple: When starting with the development of a knowledge base, the considered knowledge is already available in varying forms, for example documents, recorded application cases or often in the heads of the experts. The usual task is to find an appropriate knowledge model which serves as the target of knowledge formalization. Here, often the issue arises, when some parts of the knowledge cannot be formalized into the selected model, or its formalization would be too costly with respect to the *cost/benefit principle* [1, p. 56]. The knowledge formalization continuum frees developers from putting the knowledge into a single, fixed representation scheme at the very beginning, but asserts that even text and figures are "first class" knowledge objects. The original nature of the knowledge itself remains the same no matter if it is represented by a textual document or by a rule base. Thus, the formalization from the textual form to an explicit rule base means a gradual transition, but the original nature of the considered knowledge remains unchanged.

It is important to notice that the knowledge formalization continuum is neither a physical model nor a methodology for developing knowledge bases. It rather should

[1] see WordNet/Wikipedia for full definitions/explanations

be seen as a metaphor of the knowledge development process. It helps the domain specialists to see even raw data, such as text and multimedia, as first-class knowledge that can be transformed by gradual transitions to more formal representations when required.

In summary, knowledge can be represented at different degrees of formality, and within the knowledge formalization continuum transitions of these degrees are proposed. In the extreme cases knowledge about a domain is given as data at a very informal level (images, text) or the knowledge is represented by formal knowledge representations such as decision trees or functional models. On the one hand, data given in *textual documents* denotes the lowest possibility of formality. On the other hand, *functional models* store knowledge at a very detailed formality. See Figure 1 for an example depiction of the different knowledge representations possible in the knowledge formalization continuum. This is certainly not an exhaustive enumeration of all possible representations of knowledge here, and the depicted order of representations between data and knowledge is not meant to be explicit. In fact, it appears difficult/impossible to define a total order of the representations in a general manner. The depicted order was motivated by the level of possible expressiveness with respect to the reasoning power of built system using the particular representation as knowledge. For example, text can be used for standard keyword-based search and retrieval, whereas semantically annotated text allows for semantic queries and navigation. At the right end, knowledge based on rules allows for even more complex reasoning capabilities.

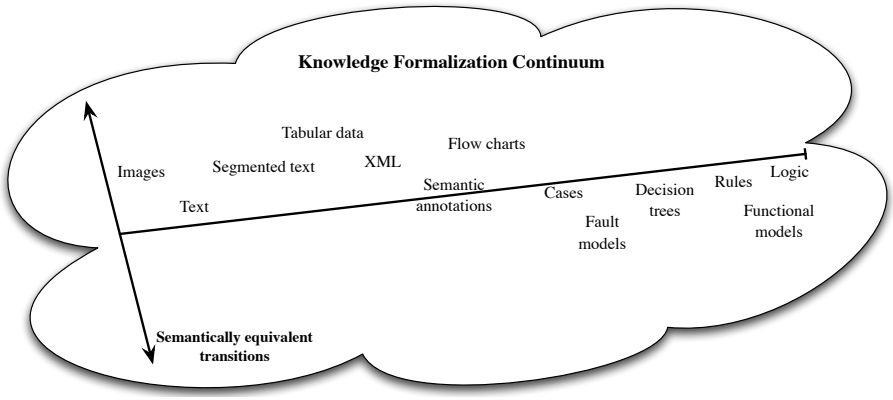


Fig. 1. Possible knowledge transitions within the knowledge formalization continuum.

Every level of formality has its own advantages and drawbacks. For example, textual knowledge can be easily elicited and often is already available in the domain. No prior knowledge with respect to tools or knowledge representation is necessary. However, automated reasoning using the textual knowledge is not possible with current state–of–the–art methods, and the knowledge can be retrieved only by using string-based matching methods but not by semantic queries. Logic rules or models are well-suited for auto-

mated reasoning, and queries can be processed on the semantic level. In contrast to textual knowledge, the acquisition of rules and models is a complex and time-consuming task. Usually, authors need prior training before effectively building knowledge bases on the explicit level with respect to knowledge engineering principles as well as tools that support such knowledge modeling. For a given knowledge base, that is formalized in a particular knowledge representation, there often exist semantically equivalent transitions (indicated by the second axis in Figure 1). For example, a fault model based on set-covering models can be often also represented in a rule base which in turn may be modelled by a special purpose logic dialect. However, representations on the right side are usually able to store more expressive knowledge. Often, the knowledge is brought to a semantically equivalent transition in order to simplify the extension by additional domain knowledge. For example, a knowledge base represented by fault models can be transferred to a rule base in order to allow for a fine-grained definition of conditioning findings for a target concept.

Between the two extremes (text vs. logic) there exists a wide range of formats representing knowledge at different degrees of formality. Any degree can be the most useful representation for building a knowledge base in a specific application project. For a given project it is an important and difficult task to select the most appropriate transition as the target representation. Since often (fragments of) knowledge are already available in textual or tabular form, the development process focuses on bringing the existing forms to an appropriate level. Although, it typically becomes necessary to fill in missing parts of the knowledge, the original nature of the knowledge remains. Thus, moving to a more formal transition can require the more explicit description of the knowledge and can enrich the resulting knowledge with additional semantics made explicit. It is worth noticing, that every transition is a distinct operation that modifies the knowledge representation. However, the mental model of the knowledge remains basically the same.

2.1 Methods for the Knowledge Formalization Continuum

The movement between two transitions is supported by already existing and established methods. Results from the following research areas can be applied, when going from explicit transitions to less explicit levels of knowledge:

- *Natural language generation* techniques, for example [2].
- *Visualization techniques*, for example [3].
- *Knowledge explanation* methods, for example [4].

The transition of the available knowledge to a less formal level is sometimes required for a number of reasons: For example, in commercial systems the built knowledge base needs to be reviewed by external specialists before deployed into practice. The transformation to a natural language text in addition to visualizations can help to present an understandable but precise version of the knowledge base for non-knowledge engineers. Furthermore, the presented methods are useful to produce a human-understandable output of the derived facts of the knowledge base, thus giving explanations of the system's behavior.

Typically, little structured/unstructured information is transformed to a more explicit level; here methods from the following disciplines will be helpful:

- *Text Mining*, *Ontology Learning*, and *Natural Language Processing* in general for the machine–enabled extraction of concepts from texts, their taxonomic ordering and the discovery of basic relations between found concepts, see [5] for example.
- *Controlled Languages* to automatically interpret a restricted subset of natural language text as logic formulas, for example an overview is given in [6].
- *Refactoring* methods to support manual changes of explicit knowledge without changing the intended semantics, for example [7]. They are often used to accomplish vertical transitions to a semantically equivalent version within the same knowledge representation, but are also helpful to restructure the knowledge to a less/more formalized level.
- *Manual Knowledge Elicitation* methods, that are applied when it is not reasonable or tractable to use (semi-)automated methods sketched above.

In an example application project we have knowledge already available contained in a textual form such as Word files and semi-structured Excel sheets. By using ontology learning methods we are able to extract relevant ontological concepts and basic relations afterwards. In subsequence, strongly formalized models are (manually) defined to formulate enhanced relations between the concepts. The initial textual knowledge is still available but now annotated by the added forms of formalized knowledge.

2.2 Implications

The knowledge formalization continuum embraces the fact that knowledge is usually represented at varying levels of formality. The continuum supports the entrance of the knowledge engineering process at an arbitrary level of formality and offers possible transitions of the knowledge to a level where its *cost/benefit principle* [1, p. 56] is (in the best case) optimal. In typical projects, prior knowledge of the domain is already at hand, often in form of text documents, spreadsheets, flow-charts and data bases. These documents build the foundational reference of the classic knowledge engineering process, where a knowledge engineer models the domain knowledge based on these documents in addition to further knowledge provided by domain specialists. The actual utility and applicability of the knowledge usually depends on the particular application.

The knowledge formalization continuum does not postulate to transform the entire collection into a knowledge base at a specific degree, but to perform transitions on *parts* of the collection when it is possible *and* appropriate. This takes into account that sometimes not all parts of a domain can be formalized at a specific level or that the formalization of the whole domain knowledge would be too complex, considering costs and risks. In consequence, a system working on the knowledge formalization continuum need to be able to support the knowledge engineering process at different levels of formality. However, it also should be able to support the knowledge sharing process, i.e., its actual usage, at varying formalization levels.

Following, the *cost/benefit principle* it need to be possible to transform the parts of the knowledge to a level of formality, where the (knowledge engineering) costs are minimized and the benefits of using the system are maximized. Therefore, the knowledge

formalization continuum not only needs to support the transitions of particular parts of the knowledge but also should be able to keep references between the less and more formalized parts of the entire knowledge collection.

3 A Semantic Wiki as an Integrated Tool to Support the Knowledge Formalization Continuum

We motivate that an extensible semantic wiki is useful to serve as a knowledge engineering tool on the knowledge formalization continuum, since it allows the integration of knowledge at different levels of formality. Thus, it tries to weaken the *flexibility dilemma* described in the introduction. The use of a semantic wiki additionally helps to target the first dilemma, i.e., the *single expert dilemma*. A semantic wiki naturally allows for a distribution of the development process over a group of domain specialists due to its open and web-based implementation. Collaboration is supported by many standard features of wikis, for instance versioning and discussion pages. However, the dilemma is only weakened by providing a technical platform for a collaborative engineering process, the interesting question of how to ensure a certain level of quality of the knowledge remains, and needs to be solved by appropriate evaluation methods, for example see [8]. The following example demonstrates the idea of the knowledge formalization continuum by a possible engineering trail of a recommendation system that is built using the semantic wiki KnowWE [9].

3.1 The Semantic Wiki KnowWE

In most semantic wikis every concept is represented by a distinct wiki page, where the concept is described by textual documents and multimedia content. Text phrases are semantically annotated with properties of a given wiki ontology; in most cases new properties can be defined in an ad-hoc way. Recent examples of semantic wikis are Semantic MediaWiki [10], IkeWiki [11], and SweetWiki [12]. The semantic knowledge wiki KnowWE [9] further allows for the intuitive capture and use of explicit problem-solving knowledge that is applied to derive particular concepts. In addition to the possibility to express strong problem-solving knowledge, such as rules and models, it also provides alternative interfaces to engineer knowledge at lower levels of formality.

Figure 2 shows a sports advisor wiki, which is an example system for demonstrating the functionality of KnowWE. Here, the form of sports *Swimming* is shown by a describing text, a picture and interactive elements within the text. A visitor can use the wiki as a recommender system in order to get a proposal of a sports form for an entered user profile. For example, the vistor enters new facts by clicking on links in the system; in the shown example (2a) the user enters some values for the question *Motivation*. The system instantly derives solutions (recommendations) when new facts (attributes of the user profile) are entered. Here, the solutions can be inferred based on the given findings (2b). All appropriate solutions are shown in the right pane of the wiki, for example the solution *Swimming* was derived as a suitable solution, but the solution *Jogging* is also suggested for further consideration. By clicking on the solution names the user can easily navigate to the corresponding wiki articles describing the sport forms in more detail.

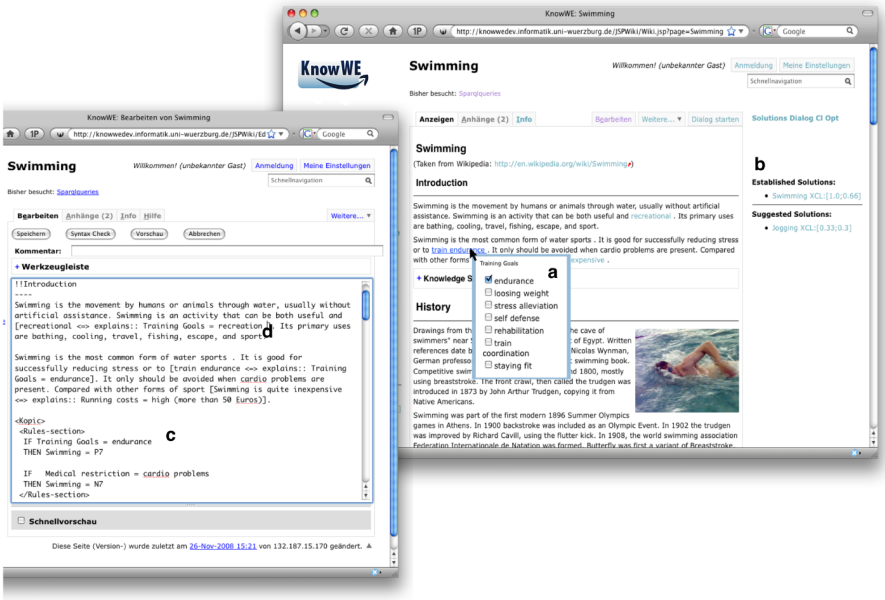


Fig. 2. The semantic wiki KnowWE at a glance: Interactive interviews with the user (a) are used to derive new concepts as solutions (b). Knowledge is entered, for example, by inline annotations (d) or explicit problem-solving knowledge such as rules (c).

The derivation knowledge for every solution is entered together with the remaining content of the wiki article. By clicking the edit button the wiki page and its corresponding content, respectively, can be modified. Here, the user can insert a special knowledge topic (Kopic) into the standard text, for example, to enter rules describing the domain knowledge (2c), but he is also able to semantically annotate particular text phrases with concepts of the application ontology (including solutions and findings of the shown sports advisor example, see 2d). Existing annotations are used for inline answers in the view mode of the wiki.

3.2 Building a Sports Recommender based on the Knowledge Formalization Continuum

The following example shows subsequent steps that describe transitions within the knowledge formalization continuum. We use the sports advisor demonstration sketched above in our example. Here, the knowledge already available in the continuum describes relevant facts about the forms of sports, such as accomplished training goals, costs, and medical restrictions. In subsequent steps we drive the existing knowledge to more formalized transitions.

Initial Filling. We start by filling the wiki with text and multimedia (pictures and movies) describing the different forms of sport, for example *Running*, *Swimming*, and

Cycling. It is reasonable that for every form of sports a wiki page should be created, i.e., after the initial filling phase there exist pages about running, swimming, and cycling. However, also wiki articles about further domain facts exist, for example allergies or muscles. In general, it is reasonable to set up one distinct wiki article for each distinct concept of the domain, thus following a common paradigm of (semantic) wikis. For example, an excerpt of an article about swimming is as follows

> ...Swimming is the most common form of water sports. In particular it is recommended for people with back problems because it trains the back muscles ... However, people with skin allergies should avoid swimming. ...

At this point the wiki can be used as a simple and traditional information system specialized on sports, where users can search and browse through the available content.

Annotating Articles. We propose to annotate every wiki article with its semantic concept, thus making explicit that a specific article *is about* a specific concept. For instance, we annotate the article about swimming with the concept *Swimming*. At this point, only a very general ontology of concepts is required to represent the domain concepts already contained in the wiki. As a benefit of this step it becomes possible to offer a low-end version of semantic search and navigation, that will be more useful when concepts are carefully structured in a hierarchy.

Annotation by Properties. The next step tries to identify the typical features of every concept described in the available text. These findings are then annotated as properties of the article's concept. In the example above the text about swimming would then transform as follows (new/changed text is given in bold letters):

> ...Swimming is the most common form of **[hasFinding::water sports]**. In particular it is recommended for people with back problems because it **[hasFinding::trains the back muscles]**. ...
> However, people with **[isContradictedBy::skin allergies]** should avoid swimming. ...

In the given example, the text phrases *water sports*, *trains the back muscles*, and *skin allergies* are annotated by the properties *hasFinding* and *isContradictedBy*, respectively. Each annotation performs the creation of an RDF triple with the article's concept (here, *Swimming*) as the subject, the property's name as the predicate, and a reference to the particular text phrase as the object. The use of properties implies the extension of the simple domain ontology of sport forms defined before. In the given example, we introduced the properties *hasFindings* and *isContradictedBy*. With the properties defined in the wiki an extended version of semantic search and navigation becomes possible. For example, we are now able to query findings (as text phrases) that exclude a specific form of sport, i.e., "return all text phrases that represent the contradiction of a given sports form".

In a further step, it is reasonable to "semantify" the text phrases representing the particular properties of a concept. Thus, we gradually extend the existing annotation by explicit concepts describing the ranges of the properties.

...Swimming is the most common form of water sports [hasFinding:: **Medium = in water**]. In particular it is recommended for people with back problems because it trains the back muscles [hasFinding:: **Trained muscles = back**]. ... However, people with skin allergies [isContradictedBy:: **Medical restrictions = skin allergy**] should avoid swimming. ...

In the shown example, the text phrase *trains the back muscles* is moved out of the annotation and replaced by the explicit concept *Trained muscles* having a concrete value *back*. Furthermore, the last annotation describes that the text phrase *skin allergies* is annotated by the value *skin allergy* assigned to the concept *Medical restrictions*. This implies the extension of the ontology by appropriate concepts representing the findings for the different forms of sport. If these concepts are defined in advance, then natural language processing methods can be used for a semi-automatic annotation of the text. In consequence, a full-fledged semantic search and navigation becomes possible, where the relation of a specific finding value to all available sport concepts can be queried, for example.

Generation of Explicit Problem-Solving Knowledge. In some cases, the use of semantic annotations is not sufficiently expressive for a given application project. Then, it becomes necessary to transform to a higher level of formality by generating and extending strong problem-solving knowledge out of the existing annotations. In the following we aim to define knowledge to actually *derive* particular forms of sports based on entered user findings. For this reason, we collect all properties, that set a form of sport in relation with a finding that can be entered by the user. In the given example, we collect the properties *hasFinding* and *isContradictedBy*. The semantic wiki KnowWE offers scripts that automatically convert these properties either into set-covering models or rules. For further properties with a different semantics the scripts certainly need to be adapted. In the initial step, such a conversion denotes the transition of the available knowledge into an (almost) semantically equivalent version. However, in this case the target representation allows for richer possibilities to represent further elements of the knowledge base.

Set-Covering Models. The following shows a transition of the annotation to a set-covering model [13], where a set-covering model describes all typical/relevant findings for a solution. The given textual markup to be used in wikis was introduced in [14]. In our example, the solution concepts are corresponding to the concepts representing the wiki articles, and findings are defined as the target concepts of the included properties. Each of the collected properties is compiled by the script into a line of the set-covering model. The value of a *hasFinding* property is represented as a simple line (denoting the positive expectation of this finding), for example *Trained muscles = back*. For a *isContradictedBy* property the conversion additionally adds a *[- -]* at the end of the generated line in order to represent the negative expectation of this finding, for example see *Medical restrictions = skin allergy*.

```
Swimming {
   Medium of sports = water
   Type of sport = individual
   Trained muscles = back
   Running costs >= medium
   Medical restrictions = skin allergy [- -]
}
```

Bold-faced letters are (hand-crafted) additions to the model, that have been made after the transition. For instance, two further findings are describing the type of sport and the running costs. The explicit representation in the model points to an extension of the formalized knowledge, although this information is already available in the text of the wiki article.

Rules. In the following example block, a simple rule-based version of the annotations made is shown. In this simple example, one rule is created by a script collecting all *hasFinding* properties as well as one rule for every *isContradictedBy* property. Of course, this simple conversion not necessarily conforms with the intended semantics of the made annotations and therefore is meant as a starting point for further (manual) adaptations.

```
if Medium of sports = water
   and Type of sport = individual
   and Trained muscles = back
   and Running costs >= medium
then derive Swimming

if Medical restrictions = skin allergy
then exclude Swimming
```

The transition to a more expressive knowledge representation such as set-covering models and rules becomes necessary when artifacts of the domain cannot be expressed by semantic annotations anymore. As a benefit, the knowledge can then be used for more effective reasoning ranging from complex semantic queries to the generation of problem-solving interviews, where appropriate solutions for a given problem are derived based on an interactive interview.

4 Conclusions

Domain knowledge is commonly available at different levels of formality. We introduced the knowledge formalization continuum to cope with this problem, and we sketched methods to work with the continuum. The semantic wiki KnowWE was introduced as a tool to support the knowledge formalization continuum, whereas many other semantic wikis also can be used to serve as suitable platform. The presented idea of the knowledge formalization continuum is related to the ontology classification using a three-dimensional matrix as introduced in [15]. Here, a categorization of the knowl-

edge to be modelled is given when designing a knowledge-based system. Schaffert et al. distinguish between *model scope*, *model acceptance* and the *level of expressiveness*, where the latter defines a subspace of the presented knowledge formalization continuum. The level of expressiveness ranges from light-weight ontologies, with term lists as the least expressive one, to heavy-weight ontologies with very-expressive constraints as the most expressive representative. Whereas functional models and logic programs can be interpreted as "very-expressive constraints" in some ways, the knowledge formalization continuum also considers textual documents as less expressive occurrences of knowledge apart from term lists.

References

1. Lidwell, W., Holden, K., Butler, J.: Universal Principles of Design. Rockport Publishers (October 2003)
2. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press, Cambridge (2000)
3. Geroimenko, V., Chen, C., eds.: Visualizing the Semantic Web. 2 edn. Springer (2006)
4. Roth-Berghofer, T.R.: Explanations and Case-Based Reasoning: Foundational issues. In Funk, P., González-Calero, P.A., eds.: Advances in Case-Based Reasoning, Springer-Verlag (September 2004) 389–403
5. Dale, R., Moisl, H., Somers, H., eds.: A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text. Marcel Dekker Inc. (2000)
6. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for knowledge representation. In Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web, Fourth International Summer School 2008. Number 5224 in Lecture Notes in Computer Science, Springer (2008) 104–124
7. Baumeister, J., Seipel, D., Puppe, F.: Refactoring methods for knowledge bases. In: EKAW'04: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, LNAI 3257, Berlin, Springer (2004) 157–171
8. Baumeister, J., Nalepa, G.J.: Verification of distributed knowledge in semantic knowledge wikis. In: FLAIRS'09: Proceedings of the 22th International Florida Artificial Intelligence Research Society Conference. (2009)
9. Reutelshoefer, J., Baumeister, J., Puppe, F.: Ad-hoc knowledge engineering with semantic knowledge wikis. In: SemWiki'08: Proceedings of 3rd Semantic Wiki workshop - The Wiki Way of Semantics (CEUR Proceedings 360). (2008)
10. Krötzsch, M., Vrandecić, D., Völkel, M.: Semantic MediaWiki. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273, Berlin, Springer (2006) 935–942
11. Schaffert, S.: IkeWiki: A semantic wiki for collaborative knowledge management. In: STICA'06: 1st International Workshop on Semantic Technologies in Collaborative Applications, Manchester, UK (2006)
12. Buffa, M., Gandon, F., Ereteo, G., Sander, P., Faron, C.: SweetWiki: A semantic wiki. Web Semantics **8**(1) (2008) 84–97
13. Peng, Y., Reggia, J.A.: Abductive Inference Models for Diagnostic Problem-Solving. Springer, Berlin (1990)
14. Baumeister, J., Reutelshoefer, J., Puppe, F.: Markups for knowledge wikis. In: SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop, Whistler, Canada (2007) 7–14
15. Schaffert, S., Gruber, A., Westenthaler, R.: A semantic wiki for collaborative knowledge formation. In: Proceedings of SEMANTICS 2005 Conference, Trauner Verlag (2006)

MoKi: the Modelling wiKi

Marco Rospocher[1], Chiara Ghidini[1], Viktoria Pammer[2], Luciano Serafini[1], and Stefanie Lindstaedt[3]

[1] FBK-irst, Via Sommarive 18, 38123 Trento Povo, Italy
[2] Knowledge Management Institute, TU Graz. Inffeldgasse 21a, 8010 Graz, Austria
[3] Know-Center, Inffeldgasse 21a, 8010 Graz, Austria.

Abstract. Enterprise modelling focuses on the construction of a structured description of relevant aspects of an enterprise, the so-called *enterprise model*. Within this contribution we describe a wiki-based tool for enterprise modelling, called MoKi (Modelling wiKi). It specifically facilitates collaboration between actors with different expertise to develop an enterprise model by using structural (formal) descriptions as well as more informal and semi-formal descriptions of knowledge. It also supports the integrated development of interrelated models covering different aspects of an enterprise.

1 Introduction

An enterprise model is "a computational representation of the structure, activities, processes, information, resources, people, behavior, goals, and constraints of a business, government, or other enterprise" [1]. Often, an enterprise model focuses in the description of two specific aspects of an enterprise: (i) its processes and activities, and / or (ii) the business domain within which the enterprise operates. Other aspects of an enterprise, like goals, human resources, organisational structure and roles, competencies, etc. may also be important assets to be described in an enterprise model. This is due to the central role that enterprise models are playing in the development of a large number of applications, including Internet and (Semantic) Web based applications.

Building an enterprise model requires a number of skills. These skills span from *knowing* the different aspects that have to be described in the models to having the ability of *encoding* such knowledge into formal statements, to having the ability of *integrating* different aspects, such as structure, activities, processes, information, resources, people, behaviour, goals, and constraints into a uniform and coherent vision. Given the complexity of enterprise modelling, it is unrealistic to assume that any one person possesses all the above skills, and the contribution of multiple actors is necessary. For this reason enterprise modelling is inherently a *collaborative* activity. Our research focuses mainly on collaboration between actors with different skills. Naturally we also recognise the relevancy of other aspects of collaboration such as resolution of conflicts of opinion or interest (considered for example in Collaborative Protégé [2]), or more fundamental requirements regarding access rights, simultaneous modification of

models, versioning etc., but we plan to consider them at a later stage of our work.

To support actors with different skills, we envisage a system in which content can be represented at different degrees of formality. Domain experts need to create, review and modify models at a rather informal/human intelligible level. Knowledge engineers need to check the quality of the formal definitions and their correspondence with the informal parts they intend to represent. In order not to increase the overhead of human work, translation between different levels of formality must be as automated as possible. To support a coherent development and integration of the different components of the enterprise model, such a modelling tool must support the modelling of all the relevant aspects of an enterprise in a collaborative, cooperative and integrated manner. This is in order to exploit the synergy of "having to think the same thing out only once".
MoKi (Modelling WiKi) is developed in order to meet this vision:

1. It supports access to the enterprise model at different levels of formality (informal, semi-formal, and formal);
2. It integrates modelling of several aspects of an enterprise; and
3. It ensures a coherent development of the formal part.

2 Conceptual framework

The key modelling aspects that MoKi aims to support are collaboration and integration. This section goes into detail about how we understand these terms and why they are relevant in the context of enterprise modelling.

2.1 Collaboration

Developing an enterprise model is inherently a collaborative activity, since a variety of skills are required which are unlikely to be found in a single person, as has already been argued above. In practice, different actors have very different expertise in encoding content into formal languages, or may know only of specific aspects of an enterprise. At this point it is necessary to understand that as a direct consequence different contributors, as members of a *modelling team*, also have different requirements on the modelling environment, especially with regard to the presentation of the models' content. The primary goal of our research with respect to collaboration is to derive requirements on a modeling environment by actors with different background knowledge and to develop appropriate ways to access models accordingly.

To support collaboration between the modelling team, and to allow great flexibility in the cooperative modelling activity, we therefore adopt a *collaborative modelling* paradigm, illustrated in Figure 1. This paradigm is inspired by recent Web 2.0 collaborative solutions, of which wikis are one example, and was already proposed in [3, 4] as a way to support modelling activities. In this paradigm all the actors asynchronously collaborate toward the creation of an

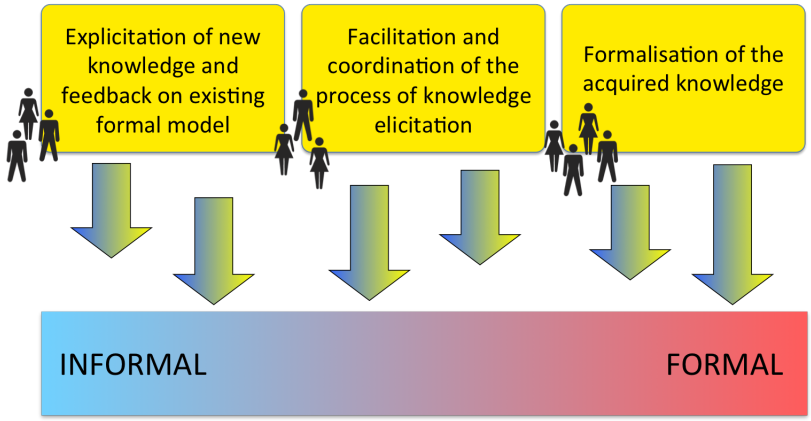


Fig. 1. Collaborative Modelling

integrated enterprise model by inserting knowledge (either formal or informal), by transforming knowledge (from informal to formal) and by revising knowledge. The domain experts enter the missing knowledge - using a form of informal language - into the models or provide feedback on the formal models created. The system semi-automatically translates part of the informal knowledge into a formal specification and vice-versa. Asynchronously, the knowledge engineers can refine the formal model by inserting new elements, by modifying existing knowledge or by asking clarifications to the domain experts. The usage of a robust collaborative technology, as the one provided by the wiki, allows the provision of state of the art functionality like simultaneous access and online communication via the platform.

Another important characteristic of our approach lies in the capability of the system to maintain the alignment between the informal specification of the enterprise model and its formal version. This can provide an added extra value, as the documentation contained in the informal part is often critical to fully understand its formal version. Traditionally, the main goal of enterprise modelling is the production of an integrated formal model in which the different aspects of an enterprise are integrated in a unique model. This integrated formal model is an artefact that nevertheless requires a strong connection with its informal part in order to be fully exploited both by humans and machines. Thus, to support the exploitation of an enterprise model also by humans we adopt a structure (also referred as *the meta-model*) which not only contains the formal meta-model of the enterprise, but also the informal versions of this knowledge.

2.2 Integration

Relevant to our idea of modelling different aspects of one enterprise is that the various models are interconnected, and thus constitute an *integrated* model.

In the current implementation of MoKi, we focus on an enterprise model describing the domain, the processes and the competencies of an enterprise; Figure 2 shows the current version of the integrated enterprise meta-model considered. The choice of these aspects, which constitute typical parts of an enterprise model,

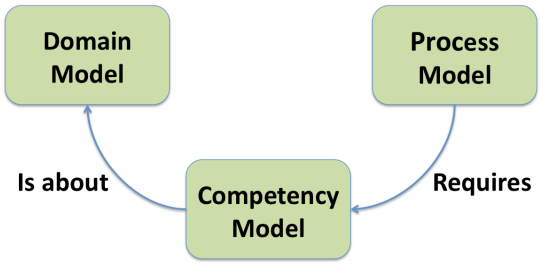


Fig. 2. The current version of the integrated enterprise meta-model considered in MoKi

was originally motivated by the EU-project APOSDLE[4] in which MoKi was first developed and used. Nonetheless, MoKi has also been used in different contexts already (see Section 4). Also, more complex enterprise models can be considered, and we have designed our approach with the explicit intent to be open and extendable to other aspects of an enterprise.

Below, we specify what we mean by domain, process and competency model, and illustrate how we see integrated modelling using these spcific aspects.
The *domain model* provides the description of the business domain within which the enterprise operates. It is a conceptualisation of the entities and the relations between them, which are relevant to the activities of an enterprise. This description is provided in terms of concepts, relations and objects. Following the growing popularity of Semantic Web technologies, we decided to base our representation of a domain-specific model upon the OWL ontology language[5]. This approach allows one to express classes, properties, instances, and axioms among them.
The *process model* provides a description of the patterns and procedures occurring in a business domain of an organisation. The very core of a process model is a control flow. In the e-learning application scenario described in [3] it was enough to consider a task to be either atomic or composed of a bag of subtasks, regardless of any execution control. In this case a simple hierarchical structure representing the task/sub-task relation was sufficient and we adopted an OWL ontology that encodes the part-of relation. In a different situation where tasks were complex structures described in the BPMN[6] language [5], a more complex model was adopted, in which processes are described by means of the primitives

[4] See www.aposdle.org.
[5] www.w3.org/TR/owlfeatures/
[6] Business Process Modelling Notation `www.bpmn.org`

defined in an OWL ontology that represents BPMN[7].
The *competency model* describes the attitudes and the capability of people employed in an organisation to fulfill their tasks and to reach their objectives and goals. Elements of the competency model are competencies, which express knowledge about domain concepts. Tasks are related to competencies, in that a competency may be required to perform a task, and vice versa the (successful) execution of a task indicates that person possesses a certain competency. Such a competency model allows describing users in terms of knowledge about concepts of the business domain and skills to perform the tasks of the process model. Clearly, such a competency model serves as connection between domain and process model. Practically, this connection is established by assigning tasks to competencies (domain model element plus skill type) which are required for performing the task. In the e-learning application scenario described in [3], the competency model was built focusing on the support of individual learning in the process of working tasks [6].

3 Enterprise modelling using MoKi

MoKi is based on Semantic MediaWiki (SMW) [7], extending it to offer particular support for enterprise modelling. Based on a predefined meta-model as the one described above, MoKi adds to SMW the following groups of functionalities: (i) import functionalities to load existing models from various formats, (iii) modeling functionalities for model management and representation (iii) export functionalities to translate models developed within MoKi into standard formats. These functionalities are described in more details throughout this section.

The choice of developing MoKi on top of a semantic wiki was made for several reasons. Wikis provide a state of the art robust collaborative tool, which enabled us to focus on the aspect of collaboration between actors of different skills and still getting an environment with more broad collaboration support. Due to the growing popularity of wiki-based web sites (e.g. wikipedia), users are quite familiar with wikis and the editing of wiki pages. Furthermore, the SMW framework already provides many important functionalities such as access control and permissions, tracing of the activity, semantic search, and so on, without the need to install specific client applications. Finally, only a web-browser is required on the end user side to use the system. The second important reason for choosing a semantic wiki was the fact that the wiki can provide a uniform tool and interface for the (informal) specification of the different components of an enterprise model (domain, processes, and competencies in our case). This is in opposition to the usual procedure, where dedicated but often disconnected, modelling tools are used to model each aspect. The usage of a uniform tool for the integrated modelling of different aspects of an enterprise provides a great opportunity to make modelling easier for domain experts. It is also a prerequisite for modelling different aspects of an enterprise in a truly integrated way, as described above. As a final reason for implementing MoKi on top of a semantic

[7] `http://dkm.fbk.eu/index.php/BPMN_Ontology`

wiki, the natural language descriptions inserted in a semantic wiki can be structured according to predefined templates, with the help of semantic constructs like properties. As a consequence, the informal descriptions in natural language contain enough structure to be automatically translated in formal models, thus allowing the reuse of informal descriptions for automatic ontology creation.

3.1 Describing knowledge in a MoKi page

MoKi integrates different views over portions of knowledge. The main idea behind MoKi is to associate a wiki page[8] to each (simple or complex) element of the formal models so that this page contains an informal but structured description of the element itself. The typical page contains[9]:

- An informal description of the element in natural language (images or drawings can be attached as well). The purpose of this part is to document the model and clarify it to users not trained in the formal representation (e.g., reference to source documents, notes about modelling choices and open problems, etc.). Comments can be added by each user and are not translated to the formal model;
- A structured part, where the element is described by means of triples of the form *(subject, relation, object)*, with the element itself playing the role of the subject. The purpose of this part is to represent the connection between elements of the same model (like class/sub-class relation between elements of the domain model, or task/sub-task relation between elements of the process model) as well as connections between elements of different models (like a relation denoting required knowledge between elements of the process and the domain model).

This natural language based, but also structured, description provides a natural bridge between formal and informal representation of knowledge. The user fills a page via forms (see the Semantic Forms extension[10]), so he/she does not need to know any particular syntax or language to participate in the creation of the enterprise model. All the actors involved in the modelling activities can also interact with each others and exchange further ideas and comments using the *discussion* SMW's built-it functionality. An example of a MoKi page describing an element of the domain model is shown in Figure 3 while an example of a MoKi page describing an element (task) of the process model is shown in Figure 4.

[8] Wiki categories could have been used as well to represents the concepts of the domain model. However, when we started developing the tool, the support for categories in SMW was rather preliminary, so we decided to represent domain concepts using standard pages.

[9] Note that in this section we use the term "model element" to indicate a basic component of the model. For instance, a concept or a relation of the domain model is a model element, a task of the process model, a competency, and so on.

[10] http://www.mediawiki.org/wiki/Extension:Semantic_Forms

The important point to stress here is the usage of semantic forms to realise appropriate templates to guide domain experts in providing their informal, but structured descriptions. Templates are the key to customise MoKi for modelling different kinds of model elements (e.g. domain concept, task, competency etc.) with respect to which knowledge shall be specified about the kind of element.

3.2 MoKi functionalities

MoKi provides several groups of functionalities to support modelling, all of which can be accessed via a wiki's style menu. This section contains a description of the functionalities currently available[11]. Concerning future extensions, MoKi is built in a modular way in order to facilitate the plugging-in of new or existing state-of-the-art tools.

Import Functionalities. We provide three types of import functionalities:

- *Import of available domain/task formal models.* With this functionality the user can set up MoKi with an already available domain or task model instead of starting modelling from scratch. From the technical point of view, the XML serialisation of the OWL formal model is parsed in order to obtain its relevant elements, and a page is created for each one of them. All pages are collected in a XML file, which then is given as input to the `Import pages` functionality available in SMW.
- *Input of structured lists of elements.* With this functionality the user can create new elements of the models by inserting lists of concepts (resp. tasks), organized according to predefined semantic structures, e.g. a taxonomy or a partonomy (resp. task/subtask decomposition structure). Figure 5 shows the loading of a list of concepts organized according to a partonomy in the domain model. Also this functionality takes advantage of the `Import pages` functionality available in SMW.
- *Text analysis functionalities.* To support the utilization of available unstructured knowledge relevant for the modelling activity, MoKi includes an extension which allows to extract relevant terms from digital resources, and to cluster such terms according to their relatedness. These functionalities are provided by the KnowMiner, an advanced text analysis tool developed by the Know-Center. The corresponding extension works in analogy to the extensions realised for Protégé in earlier work [8].

Model Management Functionalities. This set of functionalities provides the basic functionality each modelling tool necessarily provides: Creating, editing and deleting model elements. Depending on the type of element, pre-defined templates are loaded when it is created or edited. Such templates contain for

[11] A demo version of MoKi can be tried out on-line at the MoKi web site: moki.fbk.eu. A detailed description of the current version of MoKi is contained in the MoKi manual, available at the same web site.

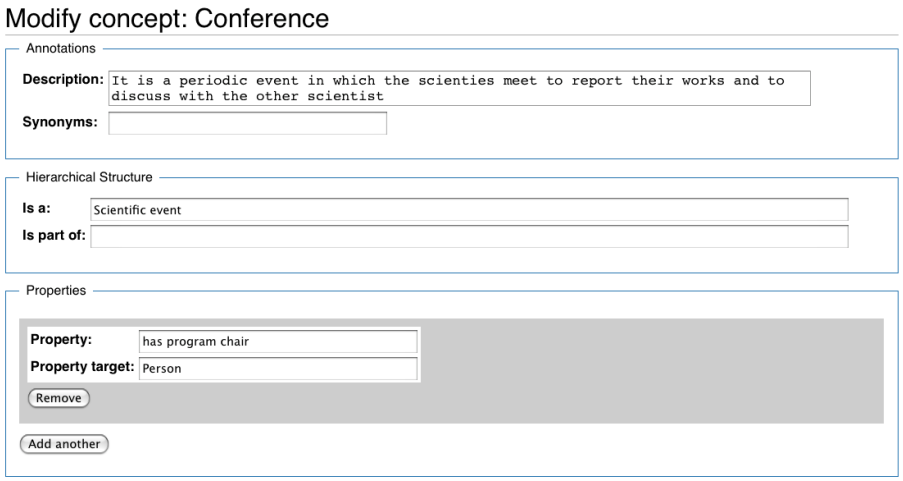


Fig. 3. An example of a MoKi page for a concept.

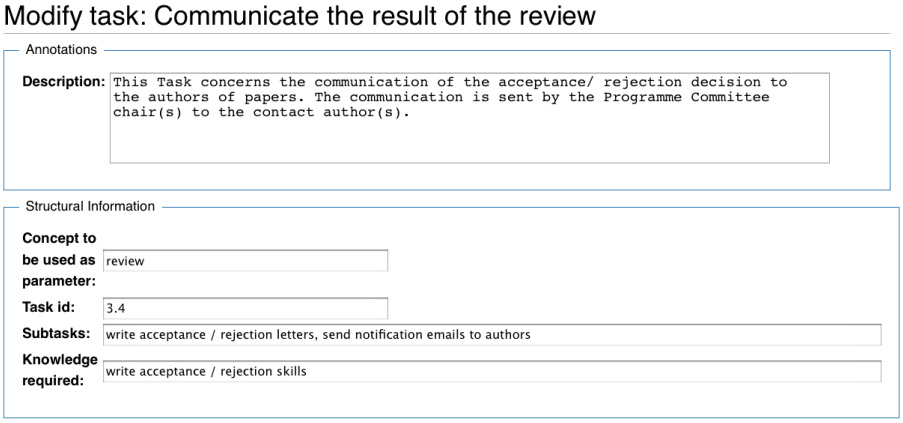


Fig. 4. An example of a MoKi page for a task

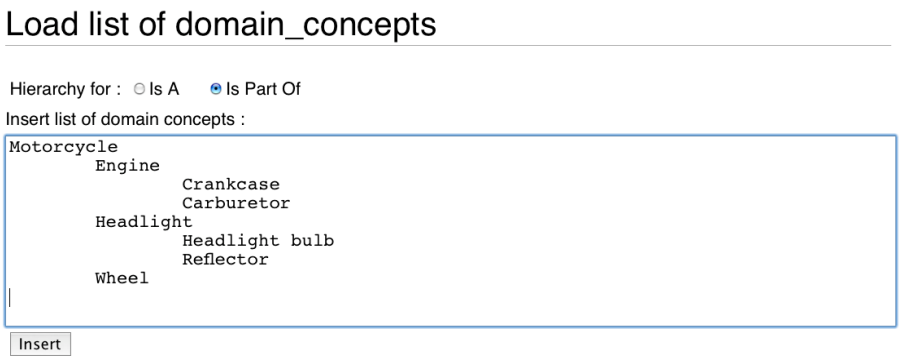


Fig. 5. Adding a list of concepts organised according to a part of hierarchy, via the `Load list of concepts` functionality.

instance properties for specifying a taxonomy or partonomy, or a sequence in the case of tasks.

Visualization Functionalities. These functionalities allow to produce different types of graphical overviews of the models: they help the actors to deal with the global picture on the models and not only with the single model elements. In particular, the tool allows two kinds of overviews of the model, a tabular-based one and a graphical-based one.

In the *tabular-based view*, the user sees a table listing all the elements of the domain model or the process model, where for each element some relevant information is shown, e.g. its description, the concepts of which it is a specialisation (for domain elements), its subtasks (for tasks), and more. A short extract of a list of element in a domain model is shown in Figure 6. This functionality is based on the `ask` query mechanism available in SMW.

In the *tree-based view*, called *IsA/PartOf Browser*, a tree-like view shows the hierarchy of the domain elements according to either the subclass or part of relation. This tree-like view, which can be see in Figure 7, is dynamically created from the content of the MoKi pages. The user has the possibility to expand/collapse only parts of the tree, thus allowing him or her to efficiently browse even large and complex models. Actually, this is not just a visualization, since the user can easily rearrange via drag 'n' drop the taxonomy and partonomy of concepts in the domain model, and the changes performed within the browser are propagated to the pages describing the elements involved. This functionality is an adaptation of the DHTMLx-Tree library [12], originally not meant for this purpose.

[12] http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml

List domain concepts

Number of concepts in the Domain Model: 35

Concept	Description	Is a	Is part of
Carburetor			Engine
Chi-Square-Distribution	The chi-square distribution is the distribution of squared values of a standard normal distributed random variable.	Probability Distribution	
Comparing Means	The term comparing means refers to a group of statistical tests for identifying statistically siginficant differences between means from different groups.		
Covariate	A covariate (also covariable) is a variate occurring concomitantly with the variate of primary interest and measured for the purpose of making informed adjustments on the variate of primary interest. It is a correlational variable (usually a characteristic of the subject) included in an experiment to help reduce the error variance in statistical tests.	Variable	
Crankcase			Engine
Dependent Variable	The dependent variable (criterion variable) is the variable that is observed to change in response to the independent variables. The dependent variable reflects the impact of changes in the independent variable. This means, the values of the dependent variable are depending on variations of values of the independent variable.	Variable	
Descriptive Statistics	Descriptive statistics are used to describe the basic features of the data gathered from an experimental study in various ways. The core of descriptive statistics are measures of central tendency (e.g. mean, median, modal value), and measures of dispersion (e.g. range, variance).		
Effect	The term "effect" is used as an umbrella term for the different effects that can occur as a result of a two-way ANOVA (main effect, interaction effect).		
Engine			Motorcycle
Experiment	An experiment is a specific type of method used in scientific inquiries, and personal questioning, to find an answer to a question. The essence of an experiment is to introduce a change in a system (the independent variable) and to study the effect of this change (the dependent variable).		

Fig. 6. Extract of a tabular-based view of the domain model.

Export Functionalities. These functionalities support the automatic export of knowledge of the enterprise model into standard knowledge representation languages. At the moment, the formal representation of all parts of the enterprise model is an OWL ontology. On-going work is devoted to the addition of other formal languages especially for task/process specification. The process model and the domain model can be exported separately. Technically speaking, the starting point to the automatically created the OWL ontology from the informal domain model is the built-in Semantic MediaWiki `Export pages to RDF` functionality. Using this functionality, it is possible to generate a document in OWL/RDF format containing information on the properties used in the pages describing the model. However, since this functionality has been developed independently with respect to the use of the Semantic MediaWiki that we propose, an automated postprocessing of this file is necessary in order to be able to generate an OWL ontology consistent with the informal model designed. For example, a page describing a domain concept is mapped by the `Export pages to RDF` functionality to an instance of a top class `smw:Thing`, while in our approach it should be mapped to an OWL class. Similarly, the “is a” relation is mapped by the `Export pages to RDF` functionality to an object property named `is a`, while in our approach this relation needs to be mapped to the RDFS `subClassOf`

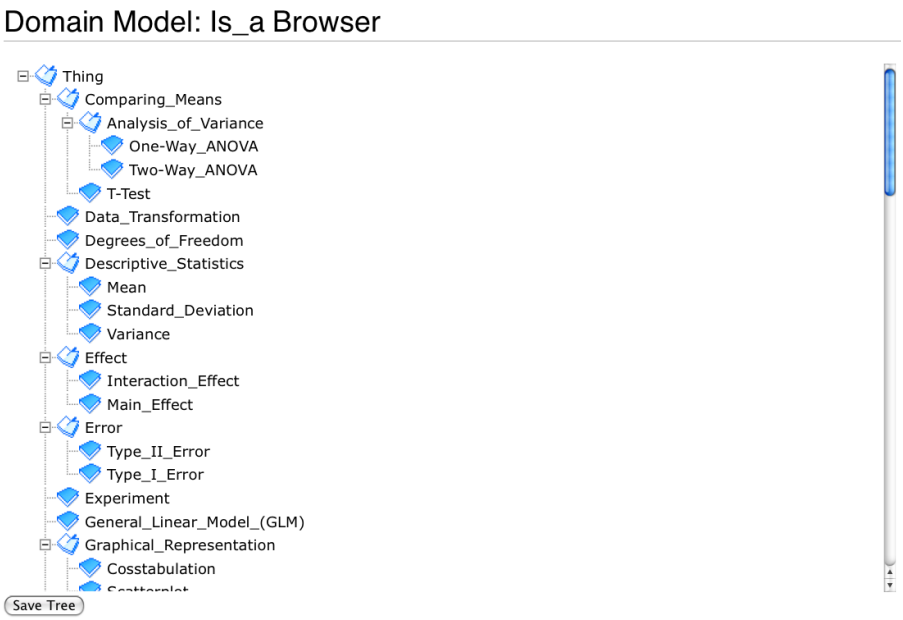


Fig. 7. The taxonomy of the concepts in a domain model shown via the IsA browser. Note the `Save Tree` button, which allows to save the class/subclass hierarchy after changes made via drag 'n' drop.

relation.

Reviewing MoKi against the claims made in the beginning of the paper, it:

1. Supports access to the enterprise model at different levels of formality (informal, semi-formal and formal) in that it (i) accommodates highly informal modelling based merely on hyperlink connected wiki pages as well as (ii) semi-formal modelling where pages and links are raised to a semantic level, and (ii) enables formal modelling by an easily accessible translation into formal models via an export functionality.
2. Supports integrated modelling of domain, processes and competences within an enterprise by providing one homogeneous interface for modelling all relevant aspects of an enterprise, and enabling knowledge engineers to interconnect models describing these aspects in a quite natural way.
3. Ensures a coherent development of the formal part by providing an import functionality which allows a re-translation of formal models into MoKi.

4 Use Cases and User Study

The MoKi has been successfully applied within the EU-project APOSDLE to develop enterprise models in six different domains: Information and Consulting

on Industrial Property Rights, Electromagnetism Simulation, Innovation and Knowledge Management, Requirements Engineering (the RESCUE methodology), Statistical Data Analysis and Information Technology Infrastructure Library. Some of the experiences of an early usage of the system are described in [3]. In addition, MoKi is used in applications that go beyond typical enterprise modelling: the representation of medical guidelines encoded in the ASBRU language[13], and the collection of data for the Personal Health Record of the Province of Trento, Italy[14]. The work done in these projects, as well as the analysis of the usage of MoKi in APOSDLE constitutes an important step towards the improvement of the tool and realisation of the full framework.

User study A qualitative evaluation based on the usage of the MoKi between September 2008 and January 2009 by four application partners modeling five different enterprise domains in the scope of the APOSDLE project was carried out. The evaluation took the form of structured interviews with both open and closed questions. Interview questions targeted not only MoKi but the whole modeling process implemented in APOSDLE [10]. Modeling activities in APOSDLE involved domain experts, on-site knowledge engineers as well as external knowledge engineers. the interviews were carried out with the on-site and external knowledge engineers but not directly with the domain experts.
All participants reported a positive experience of MoKi. In particular, the import (easy integration of previously available knowledge) and export functionalities (translation into formal models) were highly appreciated. Also, the homogenous modeling environment for modeling different aspects (domain, task, preliminary competency model) was found to facilitate the process. Furthermore, the participants reported that MoKi did facilitate collaboration among the modelling team.

5 Related Work

Solutions to the problem of modelling various aspects of an enterprise were proposed in several works, both in terms of definition of the meta-model and in terms of methodologies to support the creation of the model itself: a detailed comparison between state of the art approaches and the one proposed in this paper can be found in [4].

Many tools are available to support the creation of formal models in general. Most of them, e.g. Protégé [11], were born as standalone desktop applications. Despite the development of pug-ins that support collaborative features (e.g., Collaborative Protégé) the tools remain barely usable by users with limited expertise of formal languages. The MoKi does not directly compare to such tools, since it is not a modelling tool for a specific formalism but rather for specific kinds of entities (concepts, tasks etc.). The support for concrete formalisms lies in the implementation of different export functionalities. Additionally of course,

[13] Part of the OncoCure project. See [9].
[14] Part of the TreC project. See trec.fbk.eu

MoKi aims to collect information about these entities at different levels of formality. Recently, wiki systems, and semantic wikis, have been applied to support collaborative knowledge creation and sharing. We mention a few of them, and assume for all that they offer "traditional" wiki functionality, i.e. web-based, easy text edition and linking to web resources, integration of multimedia content and versioning.
There is already at least one proposal in which the modelling of processes is done using the pure Semantic MediaWiki, see Dengler et al [12]. Semantic MediaWiki+ (SMW+) [13] is a further extension on Semantic MediaWiki with a focus on enhanced usability for semantic features. Especially, it supports besides the annotation of whole pages also the annotation of parts of text and offers additional funtionalities termed "knowledge gardening" functionalities. The latter are maintenance scripts at the semantic level, with the aim to detect inconsistent annotations, near-duplicate entries etc.
IkeWiki [14] and OntoWiki [15] are two more semantic wikis, both however are completely independent from pre-existing wiki systems. Java-based IkiWiki supports the semantic annotation of pages and links between pages with semantic. Annotations are used for context-specific presentation of pages, advanced querying, consistency verification or drawing conclusions. IkeWiki also directly supports reasoning on its knowledge base. Continued development of IkeWiki now takes place within the EU-project KIWI [16]. OntoWiki seems to focus slightly more directly on the creation of a semantic knowledge base, and offers widgets to edit/author not only single elements/pages but also whole statements (subject, predicate, object).
AceWiki [17] was developed in the context of logic verbalisation, and is based on research to verbalise formal logic statements, and inversely translate backwards English statements into formal logic. AceWiki is based on Attempto Controlled English - ACE, which allows users expressing their knowledge in near natural language (i.e. natural language with some restrictions). Note that although such content may look like natural language, in contrast to the informal fields in MoKi for instance, it is actually formalised, i.e. follows some rules. In contrast to this, the content of the informal parts in MoKi, e.g. the descriptions of the model elements, is completely unrestricted.
myOntology [18] is geared towards the collaborative and community-driven development and maintenance of lightweight ontologies. In particularit has been applied within the context of E-Commerce.

What MoKi offers in addition are two main contributions:

- The support for the integrated specification of multiple aspects (in the use cases described above, this meant domain, process and competencies).
- The bi-directional transformation between formal and informal models.

6 Conclusions

In this paper we have presented MoKi, a new tool for collaborative enterprise modelling. The general framework and the tool we envisage constitute a gen-

uine contribution towards supporting a fruitful collaboration among people with different skills and levels of expertise in the modelling activities. The current implementation of MoKi, developed inside the APOSDLE EU-project already provides key functionalities towards the modelling of an integrated enterprise model in a collaborative manner, and constitutes a first version towards the realisation of the full framework.

Future work focus on improving the tool and make it more general. Examples of future work include: a better support for domain and process modelling, including better support to the modelling of all the elements of the formal models; the integration of the competency model in the MoKi[15]; better support to define templates and to adapt to different meta-models; support for validation of knowledge in the MoKi by means of the domain experts.

Acknowledgements We thank all the people involved in the modelling activities of the APOSDLE project for their useful suggestions and feedback. This work has been partially funded under grant 027023 in the IST work programme of the European Community (APOSDLE IST-project). The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG

References

1. Fox, M.S., Grüninger, M.: Enterprise modeling. AI Magazine **19**(3) (1998) 109–121
2. Tudorache, T., Noy, N.F., Musen, M.A.: Collaborative protege: Enabling community-based authoring of ontologies. In: International Semantic Web Conference (Posters & Demos). (2008)
3. Christl, C., Ghidini, C., Guss, J., Pammer, V., Rospocher, M., Lindstaedt, S., Scheir, P., Serafini, L.: Deploying semantic web technologies for work integrated learning in industry. a comparison: Sme vs. large sized company. In: Proceedings of the 7th Int. Semantic Web Conference (ISWC 2008), In Use Track. Volume 5318., Springer (2008) 709–722
4. Rospocher, M., Ghidini, C., Serafini, L., Kump, B., Pammer, V., Lindstaedt, S.N., Faatz, A., Ley, T.: Collaborative enterprise integrated modelling. In Gangemi, A., Keizer, J., Presutti, V., Stoermer, H., eds.: SWAP. Volume 426 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
5. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P.: Reasoning on semantically annotated processes. In: Proceedings of the 6th International Conference on Service Oriented Computing (ICSOC'08), Sydney, Australia (2008)

[15] The creation of the competency model was initially envisaged and performed in APOSDLE via the TAsk-Competency Tool (TACT) [10], developed by the Know Center outside the MoKi. On-going work is focused on incorporating and extending the functionalities of TACT in the MoKi to fully support the modelling of domain, processes and competencies in an integrated manner.

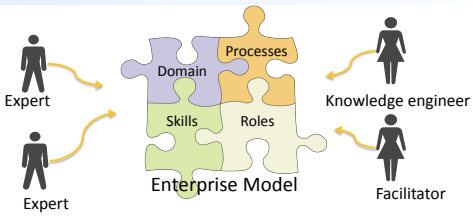
6. Lindstaedt, S.N., Ley, T., Scheir, P., Ulbrich, A.: Applying Scruffy Methods to Enable Work-integrated Learning. Upgrade (2008) in press
7. Krotzsch, M., Vrandecic, D., Volkel, M.: Wikipedia and the semantic web - the missing links. In: Proc. of the 1st Int. Wikimedia Conference (Wikimania 2005)
8. Pammer, V., Scheir, P., Lindstaedt, S.: Two protégé plug-ins for supporting document-based ontology engineering and ontological annotation at document-level. In: 10th International Protégé Conference, Budapest, Hungary, July 15-18, 2007
9. Eccher, C., Ferro, A., Seyfang, A., Rospocher, M., Miksch, S.: Modeling clinical protocols using semantic MediaWiki: the case of the Oncocure project. In: ECAI workshop on Knowledge Management for Healthcare Processes (K4HelP). (2008)
10. APOSDLE Deliverable 1.6: Integrated modelling methodology version 2 (forthcoming in April 2009)
11. Protégé: The protégé project (2000) http://protege.stanford.edu.
12. Dengler, F., Lamparter, S., Hefke, M., Abecker, A.: Collaborative process development using semantic mediawiki. In: Proceedings of the 5th Conference of Professional Knowledge Management. Solothurn, Switzerland, March 2009.
13. Semantic MediaWiki+: Business ready semantic collaboration (2008) http://wiki.ontoprise.de/ontoprisewiki/index.php/Main_Page.
14. Schaffert, S.: Ikewiki: A semantic wiki for collaborative knowledge management. In: 1st Int. Ws. on Semantic Technologies in Collaborative Applications (STICA'06)
15. Auer, S., Dietzold, S., Riechert, T.: Ontowiki - a tool for social, semantic collaboration. In: Proceedings of the 5th International Semantic Web Conference, Nov 5th-9th, Athens, GA, USA. Volume 4273., Springer (2006) 736–749
16. The KiWi Vision: Collaborative knowledge management, powered by the semantic web. (2008) Deliverable 8.5 - http://wiki.kiwi-project.eu/multimedia/kiwi-pub:KiWi_D8.5_final.pdf.
17. Kuhn, T.: AceWiki: A Natural and Expressive Semantic Wiki. In: Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges. (2008)
18. myOntology: Open ontology environment for semantic web-based e-commerce. (2008) http://www.myontology.org/.

MoKi: the Modelling Wiki

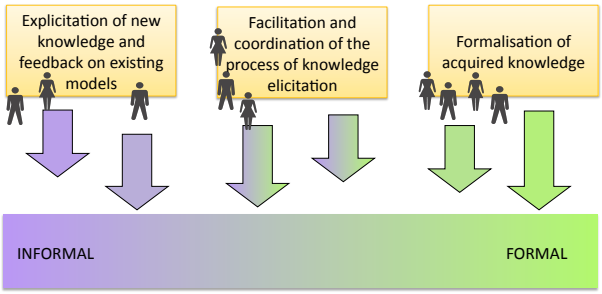
Marco Rospocher[1], Chiara Ghidini[1], Viktoria Pammer[2], Luciano Serafini[1], Stefanie Lindstaedt[3]

Enterprise modelling:

- Creates a structured description of different aspects of an enterprise (business domain, processes, goals, …) and their mutual relations
- Requires specific modelling skills and involves a team of modellers
- Is a truly collaborative activity carried out under some collaborative protocol



Our asynchronous collaborative approach:



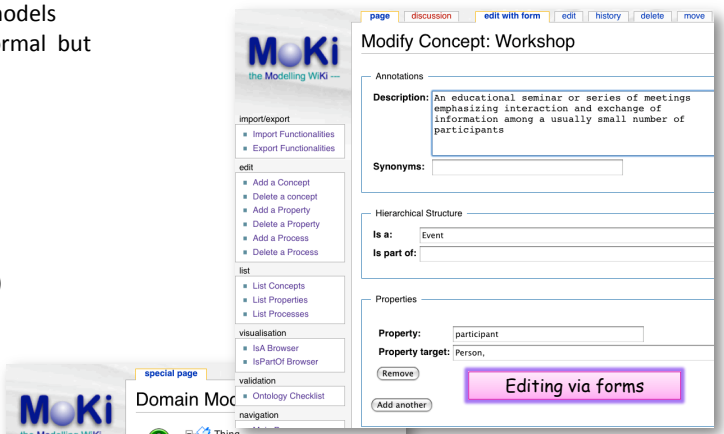
- Asynchronous collaboration of all the actors, who can:
  - insert knowledge
  - transform knowledge
  - revise knowledge
- Concurrent specification at different degrees of formality
- Automatic translation
  - from the informal specification to a formal model
  - and vice-versa

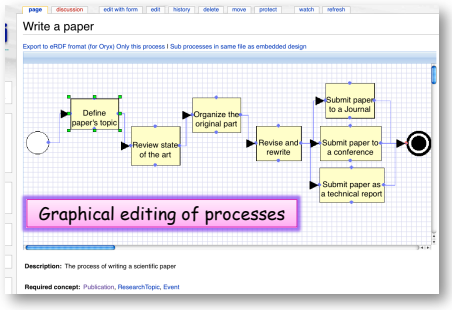
MoKi - the Modelling wiKi:

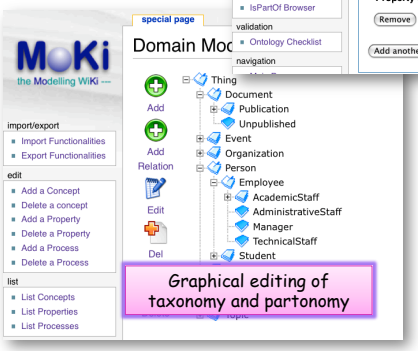
- Supports the construction of integrated domain and process models
- Each element of the formal model is described, in an informal but structured way, in a wiki page

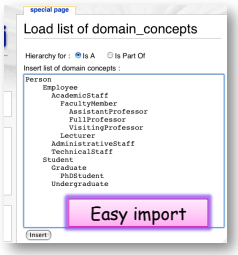
Current features:

- Easy editing of a wiki page by means of forms
- Different import functionalities:
  - Automatic import of OWL models
  - Easy import of lists of elements organized according to predefined semantic structures (taxonomy or partonomy)
  - Term extraction functionality
- Graphical browsing/editing of the domain and process models
- Hints for revision support
- Automatic export to OWL









Usage of MoKi:

- Six medium sized enterprise models in the EU-project APOSDLE (www.aposdle.org)
- Evaluation of MoKi ongoing at the *Joint European Summer School on Technology Enhanced Learning 2009*
- CLIP-MoKi: modeling of medical guidelines encoded in ASBRU
- BP-MoKi: modelling of semantically annotated business processes

moki.fbk.eu

[1]FBK-irst - Via Sommarive 18 Povo, 38050,Trento, Italy :: *surname*@fbk.eu
[2]Knowledge Management Institute, TU Graz - 8010 Graz, Austria :: viktoria.pammer@tugraz.at
[3]Know-Center - Inffeldgasse 21a, 8010 Graz, Austria :: slind@know-center.at

Brede Wiki: Neuroscience data structured in a wiki

Finn Årup Nielsen

Center for Integrated Molecular Brain Imaging, Copenhagen, Denmark;**
DTU Informatics, Technical University of Denmark, Lyngby, Denmark;
Neurobiology Research Unit, Copenhagen, Denmark
fn@imm.dtu.dk, http://www.imm.dtu.dk/~fn/

Abstract. Setup in January 2009 the *Brede Wiki* contains data from neuroscience, particularly from published neuroimaging peer-reviewed papers. Data is stored in simple MediaWiki templates and it can automatically be extracted and represented in an SQL format. Off-wiki Web-scripts can use the SQL database so items in the wiki can be queried efficiently, e.g., to find close brain activations to a given coordinate. Template content is non-nested and without wiki markup making extraction simple and complete.

1 Background

The complexity of neuroscience data has resulted in multiple neuroinformatics databases. One particular kind of database records brain activations foci from published peer-reviewed neuroimaging papers. The foci are reported with respect to a so-called stereotaxic space, such as the *Talairach space* [1], so coordinates are reasonably comparable across studies. One of the first neuroinformatics databases, *BrainMap*, records this kind of data and had its Web-presence in the early Web [2]. It continues in a modified version, and a few other coordinate databases have emerged: AMAT, SumsDB and our own Brede Database [3, 4]. Many of the neuroinformatics databases handle data entry manually. This is time consuming and may be the reason why databases are not complete. Our Brede Database uses Matlab for data entry and XML for storage. The scheme does not encourage collaborative and incremental data entry. We have constructed a plug-in that searches the database from the popular *SPM* program [5]. Extensions to the plug-in could potentially upload coordinates to the database, but that would need a submission interface on the server side.

Bioinformatics has embraced wiki technology, with e.g., the wikis SNPedia, WikiGenes, WikiProtein, WikiPathways.[1] Although some neuroinformatics wikis exist they are mostly text-oriented and used for documentation rather than

** Thanks to the Lundbeck Foundation for funding.

[1] The English Wikipedia page *Portal:Gene Wiki/Other Wikis* presently lists other bioinformatics wikis.

for recording data or describing its content more semantically. Exceptions are NeuroLex and NeuroCommons.

We have gained some experience in processing the templates of Wikipedia (more specifically the *journal* field of the *cite journal* template for scientific citations) and for submitting the processed templates to statistical analyses [6, 7]. The DBpedia effort [8] has shown the possibility for large-scale databasing with data extracted from Wikipedia.

Here I describe a system, the *Brede Wiki* (http://neuro.imm.dtu.dk/wiki/), that use MediaWiki and its template functionality to record neuroinformatics data. Associated scripts can extract the template content from the entire XML dump of the wiki and automatically construct SQL representation of the data. The SQL database enables more advanced queries than is possible with the present standard MediaWiki.

2 Methods

To avoid making data extraction unnecessarily complex and to match with the SQL language the application of MediaWiki templates has been kept simple: Present templates do not nest, have lower-case characters (except for first letter) and wiki markup is avoided within the template field values. The following is an example of an application of the `paper` template:

```
{{Paper
 | author1 = Daniela Balslev | author2 = Finn Nielsen
 | author3 = Olaf B. Paulson | author4 = Ian Law
 | title = Right Temporoparietal Cortex Activation during
           Visuo-proprioceptive Conflict
 | journal = Cerebral Cortex
 | volume = 15 | issue = 2 | pages = 166-169 | year = 2004
 | pmid = 15238438 | doi = 10.1093/cercor/bhh119 | wobib = 128
}}
```

In the present wiki one page may contain multiple templates, e.g., a page for a specific neuroimaging paper can contain the `paper` template as well as multiple templates for brain coordinates: the `Talairach coordinate` template. Most template definitions format the template content by construction of an infobox as known from Wikipedia. What is somewhat different from Wikipedia is the extensive use of wiki links within the template definitions. For example, for the `paper` template wiki links to the authors and journal is constructed, with MediaWiki template definitions like `[[{{{author1}}}]]`. External links are created from the external database identifiers, such as DOI and the PubMed Identifier (PMID). For the templates which usually come in sets, such as `Talairach coordinate`, each application of a template defines a row in a table.

An example of a page within the Brede Wiki is displayed in Figure 1 on page 4 with the `paper` template as the upper right infobox and with three formatted applications of the `Talairach coordinate` template at the bottom. The `x`, `y`

and `z` fields of the template can be combined in a query to external specialized coordinate search engines. These links are displayed in the right-most column.

As the templates are not nested relatively simple Perl regular expressions retrieve them with `m/{{(.*?)}}/sg` and subsequently extract the template name and its content with

```
m/([a-z][a-z0-9]*(?:[ _][a-z0-9]+)*)\s*(\|.*)?/si
```

Spaces are later substituted with underscores. Another regular expression in the same style extracts name-value pairs from each field. The extracted content is written to SQL tables, where the master table is presently defined for SQLite as

```
CREATE TABLE brede(id INTEGER, pid INTEGER, title, tid INTEGER, template,
  field, fid INTEGER, value);
```

`id` is the row identifier, `pid` an identifier for the wiki page which title is also (redundantly) represented in the `title` column. `tid` and `template` are identifier and name of the template, while `field` is the field name, `fid` the field number and `value` the value, i.e., the actual content of the field. An insert with the `author2` field from the above displayed `paper` template example may be issued as:

```
INSERT INTO brede VALUES(387, 31, 'Right temporoparietal cortex
  activation during visuo-proprioceptive conflict',
  136, 'paper', 'author', 2, 'Finn Nielsen');
```

Apart from the master table several other tables are built: One for each template. For the `paper` template the SQL definition for the corresponding table may look like the following:

```
CREATE TABLE brede_paper(__tid, __pid, __title, _author1, _author2,
  _author3, _author4, _title, _journal, _volume, _issue, _pages, _year,
  _pmid, _doi, _wobib);
```

The final number of columns will depend on the number of different field names discovered during reading of the XML dump. The table names are prefixed with `brede_` and the column names with an underscore to avoid clashes between Brede Wiki names and SQL reserved words. A specialized search engine use the constructed SQL database when searching for nearby Talairach coordinates to a query coordinate [9].

At the time of writing 43 papers were represented in the Brede Wiki, 31 which potentially contain Talairach coordinates. In comparison the Brede and BrainMap databases have presently 186 and 1711 papers, respectively. Apart from templates for papers and Talairach coordinates, the Brede Wiki has also templates for, e.g., brain regions, researchers, subject groups and brain volume results.

3 Discussion

There are both advantages and disadvantages with the Brede Wiki compared to our previous system. Some of the advantages are online versioning, immediate

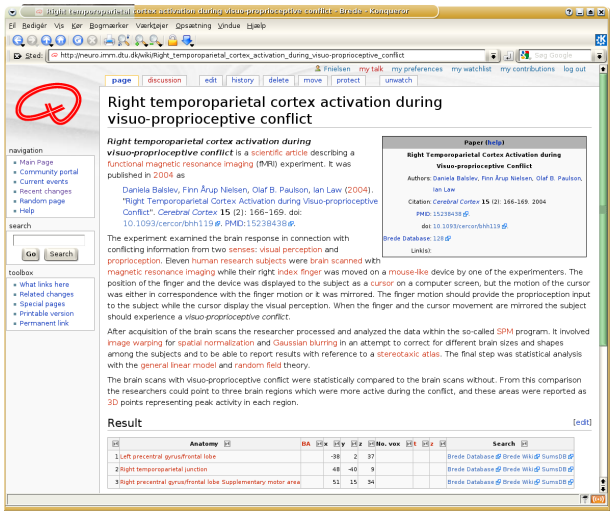


Fig. 1. Screenshot of Brede Wiki with a page about a scientific article.

access to entered data, incremental addition of data, the possibility for free format text descriptions as well as discussion pages. Furthermore the data structure is extensible, i.e., it is relatively easy for editors to add new 'columns' (fields).

Among the disadvantages are the issue of vandalism, quality control and database consistency. A paper may have multiple sets of coordinates that arise from different brain scanners, multiple subject groups and multiple experiments. If these components are described on a single wiki page the connection between them will need to be indicated with keys of some kind, e.g., to say that the fourth `Talairach coordinate` resulted from analysis of brain scans from the second `Subject group`.

The wiki does not solve the data entry problem *per se*. The data in the Brede Wiki has so far for the most part been entered manually in the raw wiki text. A small Matlab script can convert results from SPM so they appear in the Brede Wiki template format. Our small fielded wiki with form entry for personality genetics association studies [10] can also output its data for inclusion in the Brede Wiki. Entry with forms within the wiki would be a natural next step as well as possibly the adding of Semantic MediaWiki functionality [11]. Yet another option for data entry is scripts that automatically setup wiki pages from information in other databases, — an approach taking for *Gene Wiki* in Wikipedia [12]. Using this scheme it should be possible to augment the Brede Wiki with information from the Brede Database.

The present consensus on notability in Wikipedia restricts the information that a Wikipedian can enter: If it is not notable enough another Wikipedian might delete it. Ordinary researchers and individual research articles do not have sufficient notability such that individual pages can be constructed for each of them. When DPpedia relies on Wikipedia then this combination does not suffice for our purpose.

Compared to true semantic wikis the Brede Wiki cannot perform 'in-wiki' semantic queries. However, an off-wiki Web-script using the SQLite database can search for Talairach foci extracted from the `Talairach coordinate` templates, and the wiki and the Web-script link between each others. In the wiki such

links are automatically constructed by the template. Similar Web-scripts may be setup that utilize other parts of the extracted data in more complex ways.

In our small fielded wiki of personality genetics we can perform on-the-fly meta-analysis and data plotting. The vision is that the Brede Wiki can constitute the basis for large-scale Web-based meta-analyses similar to that of the (non-wiki) AlzGene database [13].

4 Conclusion

The Brede Wiki is one of the first steps in neuroinformatics with Web 2.0 and with a high degree of structured content. It shows the possibility to output structured MediaWiki content to an SQL database.

References

1. Talairach, J., Tournoux, P.: Co-planar Stereotaxic Atlas of the Human Brain. Thieme Medical Publisher Inc, New York (January 1988)
2. Fox, P.T., Lancaster, J.L.: Neuroscience on the net. Science **266**(5187) (November 1994) 994–996
3. Derrfuss, J., Mar, R.A.: Lost in localization: The need for a universal coordinate database. NeuroImage, doi:10.1016/j.neuroimage.2009.01.053 (2009)
4. Nielsen, F.Å.: The Brede database: a small database for functional neuroimaging. NeuroImage **19**(2) (June 2003) Presented at the 9th International Conference on Functional Mapping of the Human Brain, June 19–22, 2003, New York, NY. Available on CD-Rom.
5. Wilkowski, B., Szewczyk, M., Rasmussen, P.M., Hansen, L.K., Nielsen, F.Å.: Coordinate-based meta-analytic search for the SPM neuroimaging pipeline. In: International Conference on Health Informatics (HEALTHINF 2009). (2009)
6. Nielsen, F.Å.: Scientific citations in *Wikipedia*. First Monday **12**(8) (August 2007)
7. Nielsen, F.Å.: Clustering of scientific citations in Wikipedia. In: Wikimania. (2008)
8. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. In: The Semantic Web. Volume 4825 of Lecture Notes in Computer Science., Heidelberg/Berlin, Springer (2008) 722–735
9. Szewczyk, M.M.: Databases for neuroscience. Master's thesis, Technical University of Denmark, Kongens Lyngby, Denmark (2008) IMM-MSC-2008-92.
10. Nielsen, F.Å.: A small wiki for personality genetics. 37th Annual Meeting on Biochemistry and Molecular Biology: Frontiers in Genomics (October 2008)
11. Krötzsch, M., Vrandečić, D., Völkel, M.: Semantic MediaWiki. In Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L., eds.: The Semantic Web - ISWC 2006. Volume 4273 of Lecture Notes in Computer Science., Berlin/Heidelberg, Springer (2006) 935–942
12. Huss, III, J.W., Orozco, C., Goodale, J., Chunlei, Batalov, S., Vickers, T.J., Valafar, F., Su, A.I.: A gene wiki for community annotation of gene function. PLoS Biology **6**(7) (July 2008) e175
13. Bertram, L., McQueen, M.B., Mullin, K., Blacker, D., Tanzi, R.E.: Systematic meta-analyses of Alzheimer disease genetic association studies: the AlzGene database. Nature Genetics **39**(1) (January 2007) 17–23

Brede Wiki: Neuroscience data structured in a wiki

Finn Årup Nielsen

The Lundbeck Foundation Center for Integrated Molecular Brain Imaging;
Department of Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark;
Neurobiology Research Unit, Rigshospitalet, Copenhagen, Denmark.

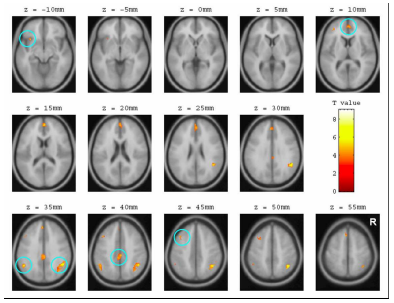
134

Summary

The Brede Wiki running on MediaWiki software represents data in templates. Data is from published peer-reviewed neuroscience articles. Further information is ontologies of, e.g., brain regions and brain functions. Since data in the templates is represented in a simple format all template data can be extracted and represented in SQL. From an SQL database specialized search can be performed.

Background

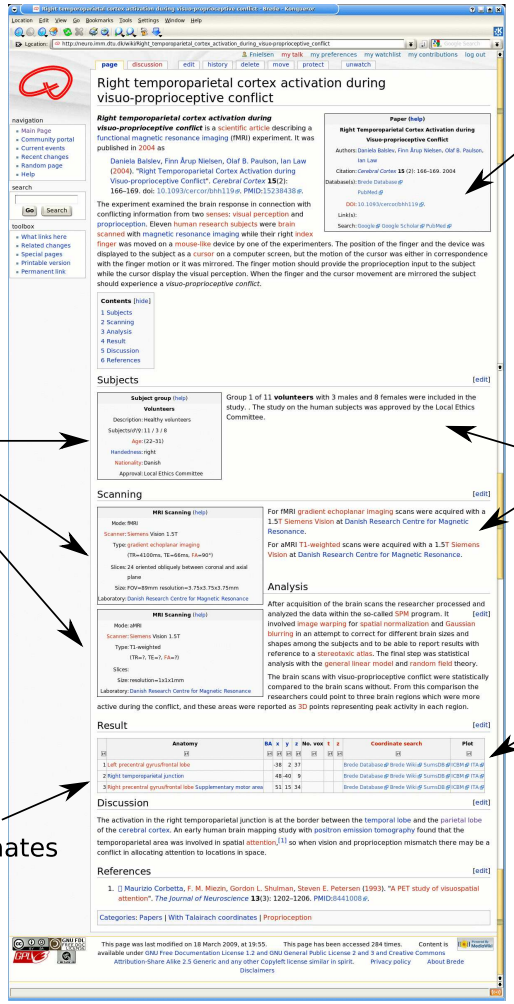
Neuroscience produces a wealth of data of different sorts. Neuroimaging uses positron emission tomography or magnetic resonance imaging brain scanners and may report results as 3D coordinates indicating foci of peak brain activation.



Example data from Lin et al. (2008)[1] CC-by.

Several databases exist for storing these data,[2] but contributing data is not straightforward. A structured wiki might be good for organizing this data.

Example page from the Brede Wiki



Brede Wiki templates

The templates used in the Brede Wiki may be grouped in four categories

1. Non-hierarchical

   Templates used to describe the concept on the wiki page. A wiki page may represent a researcher and a `Researcher` template is added on the top of the page with fields such as `name`. Other similar templates are `Paper` and `Journal`.

2. Hierarchical

   Templates used to describe the concept of the wiki page, which furthermore can be organized in a hierarchy — a simple ontology: `Brain region`, `Cognitive component`, `Organization`, `Software`.

3. On-page — single

   Besides the main template on a page multiple other templates are used in the Brede Wiki, e.g., to describe the methodology in a paper: `Subject group`, `Mri scanning`, `Pet scanning`, `Psychoexperimental condition`.

4. On-page — multiple

   Templates where multiple instantiations are formated into a single HTML table: `Talairach coordinate`, `Brain volume`, `Gene personality association`.

The template instantiations are **non-nested**, all use **lower case** field and template names and field values are **without wiki markup and links**. Links are constructed in the template definitions.

SQL of template data

SQL can be generated from the data in the templates. Since the template instantiations are simple the extraction of data from the wiki templates is complete, i.e., all template data can be extracted and represented in SQL.

Two kinds of SQL tables are constructed from the Brede Wiki template data:

1. One master table where the template field names are represented as SQL values.
2. Secondary tables where template field names are SQL table columns. Here is one table for each template definition, e.g., the {{Paper | ...}} template is associated with the SQL table called `brede_paper`

Example template in MediaWiki markup:

```
{{Paper | author1 = Tim | author2 =
Wendy | title = Semantic Web }}
```

Simplified master SQL table:

```
CREATE TABLE brede(title, template,
field, fid, value)
```

with simplified data for inserts:

```
("The Web", "paper", "author", 1, "Tim")
("The Web", "paper", "author", 2, "Wendy")
("The Web", "paper", "title", 1, "The Web")
```

Definition for secondary SQL tables can for example be

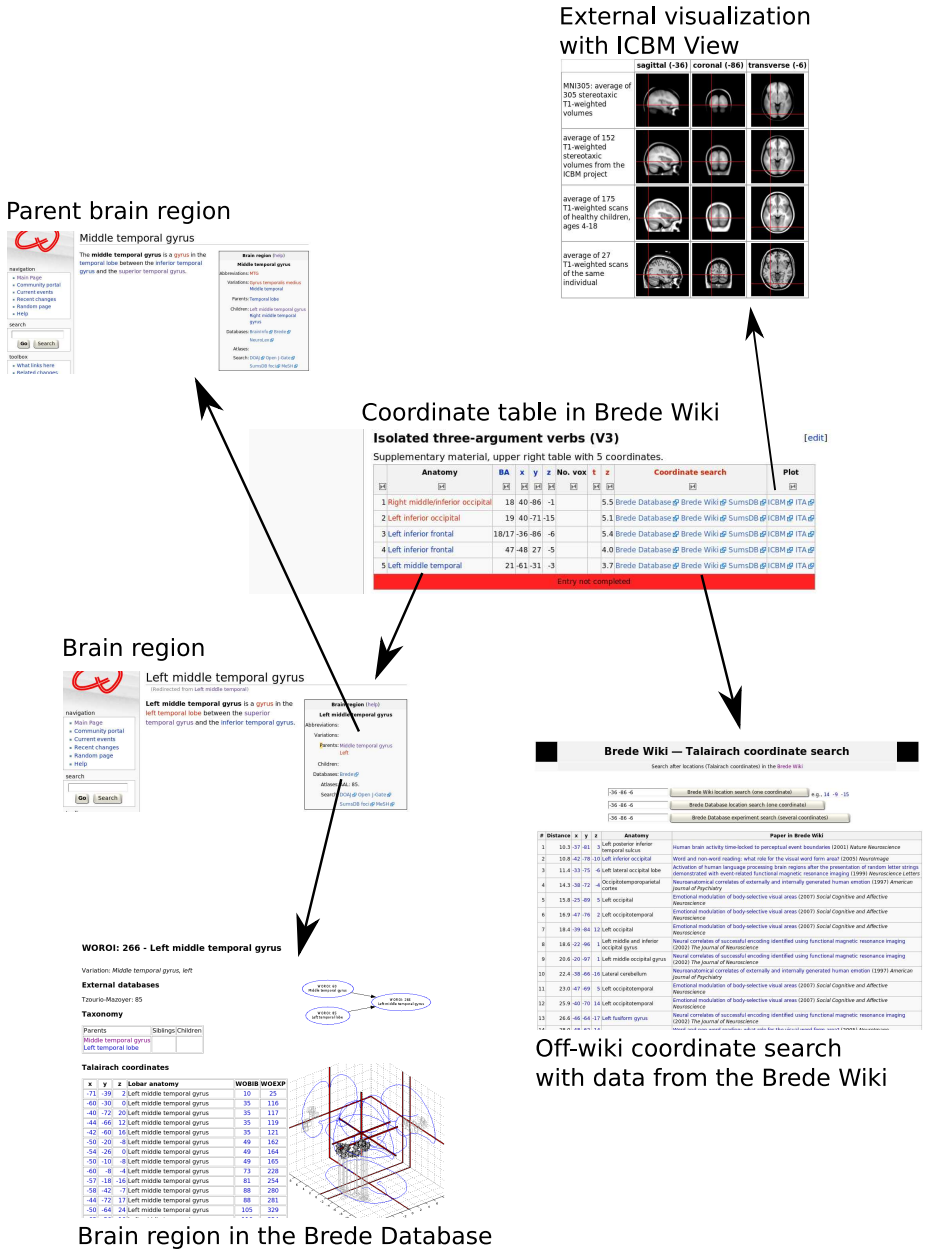
```
CREATE TABLE brede_paper(_title,
_author1, _author2, _title)
```

And simplified data for inserts:

```
("The Web", "Tim", "Wendy", "The Web")
```

With the SQL database complex queries can be made on all data encoded in the templates of the Brede Wiki. So far our only Web-service is for searching for nearby brain coordinates using SQLite.

Automated linking and querying for coordinate data



Further issues

Formating: Brede Wiki template coordinate data can be formated from a Matlab image analysis program. Another of our structured wikis can output its personality genetics data in template format ready for inclusion in the Brede Wiki. MediaWiki extensions might be of interest for in-wiki form input.

Download: MediaWiki dumps of the Brede Wiki as well as SQL and SQLite files are available from the Brede Wiki homepage. From the ontologies a SKOS[3] file is also produced.

Why not Wikipedia and DBpedia?: Much of the information in the Brede Wiki will not be notable enough, so Wikipedia administrators would delete the page. Furthermore, building our own wiki allow us to keep the templates simple so extraction can be complete, i.e., all data from the templates can be extracted.

Other database: The Brede Wiki links to, e.g., Brede Database, PubMed, NeuroLex wiki. So far it is not possible to automatically translate data from, e.g., the Brede Database to the Brede Wiki.

Acknowledgment

Thanks to the Lundbeck Foundation for funding and Lars Kai Hansen and Daniela Balslev for discussions.

References

[1] Lin CH, et al. Brain maps of Iowa gambling task. *BMC Neuroscience*, 2008; 9:72.
[2] Derrfuss J and Mar RA. Lost in localization: The need for a universal coordinate database. NeuroImage, doi:10.1016/j.neuroimage.2009.01.053, 2009.
[3] Miles A and Bechhofer S. SKOS Simple Knowledge Organization System Reference. W3c candidate recommendation, W3C, MIT, 2009.

Metasocial Wiki - Towards an interlinked knowledge in a decentralized social space

Amparo E. Cano[1], Matthew Rowe[2], Fabio Ciravegna[3]

Department of Computer Science,
University of Sheffield,
Sheffield, United Kingdom
A.Cano[1],m.rowe[2], F.Ciravegna[3]@dcs.shef.ac.uk

Abstract. This paper introduces a new approach to semantic wikis. In this approach users coming from different social networks can be merged into a common space to enable collaboration. This approach makes use of the user's identity representation and keeping track of the user's interests according to the type of annotations encountered in the content they add.
Keywords: semantic wiki, digital identity, collective intelligence, social networks.

1 Introduction

According to Metcalfe's law[1], the larger the network the more valuable it becomes. Although individual thinkers invent and discover, it is groups, which typically refine and extend innovations. Moreover, highly developed ideas rarely emerge from single and isolated thinkers, they usually come as a result of a process of interaction [2]. Even though semantic wikis have proven to be a successful tool for collaborative working, there is still a long way to go to fully exploit users collective intelligence. In this paper we describe a new approach to semantic wiki systems. In this approach the skills from users coming from different social networks are merged into a common collaborative space called MetaSocial. The MetaSocial project (a resarch proposal) introduces this approach and some of the challenges it presents.

1.1 Social Networks and Semantic Wiki Systems

Work by [2] and [3] report that users prefer to establish friend relationships with other similar users in a social space, regarding for instance socio-cultural traits. This has lead to the aggregation of people from different backgrounds into common spaces where they share similar interests and tastes. Social networks (SN) differ in the services they provide, targeting different demographics with different purposes. All SN are valuable in terms of aggregating people with distinguishable features. For instance LinkedIn[1] aims for professional exchanges

[1] http://www.linkedin.com

between individuals, while Orkut targets casual and leisure exchanges between family and friends. In that sense, SN help to define an individual in terms of a context that includes the type of people they target. Although research in SN have demonstrated the great potential that SN present for generating knowledge, less attention has been paid on how to harness users' knowledge in generating collaborative content. Some of the current SN offer capabilities for creating communities within the SN, however they dont provide tools for collaboration. Moreover, until the appearance of Open Social[2], the communication between independent SN was not possible.

The collaboration has been set aside from SN. Although collaborative systems, such as a wiki, provide tools for user participation into common tasks, e.g., discussion pages, there is still a lack of ties that prevent users from propagating information and from promoting their participation in a given task, by bringing this to the attention of the user's acquaintances. There is also still a gap between SN and collaborative systems. One possible solution to this situation would be to enable the capability of managing relationships within semantic wiki engines; however that would not be enough, as it would leave aside the potential benefit that the diversity of SN can bring into a common collaborative space.

1.2 Semantic Wiki and the linked Web

The semantic web is not about putting data together but about making links between the data [4]. So far, semantic wiki engines have acted as isolated content stores. Although efforts such as [5] have established the existence of links between same instances of data from different semantic wikis, there is still a broken link between the author of the data and the identity of the author. The integration of digital identities into semantic wiki engines can be perceived as an attempt to break away from one of the wiki principles regarding minimal access control, which refers to the capability of contributing in an anonymous way or by using a registered username, which can hold minimal authoring information.

Online identity can be defined as the representation of one's persona in a digital context[6]. It is worth noting that in the case of online identity the user has the ability to define a personal facade that represents him in a particular context. People in social environments tend to present just a facet of their identity for others to perceive. The integration of digital identities with wiki engines would allow the users to select their preferred persona to represent his authorship on a given wiki, while the wiki engine could still allow the anonymous authoring.

The integration of digital identities into wiki engines open a promising field in which the contextualization of a persona can be developed according to the activities carried out by this persona on the Web. Services integrating digital identities would not only pull information from the user's identity service provider, but also push information about the task carried out by the user that would help to better define the user's persona's interests and knowledge. The aggregation of information containing linked data coming from different data

[2] OpenSocial, http://www.opensocial.org

silos could help in building up a decentralized digital identity [7]. The openID version 2.0 OpenID[3] protocol for digital identity providers already defines a attribute exchange protocol however it doesn't provide a standard way of defining and developing users' personas' contexts.

Until now, just a few semantic wiki engines have integrated the openID standard into their engines however they use it just as an authentication service. On the other hand, none of them have introduced the use of the social intelligence included in the users' FOAF[4] files.

Incorporating that information would enable the merging of users' SN information (including profile, social graph and interests for instance). This information can help to correlate users coming from different SN. Consider a user who is part of an alternative rock community within meta-social and is interested in releasing his first album. He creates a new project with that subject. People with similar interests can be advised to join the project. The user should be able to try to establish a relationship with other users, not necessarily a friendship but a colleague relationship, advertising his project. In this way he could establish relationships with people from Myspace interested in music, which could advise him on the design of the album's cover, or with people from LinkedIn who could help him to better position his album on the market.

Integrating the user's identity and social information into semantic wiki engines would enable linkage between user's identity with the type of contents he adds (presumably the type of topics he is interested in) and the relationships he establishes. Keeping track of this information will enable better tracking of users' interests and would facilitate the improvement of suggestion engines. Moreover making this information available to the user's identity service provider, would help to decentralize the information given by the user, enabling it to be reused in any other system linked to the user's identity.

2 Proposed Approach

The proposed functionalities for Metasocial are contained within two separate parts. Each part depends on semantic technologies to enhance existing services. A semantic wiki plays a crucial role in bringing the below functionalities together as we now explain.

2.1 Social Functionalities

Metasocial will allow users to collaborate from multiple social networks, therefore managing user accounts will be addressed to map individual accounts in different social spaces. As mentioned, OpenID will be used as a single user identity URI for each user of Metasocial. Social graphs using both the FOAF and SIOC[5] specifications will be imported into Metasocial from multiple distributed social

[3] OpenID, http://openid.net
[4] FOAF, http://www.foaf.org
[5] SIOC, http://www.sioc.org

web platforms and linked together. The intuition behind this functionality is to enhance the information attributed to each user thereby offering intelligent suggestions based on their prior knowledge. Such social graph interlinking also contributes to current initiatives to address identity fragmentation and data portability [6].

Status information of each social network member will be described using the Online Presence Ontology [7], and with current initiatives such as Smesher [8] it is possible to convert such updates to semantic representations usable by the system. For example, if a person's status describes how they are busy working on a given project task, then Metasocial would not suggest additional work. Possible collaborations with work colleagues are suggested based on the imported social graphs, each social network member also has a list of interests extracted from the hosting service. Suggestions are then made based on such interests for specific projects. The converse is also true in that users are suggested projects and work based on their interests described in multiple social graphs. Combining such identity fragments we build a more complete profile of the user allowing projects spanning both the Semantic Web and Social Web to be suggested. We use this example as a very simple indicator of a trivial suggestion task [9].

A semantic wiki provides a useful means to control semantic graphs attributed to individual users, and make inferences based on the type of annotation the users have used when adding content. Metasocial will maintain a knowledge base capable of offering a useful collation of knowledge statements expressed within the wiki. One of the attractions of using a semantic wiki is the ability to effectively infer suitable projects suggestions and colleagues based on available semantic information.

2.2 Knowledge Functionalities

Within Metasocial collaborative environment projects will be described using the DOAP ontology[10] including extensions to this specification to capture knowledge describing more generic projects (at present DOAP is tailored more towards software based development projects). Looking for a project will be controlled using a semantic search mechanism by aligning the semantic concepts the user has expressed an interest in with similar projects. This allows more general and specialised projects to be returned if the user's original criteria are not explicitly matched.

Project management will require tools such as time planners, task managers and role assigners all usable on the semantic wiki. Social information assigned to users within the collaborative environment allows suitable roles to be suggested based on the semantics of the user details, i.e., `sioc:Role`. Managing

[6] Date Portability Group. http://www.dataportability.org
[7] OPO, www.milanstankovic.org/opo/ontology.html
[8] Smesher, http://smesher.com
[9] Twine platform already leverages users interests, and track of user's searches for suggesting content. See http://www.twine.com
[10] Doap, http://usefulinc.com/ns/doap

project milestones would be enhanced through interactions with semantically linked calendars; allowing project members to receive updates and reminders about upcoming milestones. Reminders would be controlled automatically by the semantic preferences stipulated by the project member.

Project work will be labeled using free text tagging, which is in turn aligned with concepts from a knowledge base, thus controlling term ambiguity and correct co-referencing. External knowledge sources can also be used [11] for greater availability of concept definitions. The internal knowledge base would allow discourse to be developed specific to that project so that knowledge generated as a result of this process could then be shared with additional projects. Natural language style queries could be asked, either returning any derived answers or relevant knowledge from the knowledge base, or allowing project members to answer the questions themselves. Completed projects tasks would become less visible to the user on the semantic wiki based on the task being semantically defined as complete. This would offer the functionality to display all completed projects of a specific type, enhancing the knowledge management functionality on the wiki and encouraging reusability of the existing projects.

3 Conclusions

Metasocial introduces various challenges, most of them concerning the representation of the user context in an standard format, the extraction of information (interests in particular) from user added data and the inference of information from the user's social graph. Offering a platform where people from different social networks can not only communicate, but also collaborate easily in the development of semantic-aware projects according to their interests will help in allocating the right user with the right skills in the right projects.

References

1. Reed, D. P.: That sneaky exponential beyond metcalfes law to the power of community building. Context Magazine (1999)
2. Rohilla Shalizi, C.: Social Media as Windows on the Social Life of the Mind CoRR abs/0710.4911: (2007)
3. Newman, M. E. J.: Mixing patterns in networks. Phys. Rev. E. Vol. 67, (2003)
4. Tim Berners-Lee:Linked Data. http://www.w3.org/DesignIssues/LinkedData.html (2008)
5. Passant, A., Laublet P.: Towards an Interlinked Semantic Wiki Farm. CEUR Workshop Proceedings. SemWiki 2008.
6. Russell, Terrell and Stutzman, Frederic :Proceedings of the American Society for Information Science and Technology. American Society for Information Science and Technology . Self-representation of online identity in collected hyperlinks. Vol.44, 2007.
7. Rowe, Matthew.: Proceedings of Linked Data on the Web Workshop 2009, Madrid, Spain. World Wide Web Conference. Interlinking Distributed Social Graphs. 2009.

[11] Such as DBPedia, http://dpedia.org and Freebase, http://www.freebase.com

Analysis of Tag-Based Recommendation Performance for a Semantic Wiki

Frederico Durao and Peter Dolog

IWIS — Intelligent Web and Information Systems,
Aalborg University, Computer Science Department
Selma Lagerlöfs Vej 300, DK-9220 Aalborg-East, Denmark
{fred,dolog}@cs.aau.dk

Abstract. Recommendations play a very important role for revealing related topics addressed in the wikis beyond the currently viewed page. In this paper, we extend KiWi, a semantic wiki with three different recommendation approaches. The first approach is implemented as a traditional tag-based retrieval, the second takes into account external factors such as tag popularity, tag representativeness and the affinity between user and tag and the third approach recommends pages in grouped by tag. The experiment evaluates the wiki performance in different scenarios regarding the amount of pages, tags and users. The results provide insights for the efficient widget allocation and performance management.

Key words: wiki, recommendation, tags, performance, adaptation

1 Introduction

Wiki is a collaborative knowledge space that can be edited by anybody who is granted permission [11]. Due to its simple usage, the wiki adoption is less about learning new technology and more about changing habits. A part from the complexity of existing Web solutions, wiki has instituted a new and democratic way of usage with simple text syntax for creating pages and cross links between internal pages on the fly. Although wikis provide an easy way for editing content pages, user interaction is still on "one way" i.e. users have to look at wiki pages to find interesting content to them. In the other direction, wikis could notify users about what they hide behind the currently viewed page. In this sense, recommendations can be utilized to lead users to unknown pages and reveal related topics addressed in the KiWi (KiWi - Knowledge in a Wiki). Furthermore, the recommendations can be tailored to user tastes and adaptively configured depending on system needs such as performance. The KiWi system addressed in this paper is a social semantic wiki in which individuals work collaboratively by editing content items and sharing knowledge. In serves as a platform for implementing and integrating many different kinds of social software services by allowing users to connect content in new ways that go beyond the level of the user interface, e.g. through semantic annotation [14].

In this work, we extend the KiWi system with three tag-based recommender approaches, which suggest links to wiki pages based on the similarity of their tags. The first approach recommends pages which share tags, the second approach takes into account external factors such as tag popularity, tag representativeness and the affinity between user and tag and finally the third approach groups recommendations by tags. The performance of the approaches is compared in different scenarios, which varies in terms of amount of pages, users and tags. The outcome from this analysis provides insights for widget allocation (where the recommendations are placed) and subsequent performance optimization. Our development is placed at KiWi [14], a semantic wiki for knowledge management built on previous experience in areas such as semantic web [5], semantic wiki [13] and personalization [6].

The paper is organized as follows: In Section 2 we discuss related work. In Section 3, a motivation scenario is presented. Section 4 introduces the recommendation approaches. Section 5 presents the experimental evaluation and results. A discussion about the results from the previous section is presented in Section 6 and finally in Section 7, we conclude the study and also point to future works.

2 Related Work

A number of semantic wiki applications have been explored over the last years and most of them utilize annotations to contextualize the content presentation and improve the navigation throughout all existing pages. *SemperWiki* is a semantic personal wiki developed for the Gnome desktop in which users can edit and annotate pages semantically [12]. In order to navigate through the wiki pages, users have to query pages containing certain annotation statements. The retrieval brings a list of links to the existing pages in the system. In addition, SemperWiki provides a history navigation section that allows users to go back and forth in their navigation history. The navigation support provided by SemperWiki is enhanced by a search and retrieval mechanism. We observe that the discovery of new pages in SemperWiki depends more on user's curiosity whereas the recommendations in KiWi are always displayed without imposing any additional work on the users. The history navigation however can be considered as a positive feature in SemperWiki because it is very practical for rapid navigation between visited pages. This feature can be adopted in KiWi to generate a new sort of recommendation triggered by history log of visited pages. Already *IkeWiki* [13] as a predecessor of KiWi provided a "references box" containing related pages triggered by annotation in the wiki pages. Similarly, *Semantic MediaWiki* [10] suggests related pages which share similar instances. Recommendations in KiWi are less formal than Semantic MediaWiki and IkeWiki since they are triggered by tags which are not bounded to any ontology. On the other hand, the flexibility of tags allows users to spill their personal feelings to a wiki page so that this generates more personalized recommendations.

Equally to KiWi, *OntoWiki* interface is surrounded by widgets that provide meta-information from semantic annotations and navigation support. Although

OntoWiki does not use tags for processing recommendations, it contains a particular widget for related pages categorized by the *Most Popular* and *Most Active* [1]. *HyperDEWiKi* is a semantic wiki intended to support domain ontology evolution [15]. It allows the user to define specific pages for instances of formally described types. In this sense, users can create a dynamic page that is better suited to support his tasks. We observe that the end view of KiWi and HyperDEWiKi can be fully customizable and render various set of information in different places and layouts. Besides the common wiki style editing with annotations, both systems provide personalized features tailored to user's tastes.

In general, the semantic wiki applications analyzed utilize annotations in the pages for navigation, rendering and search purposes. However, we observe that navigation is still centered on dynamically generated lists of related content with no or few personalized information. Personalization will drive the system features in accordance with individual tastes and preferences [7]. In addition, it is observed that adaptive techniques can be more explored in order to support wikis to present their content more intelligently [4]. Following these premises, we introduce three personalized tag-based recommendations and evaluate their allocation in KiWi interface aiming at performance optimization.

3 Motivating Scenario

In general, tags are assigned to Web resources in order to conceptualize, categorize and organize them in a way that users can be reminded later about the tagged content [9]. Invariably, tags represent some sort of affinity between user and the page that is being assigned. Users label pages freely and subjectively, based on their sense of values. This information provides useful hints about what a user thinks about the pages [8]. In this sense, we utilize tags to compute similarity between wiki pages and generate personalized recommendations without imposing any extra work on the users. We credit personalized recommendation as an important feature for supporting the main activity in wiki systems. For instance, when users are reading or editing a wiki page, recommendations of similar pages can be processed simultaneously and exhibited so that users can navigate through wiki pages following the topic that is being addressed. According to [3], recommendations have a significant importance because they expose alternative ways to the users fulfill their goals. In this sense, we provide recommendations in KiWi whereby users can follow links related to their interests, which assist them to achieve their tasks or bring further information for what they are looking for.

Figure 1 shows the current development in KiWi system in which tags assigned to the currently viewed page are located on the bottom widget on the left side (1). The recommendation widgets are highlighted on the right side: the widget number (2) contains the *standard recommendation*, the widget number (3) contains the *multifactor recommendation* and the widget number (4) contains the *recommendations grouped by tags*. The recommendations expose a variety of options for a user to visit just on a single click. If this activity is designed well,

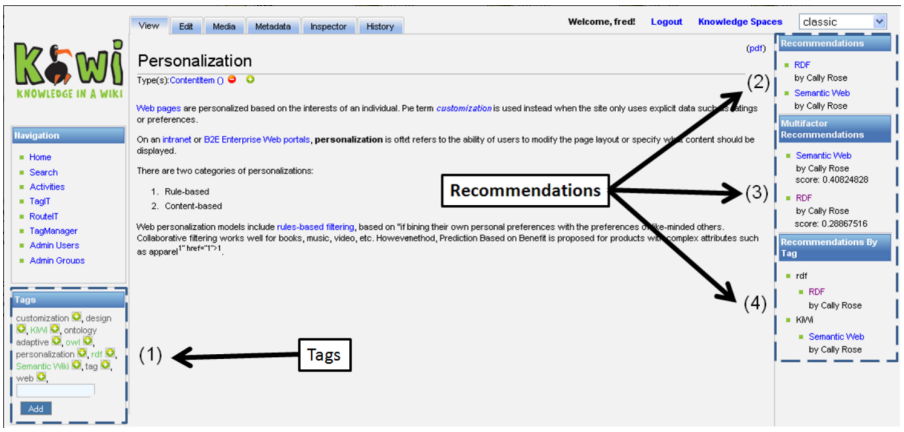


Fig. 1. Recommendations in KiWi

then the choice is easy, and the user keeps interacting with the system by visiting the related pages or adding new content. Although each recommendation approach has its own particularities (See in Section 4), very hardly the three solutions will run in parallel in real life scenario because they occupy much space in the user's interface and occasionally issue the same information (at the same time). In addition of being useless from the usability perspective, to have all three widgets running together compromise the system performance at all. As known, wiki is a collaborative space utilized for multiple interactions and any performance concern is always advisable. Based on these premises, the performance analysis is undertaken in order to find out widget combinations so that the overall performance is optimized and users take advantage of better widget arrangements.

3.1 Tags as semantic annotations for personalization

In this work we are mapping semantics of user activities based on tagging activity. Using uncontrolled tags, users are able to annotate pages without any restriction constrained by ontology vocabularies. In this sense, users are free to express their feelings about the page as they like on any purpose. The outcome of this tagging activity is a relation between user and a page through a tag property, as seen in the Figure 2.

Fig. 2. Relation between user and a page through a tag property

From the tag-based relationships, we are deriving personalized recommendations by computing similarities between tags, however, in later stages, other relevant information can be derived and utilized to annotate the wiki pages using RDF properties such as *ont:mostFrequentTag*, *ont:userMostInterested* and *ont:mostSimilarPage*. These properties would create a semantic network between content items in KiWi, and also could be utilized for other personalization goals such as group formation, semantic search and creation of link structures.

4 The Recommendation Approaches

This section depicts the standard, multifactor and recommendation grouped by tags addressed in this work.

Standard Tag-based Recommendation. In this approach, all pages that share tags with the currently viewed page are recommended. In this standard approach no further similarity processing is carried out therefore the list of recommendation is not ranked. The advantage of this approach is the performance since the recommendations relies simply on a data retrieval task. On the other hand, a single tag shared by pages may not be sufficient means to determine a similarity between pages. This approach however cannot be discarded without analyzing its applicability in the different possible KiWi scenarios. Figure 3 shows standard

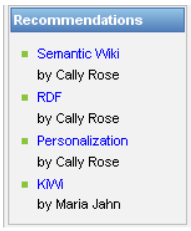


Fig. 3. Widget with Standard Recommendations

recommendations in KiWi with their respective authors.

Multifactor Recommendation. The multifactor recommendation approach computes similarity between pages considering multiple factors. The recommendations rely on calculus of *cosine similarity*, *tag popularity*, *tag representativeness* and *affinity user-tag*. We utilize a cosine similarity measure between tag vectors to calculate basic similarity of the pages. We measure tag popularity as a count of occurrences of a certain tag in total number of wiki pages. The term frequency measure is used to compute tag representativeness for a certain wiki page. The tag affinity between a user and a tag is calculated as a count of how many times the user utilized the tag at different web pages. We propose a formulae which

consider all these factors in a normalized way and gives a ranking of pages for particular user.

We define a page score as:
$Ps = \sum_{i=1}^{n} weight(Tag_i) + \sum_{i=1}^{n} representativness(Tag_i)$, where n is the total number of existing tags in the repository.

We define the tag user affinity as:
$Affinity_{(u,t)} = card\{p \in Pages \mid (u,t,p) \in P, P \subseteq U \times T \times P\}/card\{t \in T \mid (t,u) \in P_u, P_u \subseteq U \times T\}$, where t is a particular tag, u particular user, U is a set of users, P set of pages and T set of tags.

Finally, similarity is computed as:
$Similarity_{(P_i,P_{ii})} = [Ps_{P_i} + Ps_{P_{ii}} * cosine\_similarity(P_i, P_{ii})] * Affinity_{(u,t)}$.

Informally, each one of the factors in the above formulas is calculated as follows:

i. **Cosine Similarity** — Our tag similarity is a variant on the classical cosine similarity from the text mining and information retrieval [2] whereby two items are thought of as two vectors in the m dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors.
ii. **Tag Popularity** — Also called *tag weight*, is calculated as a count of occurrences of one tag per total of resources available. We rely on the fact that the most popular tags are like anchors to the most confident resources. As a consequence, it decreases the chance of dissatisfaction by the receivers of the recommendations.
iii. **Tag Representativeness** — It measures the relation between the tag and the resource it belongs to. It is believed that the most frequently occurring tags in the document can better represent the document. The *tag representativeness* is measured by the *term frequency*, a broad metric also used by the Information Retrieval community [2].
iv. **Tag Weight** — it is calculated as a number of occurrences of a tag divided by the overall number of tags in a repository.
v. **Affinity between user and tag** - It measures how often a tag is used by a user. It is believed that the most frequent tags of a particular user can reveal his/her interests. This information is regarded as valuable information for personalization means. During the comparison of two resources, the similarity is boosted if one of the resources contains top tags of the author from the other resources around.

Figure 4 shows the same recommendations as Figure 3 however sorted differently due to the quality factors calculus. In terms of performance, the multifactor approach is worse than the standard one however the ranked list provides credibility to the recommendations. Pages ranked higher are personalized to user's tastes and closer to the content discussed in the currently viewed page. Although the recommendations are more effective than the standard approach, its applicability also depends on further performance analysis in different KiWi scenarios.

Fig. 4. Multifactor Recommendations

Recommendations Grouped by Tags. In this approach, the recommendations are grouped by the tags which are assigned to the currently viewed page. Similarly to the standard approach, no further similarity processing is undertaken and the list of recommendation is not ranked. On the other hand, the user can go directly to the recommended wiki page just following the tag he/she is interested. The tag-based distribution explicitly provides a justification why the recommendations were generated and assist users to find related specific wiki pages.

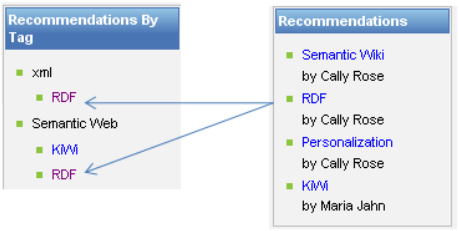


Fig. 5. Widgets with recommendations grouped by tags

The disadvantage however is the possibility of existing duplicated recommendations since two different pages can share two distinct tags as well. Figure 5 shows two tags *xml* and *Semantic Web* with their respective linking recommendations. The duplication problem is outlined since both tags recommend a link to the *RDF* wiki page. The performance analysis therefore will answer whether this duplication problem affects the performance to the point of discarding the applicability of this approach.

Scenarios	Pages	Tags	Users
1º	20	225	5
2º	50	500	10
3º	100	700	15

Table 1. KiWi Scenarios assembled for the experiment

5 Experimental Evaluation

The experimental evaluation assessed the performance of the recommender approaches by simulating a mix of scenarios regarding the amount of pages, users and tags. In addition, we discussed which widgets of recommendations should be displayed or suppressed in accordance with the performance findings. Although having in mind that standard approach is theoretically the most advisable approach in terms of performance, we evaluated when and in which conditions the other approaches become suitable and necessarily required. The goal therefore is to propose insights for widget adaptation aligned with running scenarios without compromising the system performance.

The proposed scenarios were created aiming at simulating realistic usage of KiWi. Nevertheless there is no "pattern" or "standard" about wiki activity. This is more about politics from where the system is deployed, maturity of whom is using it and time of activity. Although understanding that building wiki scenarios is a little subjective, we proposed three growing scenarios that are likely envisioned in KiWi life cycle as described in Table 1.

The variables addressed by each scenario are:

- *Amount of Pages* – each page has a set of tags that are compared for processing the recommendations. Therefore the more pages exist, the more time will be spent to calculate the similarity between the pages.
- *Amount of Tags* – the similarity of the pages is given by their tags. The whole set of tags of each page must be compared to verify which ones are similar. In particular, the amount of tags of an user impacts directly in the time for the computation of the affinity between user and tag.
- *Amount of Users* – the more users KiWi contains, the more are the chances of tagging activity. As a consequence, more time will be necessary to process the personalized recommendations.

These variables were chosen justified because they are row material for calculating the recommendations. This process is time consuming and invariably affects the system performance however it does not mean that other factors such as page size should not be considered. Finally, the scope of this work only comprises these three variables.

5.1 Methodology

Initially, the KiWi system was populated with pages and tags using a random generator to produce sufficient amount of content. The users were created manually since they were only 15 at most. The content of the pages and tags were

extracted from Web sites on the Internet and local documents. Similarly, we utilized a random generator to assign tags to wiki pages tagged by a particular user. In this study, it is not important to speculate about the random function we have utilized as we look at overall performance of the system and not the method for distributing the content. We needed just to generate adequate number of satisfactorily different pages and sufficiently different assignment of tags to them. To each scenario created, we collected the response time necessary to load the recommendations. We repeated this procedure for 10 times in order to have a more real and democratic results. In KiWi, the recommendations are processed every time a page is called, then we tested the performance after the user login, accessing a page from page link and from the own recommendations. The tests ran in machine equipped with processor Intel (R) Core (TM) 2 Duo CPU T7500 @2.20Ghz.

Expected Results Our assumption is that at some point due to the high amount of recommendations, the standard approach become ineffective although the performance continues better than the other approaches. The quality achieved with the multifactor recommendation will be needed even though the performance is decreased. Moreover, we believe the group recommendation is always useful due to its facility of identifying similar pages by tags however its performance may discourage its adoption due to the high number of duplicated pages.

5.2 Results from the First Scenario

The first scenario was setup with 20 pages, 5 users and 225 tags. Figure 6 shows 10 time stamps collected from KiWi system to calculate the recommendations for the three approaches. The average column from Figure 6 shows that standard recommendations were computed in 28 ms; multifactor recommendation in 77 ms and recommendations grouped by tags in 29.1 ms.

Approach	10 Timeshots(ms)										Average (ms)
CLASSICAL	15	15	15	31	47	47	15	31	32	32	28
MULTIFACTOR	15	16	109	93	172	109	6	110	62	78	77
GROUPED	47	10	16	78	16	16	15	15	63	15	29.1

Fig. 6. Results from the First Scenario

The standard and grouped approaches had better performance than the multifactor approach. While standard and grouped approaches achieved rates quite close to one another (about 28ms), the multifactor approach spent 175% more time than both approaches to calculate the recommendations. As already known, the multifactor recommendation generates a ranked list of recommendations more personalized and reliable. The point to be assessed therefore is whether

the ranking compensates this time necessary for ranking the recommendations. In this particular case where only 20 wiki pages are considered, the multifactor approach may be ignored depending on the visibility of the recommended pages to the users. If the widgets of standard and grouped recommendations are able to expose the whole recommendations with easy access, they will be continuously exposed and hardly forgotten. In this sense no ranked recommendation is necessary.

5.3 Results from the Second Scenario

The second scenario was setup with 50 pages, 10 users and 500 tags. Figure shows 10 time stamps collected from Kiwi system to calculate the recommendations for the three approaches. The average column from Figure 7 shows that standard recommendations were computed in 97 ms; multifactor recommendations in 121 ms and grouped recommendations in 45.3 ms.

Approach	10 Timeshots(ms)										Average (ms)
CLASSICAL	94	78	78	47	63	78	125	110	125	172	97
MULTIFACTOR	103	92	112	78	62	93	124	212	166	168	121
GROUPED	93	15	15	16	54	78	93	31	47	11	45.3

Fig. 7. Results from the Second Scenario

The grouped approach achieved the best performance followed by standard and finally by the multifactor approach. The multifactor approach lasted approximately 25% more than the standard approach to generate the recommendations. Comparing to the first scenario, it is observed a considerable approximation between standard and multifactor approach (from 125% to 25%). The issue to be discussed therefore is whether the standard recommendation approach is still desired due to the low performance variation between the standard and multifactor approach. In this scenario, 50 wiki pages are addressed and the recommendation widgets tend to enlarge with the increase of recommendations. In this case, therefore, a ranking approach becomes useful since the most similar pages are placed at the top of the list of recommendations. In addition, very hardly the recommendation widget will provide an ample visibility of whole set of recommendations. In this sense, the 24 additional milliseconds to generate the multifactor recommendations compensate the probability of having ineffective recommendations. The high performance of the group based approach can be justified by the low amount of duplicate recommendation. Furthermore, it is important to observe that although the amount of pages of this scenario is the double comparing to the first one, the current performance was decreased of only 55.6% (or 16.2 ms) from the first measure. This approach therefore becomes a strong candidate to be utilized in this particular scenario even in combination with the multifactor recommendations.

5.4 Results from the Third Scenario

The third scenario was setup with 100 pages, 15 users and 700 tags. Figure 8 shows 10 times stamps collected from KiWi system to calculate the recommendations for the three approaches. The average column from Figure 8 shows that the standard recommendations were computed in 98.1 ms; multifactor recommendation in 145 ms and grouped recommendations in 53.1 ms.

Approach	10 Timeshots(ms)										Average (ms)
CLASSICAL	93	87	94	63	141	93	94	110	125	81	98.1
MULTIFACTOR	109	113	141	163	188	141	140	187	156	112	145
GROUPED	32	46	62	63	52	43	52	31	78	72	53.1

Fig. 8. Results from the Third Scenario

Similarly to the second scenario, the grouped approach achieved the best performance followed by standard and finally by multifactor recommendation. In this turn the standard approach was approximately 50% faster than the multifactor approach. This is a considerable difference that stands face-to-face two goals: performance and effectiveness. On one side, KiWi process lots of recommendations very quickly however unsorted, on the other hand the same recommendations last at least 50% more but they are ranked. A qualitative experiment in which the users show their satisfaction in terms of performance and effectiveness would indicate in which direction to go. A possible solution however is to combine the multifactor approach with the fastest group recommendation, which attenuates the overall loss of performance for this scenario. The group approach performance decreased only 17% from the second scenario (from 45.3 ms to 53.1 ms) and still provide a special distribution of the recommendations.

5.5 Overall Results

The performance of the approaches in each scenario analyzed are presented in the Figure 9.

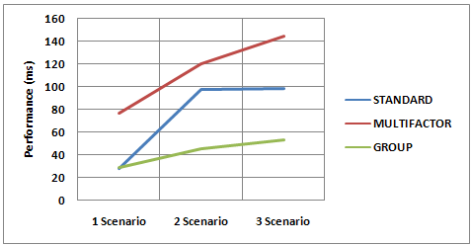


Fig. 9. Evolution Graph

In the first scenario, the multifactor approach was considerably more expensive (in terms of performance) than the others. This significant cost for generating recommendations discourages its adoption. In the second scenario, the standard approach reduces significantly its performance, which encourages the use of the multifactor approach. From the first to the second scenario, the group approach maintains a satisfactory level of performance. In the third scenario, standard and grouped approaches keep their performance approximately equal to the previous scenario, whereas multifactor approach presents a significant loss of performance.

6 Discussion

The performance outcomes from the scenarios analyzed allow us to suggest intelligent widget allocations without compromising the system performance. For the first scenario, the standard and group approaches are the most advisable to run together spending in total about 57.1 ms to calculate the recommendations. For the second scenario, we suggest the multifactor and group approaches running in parallel spending in total 166.3 ms and for the third scenario, due to the need of quality, again the multifactor and group approaches are the most advisable spending together about 198.1 ms to calculate the recommendations. Figure 10 shows the performance with the suggested combination to each scenario analyzed.

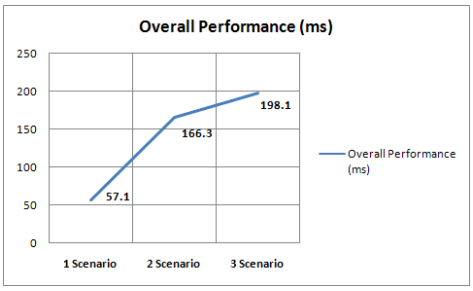


Fig. 10. Suggested Performance Graph

According to Figure 10, the performance for the first scenario is 57.1 ms, which is significantly low since two approaches of recommendation are being computed. In the second scenario, it is observed an expressive loss of performance (from 57.1ms to 166.3 ms) mainly due to the utilization of the multifactor approach. In the third scenario however the loss of performance was 31.8 ms, which is much less than the loss of performance from the first to the second scenario (109.2 ms). In fact, the multifactor approach utilized in both second and third scenarios reduced the performance, however, we assure that the best recommendations will be placed in the top of the list of recommendations. In general,

the overall result obtained shows the tendency of the performance whenever the amount of user, page and tag grows. Although the more scenarios are advisable for confirming this tendency, this preliminary outcome can be employed for predicting the system performance in emergent scenarios.

Other advantage from the results obtained, besides the widget allocation, is that they show a tendency of the performance whenever the amount of user, page and tag grows. The achieved numbers can be utilized for predicting the system performance in emergent scenarios. On the other hand, the amount of the scenarios analyzed is still low to confirm this tendency. More experiments, with a bigger setup and with multiple users using the system at the same time would provide a more realistic feedback about the performance.

7 Conclusion and Future Works

This paper analyzes the performance of three tag-based recommender approaches for a semantic wiki. Three different scenarios were assessed varying in terms of number of pages, amount of tags and users. To each scenario assembled, it was analyzed which recommendation approaches could be more appropriate taking account the system performance and user needs. The results showed that the combination between standard and group approach is feasible for scenarios up to 20 pages, which are constantly accessed. For scenarios with 50 and 100 pages with more than 10 users, the multifactor and group approaches are more advisable in spite of being more expensive in terms of performance. The grouped recommendation approach is always adequate since it provides justification for recommendation and visual support for navigating among the recommendations.

As future works, first, pre computation of some factors in multifactor recommendation will be studied to further increase performance of recommendation computing. Furthermore, semantic relatedness between tags must be considered since current recommendations in KiWi only consider the tag syntax to identify similarities. The next step therefore is to combine the tag algorithms with some reasoning on the annotations to provide more efficient recommendations. Another direction is to annotate tags with their role (or purpose) in order to formalize the relationship between tags and pages. The semantic recommendation will likely be more efficient by capturing precise needs of the users expressed by the annotations.

8 Acknowledgment

The research leading to these results is part of the project "KiWi - Knowledge in a Wiki" and has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211932.

References

1. S. Auer, S. Dietzold, and T. Riechert. OntoWiki - A Tool for Social, Semantic Collaboration. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe,

P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.
3. K. Bischoff, C. S. Firan, W. Nejdl, and R. Paiu. Can all tags be used for search? In J. G. S. et al., editor, *Proc. of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008*, pages 193–202, Napa Valley, California, USA, Oct. 2008. ACM.
4. P. Dolog. Designing adaptive web applications. In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *Proc. of Sofsem 2008: The 34th Intl. Conference on Current Trends in Theory and Practice of Computer Science*, volume 4910 of *LNCS*, pages 23–33, Novy Smokovec, Slovakia, Jan. 2008. Springer Verlag.
5. P. Dolog, B. Simon, T. Klobucar, and W. Nejdl. Personalizing access to learning networks. *ACM Transactions on Internet Technologies*, 8(2), Feb. 2008.
6. P. Dolog, H. Stuckenschmidt, H. Wache, and J. Diedrich. Relaxing rdf queries based on user and domain preferences. *Journal of Intelligent Information Systems*, 2009. published online.
7. F. A. Durao, P. Dolog, and K. Jahn. State of the art: Personalization. Technical report, 2008. KIWI project FP7, Project Number: ICT-2007.4.2-211932 Document number: ICT211932/SFRG/D2.7/D/PU/b1.
8. H. Halpin, V. Robu, and H. Shepherd. The complex dynamics of collaborative tagging. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *Proc. of the 16th Intl. Conference on World Wide Web, WWW 2007*, pages 211–220, Banff, Alberta, Canada, May 2007.
9. R. Jschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In A. Hinneburg, editor, *Workshop Proceedings of Lernen - Wissensentdeckung - Adaptivitt (LWA 2007)*, pages 13–20. Martin-Luther-Universitt Halle-Wittenberg, sep 2007.
10. K. Lassleben, D. Vrandecic, and M. Vlkel. Semantic mediawiki. In *The Semantic Web - ISWC 2006*, pages 935–942, 2006.
11. A. Majchrzak, C. Wagner, and D. Yates. Corporate wiki users: results of a survey. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 99–104, New York, NY, USA, 2006. ACM.
12. E. Oren. SemperWiki: a semantic personal Wiki. In S. Decker, J. Park, D. Quan, and L. Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop, 4th International Semantic Web Conference*, Galway, Ireland, November 2005.
13. S. Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 388–396, Washington, DC, USA, 2006. IEEE Computer Society.
14. S. Schaffert, F. Bry, P. Dolog, J. Eder, S. Grünwald, J. Herwig, J. Holý, P.-A. Nielsen, and P. Smrž. The kiwi vision: Collaborative knowledge management, powered by the semantic web. Technical report, August 2008. FP7 ICT STREP KiWi project Deliverable D8.5.
15. D. Schwabe and M. R. da Silva. Unifying semantic wikis and semantic web applications. In C. Bizer and A. Joshi, editors, *International Semantic Web Conference*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

An Extensible Semantic Wiki Architecture

Jochen Reutelshoefer, Fabian Haupt, Florian Lemmerich, Joachim Baumeister

Institute for Computer Science, University of Würzburg, Germany
email: {reutelshoefer, fhaupt, lemmerich, baumeister}@informatik.uni-wuerzburg.de

Abstract. Wikis are prominent for successfully supporting the quick and simple creation, sharing and management of content on the web. Semantic wikis improve this by semantically enriched content. Currently, notable advances in different fields of semantic technology like (paraconsistent) reasoning, expressive knowledge (e.g., rules), and ontology learning can be observed. By making use of these technologies, semantic wikis should not only allow for the agile change of its content but also the fast and easy integration of emerging semantic technologies into the system. Following this idea, the paper introduces an extensible semantic wiki architecture.

1 Introduction

Semantic wikis have become an attractive solution for collaborative knowledge formalization and sharing. One major challenge at that task is the fact that knowledge can be represented on many different levels of formality and in a wide variety of formalisms [1]. Further, the requirements for a semantic wiki application strongly depend on the domain and the targeted community. In consequence, a wide range of diverse semantic wiki approaches has evolved employing different techniques, e.g. different types of formalized content or reasoning capabilities. While, for example, Semantic Media Wiki [2] focuses on efficient reasoning on large data, IkeWiki [3] allows for easy ontology editing with rich expressiveness. SweetWiki [4] provides an elaborated *Wiki Object Model*. The OntoWiki system [5] supports the combination and visualization of multimedia data. AceWiki [6] follows a different knowledge acquisition strategy using a controlled language. Further wikis work with mathematical, prolog-based or classification knowledge [7–9]. All these systems were created having various distinct application scenarios in mind. Accordingly, they are utilizing a wide range of strategies and technologies, e.g., efficient reasoners, specialized reasoners, controlled languages, various markups, different content browsers and visualizations. Each of these systems is fitting well to its particular intended purposes but if one intends to support a specific (novel) semantic wiki application each of the systems reveals advantages *and* disadvantages. Without a suitable extensible semantic wiki one has to choose a suboptimal solution or implement an entire new semantic wiki system from scratch. We claim that the optimal solution to support a semantic wiki application needs to be adapted precisely to its requirements, considering the domain, the targeted user community and the envisioned

use-cases thoroughly. This analysis reveals the appropriate formalisms, reasoning support, visualization and querying capabilities needed. But in order to serve a wider range of these possible requirements and to be able to optimize each semantic wiki application to its domain and community it is beneficial to have an extensible semantic wiki architecture with a basic toolkit for knowledge formalization, knowledge visualization and reasoning. We envision that this kind of extensible semantic wiki architecture enables the customization of a semantic wiki system to a particular application at low (software) engineering costs offering various reusable and extensible components. Providing optimal support to any (non-technical) domain and community will raise the general acceptance and spread of this technology. Beside selection and integration of existing techniques into a semantic wiki also the agile integration of novel technologies can improve the general semantic wiki functionality and customization to specific applications. We presume, that especially advanced reasoners and NLP-techniques employed for semi-automated knowledge formalization can bring considerable benefit when being combined with the existing wiki technology.

In this paper we describe the concept of an extensible semantic wiki architecture and motivate the emerging possibilities. With the system KnowWE we present a prototype of an extensible semantic wiki architecture and show its current extensions.

2 Challenges and Dimensions of extending Semantic Wikis

We briefly outline a conceptual view on semantic wikis in general followed by the discussion of the possibilities and challenges extending semantic wikis. Figure 1 shows the three components of what we call the "knowledge pipeline" in semantic wikis. It shows the flow of the formalized knowledge from the contributing user role to the consuming user role through the *Knowledge Formalization Component*, the *Reasoning Component*, and the *Knowledge Presentation Component*.

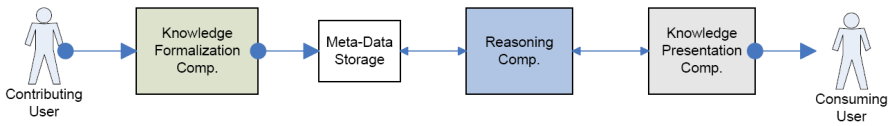


Fig. 1. Sketch of the "knowledge pipeline" of a semantic wiki.

– **Knowledge Formalization Component:** The knowledge formalization component allows the user to formalize parts of the (textual) knowledge. This usually is done by markups or (semantic) forms. These markups are extracted and transformed into a target representation which is commonly

stored explicitly (e.g., in RDF) to allow for efficient reasoning. This transformation implicitly defines the semantics of the formalized knowledge having the target reasoner and the ontology in mind.
- **Reasoning Component:** A reasoning component uses the formalized knowledge created by the knowledge formalization component and is able to deduce higher-level information from it. While most semantic wikis employ an RDF-reasoner, there are several other reasoning approaches present, that are beneficial in particular application scenarios.
- **Knowledge Presentation Component:** This component describes the method how the additional functionality provided by meta-data and reasoner is used to supply the user with the right (high-level) information in a suitable form. This includes as an important aspect the visualization of the results or functionality. The reasoning capabilities can be used to provide semantic navigation, querying, rendering fact sheets, meta-data browser, and more.

These components together provide the additional value of a semantic wiki. In the following the possibilities of extending each of these components are discussed in more detail.

2.1 Dimensions of Semantic Wiki Extensions

As mentioned in Section 1 the possible extensions of semantic wikis are manifold and cannot be foreseen in general. In order to allow for many kinds of extensions we discuss each of the three components separately:

1. **Formalization extension:** Given any methods (e.g., markup) to insert atomic formal relations, in a technical point of view any knowledge base can be created. However, the widespread employment of semantic technology is hindered by the formalization task being not simple and efficient enough (*Knowledge Acquisition Bottleneck* [10]). One way to counteract is lowering the barriers of knowledge formalization. The development of (domain specific) high-level markup languages with comfortable editing support can help to make knowledge definition compact, transparent and efficient. The use of controlled language is one approach in this direction [11]. Another possibility for reducing the workload of the domain specialists is the integration of (preconfigured) text mining methods, that propose formalizations based on the informal text content. Thus, the users only have to decide whether to confirm or dismiss a formalization proposition.
2. **Reasoning extension:** Although basic reasoning engines are currently available there are still challenges with respect to scalability and expressiveness [12] to be addressed. Further, there is ongoing research to cope with inconsistent knowledge, incompleteness and uncertainty [13–15]. For some applications it will be valuable to replace or enhance the basic reasoning engine by an early prototype result from such research work.
3. **Presentation extension:** The challenge of these kinds of extensions is to present the user the right high-level information in the right form at the

right time (without overflowing him). These extensions must be specified according to the use-cases addressed by the intended application. One frequent application might be precompiled (possibly parameterized) use-case specific queries decorated by a GUI component for execution and having a visualization component attached for result presentation (e.g., table-based, graph-based, highlighted).

When designing an extensible semantic wiki architecture these three levels of extension need to be considered as shown in Figure 2.

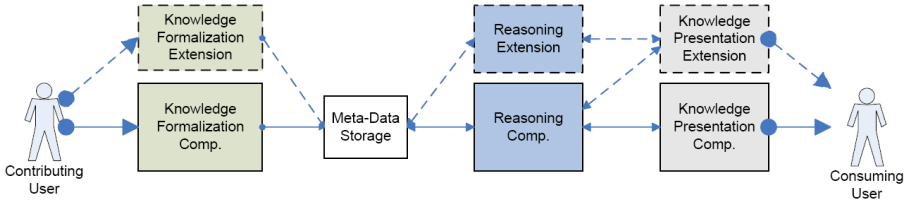


Fig. 2. Sketch of the "knowledge pipeline" of a semantic wiki with extensions.

Examining the possibilities of the extensions of each level it becomes evident, that an extension on one component can extend the entire functionality of a semantic wiki system. Thus, the three components can be extended separately or combined. This results in the semantic wiki extension space sketched in Figure 3. Assuming that the core semantic wiki system itself already provides some functionality in each component/dimension extensions on the three dimensions can separately or combined contribute to the total functionality of the semantic wiki. Hence, an extensible semantic wiki architecture should allow for (independent) extension of these three dimensions. If the core functionality of the extensible semantic wiki nearly fits the requirements single dimensions can be extended denoting "refining" extensions. Heavy-weight extensions along all three dimensions might have their own language, reasoning and presentation functionality. To clarify the threefold distinction in a more practical context we present three extensions along three, two and one dimensions respectively in Section 3.

2.2 Decorating Semantic Wikis

As already mentioned in the introduction we claim, that a semantic wiki system should be precisely tailored to a semantic wiki application considering the domain, community, and use-cases. This method is in compliance with the ideas presented recently by Yaron Koren at the semantic wiki mini series[1]. One can assume that there are many domains where semantic wiki technology could be employed beneficially. One must not assume that every possible user in any domain is able *and* willing to get used to concepts like *Semantic Wiki*, *ontology*,

[1] http://ontolog.cim3.net/cgi-bin/wiki.pl?ConferenceCall_2008_12_11

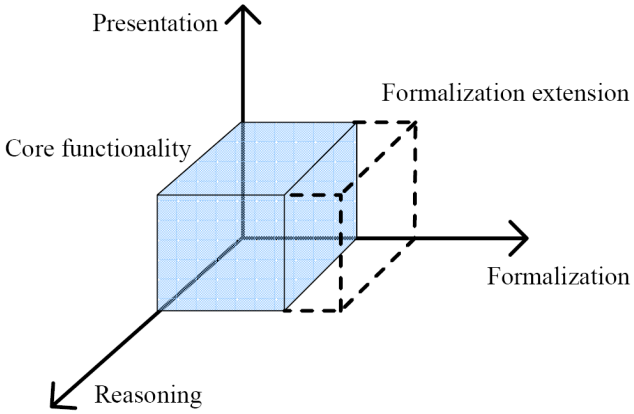


Fig. 3. The semantic wiki extension space.

RDF-triple, *SPARQL* or *DL-Reasoning*. Nonetheless it is possible to create semantic wikis that allow for efficient knowledge sharing and use for these user groups at the cost of some customization. The intended use cases and the user's mental model of the knowledge need to be identified in advance. Then, the ontology capturing the knowledge that is necessary to support the use cases can be modeled. Further, a method for knowledge formalization (e.g., user and domain specific markup) is designed that fits the mental model of the users. This markup or editing component implicitly creates (possibly complex) RDF-structures. At last an extension of the knowledge presentation component is implemented. This extension exactly supports the use cases revealed be the requirements analysis. It executes the necessary calls of the reasoning engine and presents the result in a visualization matching to the mental model of the user. One example might be some buttons with underlying predefined SPARQL queries with result sets rendered in some (domain specific) visualization. Any technical details (e.g., property names, creation of triples, SPARQL queries) are completely hidden from the user. The typical extension pattern for decorated semantic wikis is shown in Figure 4.

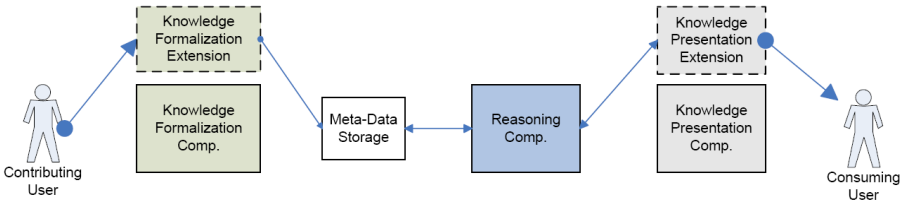


Fig. 4. Extension pattern of a "decorated" semantic wiki application.

The core functionality for knowledge formalization and knowledge presentation should be hidden from the untrained user to reduce confusion. This approach has been implemented on the HermesWiki project that is described in more detail in Section 3.

2.3 Challenges towards an Extensible Semantic Wiki

In the following we discuss the three most important aspects when designing an extensible semantic wiki:

1. **Basic functionality:** In order to allow powerful extensions with advanced features with little implementation costs, it is necessary to decide what semantic core-functionality comes along with the basic semantic wiki architecture. Enabling applications that have to deal with large data sets and high user activity, requires a slim and scalable text-processing and reasoning engine. Reasoners that focus on processing of more expressive or inconsistent knowledge are often consuming more computational power and need to be included by an extension if necessary.
2. **Usability:** One of the most important reasons for the wide acceptance and success of wikis is their high usability and the low training costs for new users. Semantic wikis are bringing new possibilities to wikis and thus are inevitably adding some complexity to its usage. Adding these new functionalities to the Wiki interface with the least mental overload is a critical issue in semantic wiki design. The core strength of the Wiki approach being simple otherwise can easily vanish. In every case it is sensible to enable 'non-semantic' users to work like in a 'non-semantic' wiki with no adaptation allowing them to discover single additional functionalities step by step. We propose the idea of a 'shadow'-semantic wiki hiding all of its advanced functionality at the beginning. Advanced features (e.g., fact sheets, meta-data browsers) can be added to pages using tags or configuration settings by more experienced users when necessary.
3. **Extension mechanism:** Various extension mechanisms on the software engineering level are applicable to create feature-rich and flexible software. However, there are several challenges specific to the context of semantic wiki functionality. The mechanism should be able to support light-weight "refining" extensions in a very simple way and at the same time still allow for complex extensions (e.g., with own markup, meta-data representation, reasoners and result visualization). In general, an unpleasant issue in modular software engineering in general is the (programming-)language barrier. For technical reasons combinations of software components written in different programming languages are often insufficiently manageable and inefficient. Unfortunately, this poses some kind of barrier for employing various implementations of semantic technologies to an extensible semantic wiki architecture.

The best solutions to all these aspects cannot be stated straight forward containing several trade-offs that might have to be revised after future experiences.

3 The Extensible Semantic Wiki KnowWE

We have designed and implemented the extensible semantic wiki *KnowWE* (Knowledge Wiki Environment) aiming to the concept of extensibility as stated in Section 2. As many software components on NLP and reasoning engines are implemented in Java, it appears helpful having our system also implemented in this language. We especially focused on the simple definition of new markups and tools allowing for easy text-based refactoring of these. In this chapter we introduce the core functionality of KnowWE. Additionally, three existing extensions are presented to demonstrate our approach.

Currently, KnowWE runs on top of JSPWiki[2], but it can be easily used with any (java-based) wiki engine having an extension mechanism similar to JSPWiki's *PageFilter* concept. Only the wiki connector providing page- and user-management has to be reimplemented accordingly.

3.1 The KnowWE Core

In KnowWE each article content is held in a tree-based data-structure. This tree can be used for refactoring, rendering (e.g., personalization, syntax highlighting), navigation, external editors and also allows the mapping between the textual content and the extracted meta-data. Arbitrary semantic extensions along any dimensions can be created without touching the core components of KnowWE. For any declared formalization extension (e.g., custom markups) the parse-tree is generated and hang up into the article content tree.

To provide a flexible library of reusable components, we integrated tools that are valuable for a wide range of possible scenarios managing formalized knowledge in a Wiki:

- A semantic renaming tool and an annotation browser which is demonstrated Section 3.4
- Table editing functionality allowing for visual editing of structured data.
- A flexible include mechanism for reusing arbitrary page snipplets across wiki pages for modular content and knowledge management.
- Parsing components for table-based, line-based, xml-based and regex-based markups to simplify the declarative definition of markup without or low efforts on parsing functionality.

Without any extensions, KnowWE comes with a set of basic functionality comprising the general features of a semantic wiki. To include formalized knowledge the KnowWE core version provides simple markup and the possibility to import knowledge from external sources. New properties can also be introduced ad-hoc to the system by the use of property definition sections on every wiki page. The properties defined within these tags are not created as standard OWL properties but rather as N-ary relations[3]. This allows us to automatically add

[2] http://www.jspwiki.org
[3] http://www.w3.org/TR/swbp-n-aryRelations/

supplementary information to the created knowledge. One of the automatically added nodes is a *TextOrigin* node that contains a reference to the textual position within the wiki text where the annotation was made, as well as revision information of the annotation's creation. There are several predefined properties which can be used to create annotations. Those key properties like `subClassOf`,`type` and `subPropertyOf` are treated separately from the user created properties. They are imported into the wiki-knowledge as their RDFS counterparts, allowing the reasoner to work on the generated knowledge without further translations.

An annotation is created by a simple link-like markup syntax. In its basic form an annotation is similar to the markup used in Semantic MediaWiki [2]. The following example is taken from a consumer domain describing digital cameras and shows the annotation of the concept associated with the local page by the property *hasBrand* with the value *Canon*.

```
Canon released the new 50D [hasBrand::Canon] a while ago.
```

In this form the subject of the annotation's property is the default concept of the page. Beside the double square brackets the differences to a Semantic MediaWiki annotation is that the annotation in this form has no explicit textual representation in the page view. Thus there is no Link to *Canon* created in the text but the formal knowledge is attached to the preceding word in this case. The markups can be extended by optional components like a subject different from the current page concept. Another way of modifying an annotation is by including a specific piece of text to be annotated. The following example shows the annotation of the text phrase "new camera" by a formal relation connecting the camera instance *Canon EOS 50D* and the brand instance *Canon* by the *hasBrand* relation.

```
Canon released the [new camera <=> Canon EOS 50D hasBrand::Canon]
a while ago.
```

This additional syntax provides more flexibility compared to the standard annotation. It allows to define relations outside the wiki page representing the instance. Although recommended in KnowWE, it is technically not necessary that every concept has its own wiki page. Further, it allows for precise attachment of a formal relation to a text phrase for documentation. This also enables queries to find all text phrases that for example were annotated with a *hasBrand* relation. KnowWE also provides a basic fact sheet markup to define multiple relations in a compact manner.

Our wiki uses the Sesame RDF storage[4] for saving and querying the created knowledge. The reasoning capabilities of our system are provided by the OWLIM engine (version 3.0b9)[5]. The default setting on the reasoning level is owl-max providing full RDFS reasoning as well as OWL-Lite semantics.

[4] http://www.openrdf.org
[5] see http://www.ontotext.com/owlim/

The reasoning and the N-ary representation of properties provide the background to answering SPARQL queries on the wiki ontology and imported ontologies. A SPARQL query is embedded into a wiki page using the simple XML-tag *sparql*. The query itself is being processed by the sesame query engine and the results are rendered in a table on the wikipage. For example, the following rather simple query produces a list of all concepts, in this case all cameras which are produced by `Canon`.

```
partial product list described in this wiki:
<sparql render="links">
SELECT ?cam
WHERE {
?t rdf:subject    ?cam  .
?t rdf:predicate  lns:hasBrand .
?t rdf:object     lns:Canon .
} LIMIT 5
</sparql>
```

The abbreviation `lns` is replaced by the local namespace which is constructed from the url of the wiki installation. The results are rendered as links to the pages where they are defined as shown in Figure 5. Omitting the `render` attribute creates a simple table of the Canon products.

Fig. 5. The result of a simple query for Canon products.

3.2 The d3web Extension

The d3web extension is a KnowWE extension to enable the definition and use of classification knowledge. We outline this extension only briefly since it has previously been described in detail [9].

– **Knowledge formalization extensions:** Different markup languages have been developed to capture various types of classification knowledge, e.g., covering models, rules, decision trees [16]. Here, the KnowWE architecture

allows for the integration of context sensitive editing support and syntax-highlighting. The textual markup is compiled into two different representations. First it is transformed into the proprietary object structure that is used by the integrated d3web[6] reasoning engine. To exploit querying capabilities, it additionally is compiled into an OWL-ontology using an upper ontology for classification which is explained more detailed in [17].
- **Reasoning extensions:** The d3web project summarizes a set of reasoners for diagnostic problem-solving. The reasoning engine is employed in this extension to work on explicit knowledge created by the use of the markup languages described above.
- **Knowledge presentation extensions:** Beside visualization and browsing mechanisms for the formalized problem-solving knowledge several possibilities for the execution of knowledge bases are included. A user can initiate a problem-solving session either by starting a structured interview or by freely answering questions that are rendered in the wiki pages. After each entered answer the currently most appropriate solutions (calculated by the d3web reasoning engine with the knowledge base) are displayed. Such a problem-solving session can be considered as an incremental personalized query to a classification system.

The d3web extension represents a heavy-weight form of a semantic wiki extension coming along with libraries containing reasoners, parsers and dialog components. However, this extension is especially intended for small to medium sized data sets and small user communities.

3.3 The HermesWiki Extension

The HermesWiki is a KnowWE-based Wiki in the historical domain developed in German language. It is built in cooperation with historians from the University of Würzburg. The main purpose of the HermesWiki is to provide an overview on ancient Greek history for teaching purposes of (undergraduate) students. Additionally, the Wiki provides direct links from the descriptions of historical events to translations of their (historical) sources. The Wiki consists of three parts: A collection of about twenty essays giving a comprehensive domain walkthrough, translations of the describing ancient sources, and an extensive glossary. Entries in the glossary are semantically tagged, e.g., as "politician" or "poet". The project, started in summer 2008, currently features more than 500 wiki articles, often illustrated with maps and pictures.
Technically, the wiki implements extensions along the dimension of formalization and presentation. The most important formalization extension of the HermesWiki is a specialized markup to explicitly define historical events in the main essay articles. This markup was developed in cooperation with the historians to allow for maximum usability in this community. Each historical event is defined in its own text block, structured as the following example:

[6] www.d3web.de

```
<<Lamian War (2)
323b-322b

After Alexanders death the Greeks revolted against Macedonian rule
under the lead of the Athenians.
[...]

SOURCE: Paus.::1.25.3-6
SOURCE: Diod.::18,8-18
>>
```

Each event is enclosed with double angle brackets. In the first line the title of the event ("Lamian War") is given, followed by a single number in parentheses, which describes the importance of the event. For example, events with an importance rating of '1' are considered essential while events with a rating of '3' are categorized as "additional knowledge". In the second line of the markup the point or interval in time when the event occurred is noted in a compressed form, e.g., the string "323b-322b" points out, that the event occurred from 323 BC until 322 BC. Further annotations can reflect more precise dating as well as uncertainty. After one empty line a free text description of the event follows. At the end of each event block ancient sources of the event are mentioned, explicitly marked by the keyword "SOURCE:" as the first word of a new line. This markup for historical events is currently used around 600 times in the HermesWiki. A time event will be modeled in OWL using a small ontology containing a class for timeline events with the properties as *hasImportance*, *hasTimeDesignation*, *hasDescription* and *hasSource*. Having this information extracted to OWL allows for several forms of exploitation in the presentation dimension. HermesWiki can generate different views on the timeline events by filtering them on constraints regarding the time, in which an event occurred, on event importance, and on the article, where an event was defined or on sources occurring. Figure 6 shows an exemplary generated view on the timeline in the HermesWiki featuring parts of the conquests of *Alexander the Great* (in German language). The importance of the events is color coded. One (domain-specific) use case supported by this extension is the (semantic) navigation "through the time" using the links provided by the ordered time line views. This Wiki project demonstrates how a rather light-weight domain specific extension along the dimensions of formalization and presentation can yield a "decorated" wiki as described in Section 1 that is usable by domain specialists which are not familiar with semantic techniques in general.

3.4 A POS-Tagger Extension

To demonstrate the possibilities achieved with low efforts extending KnowWE we have implemented a small toy-extension to enable Part-Of-Speech-Tagging. Further, it serves as a tutorial for the KnowWE extension mechanism. To point out the simple integration of NLP-tools and libraries we employed the Stanford

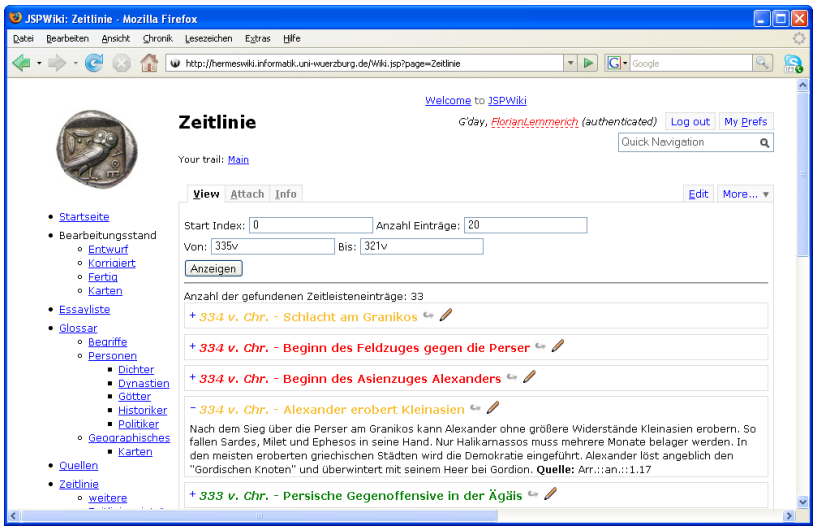


Fig. 6. A generated timeline in the HermesWiki

POS-tagger[7] to analyze the wiki content. Figure 7 shows the resulting wiki view with this extension activated.

In this example configuration the extension calls the POS-tagger and marks all verbs found as *VerbType* nodes in the content tree. To show the results in the wiki a yellow highlighting-renderer is attached to the node type *VerbType*.

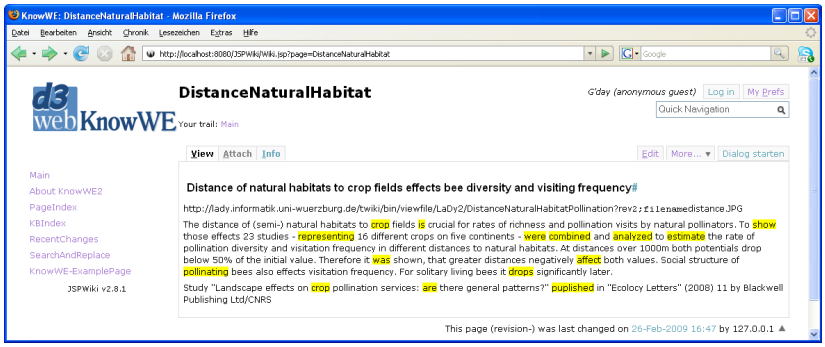


Fig. 7. The view of a wiki article with the POS-Tagging-Demo extension enabled.

This extension, consisting of a few lines of code calling a POS-Tagger library, enables to use the following tools coming along with KnowWE:

[7] http://nlp.stanford.edu/software/tagger.shtml

The annotation browser This tool shown in Figure 8 allows for browsing the wiki by annotation-types. For each annotation a link to the occurrence in the wiki page is provided. The column *ancestors* shows the types on the path of the content tree depicting the semantic context of the finding, which is of course *TaggingDemo* in this example. In this verb-tagging scenario this could be used to review the POS-tagging results manually. The browser can also be extended to support a two-step semi-automated workflow, where users can confirm or dismiss annotations to create verified meta-data. In general, this tool can serve as a statistical overview on formalized content.

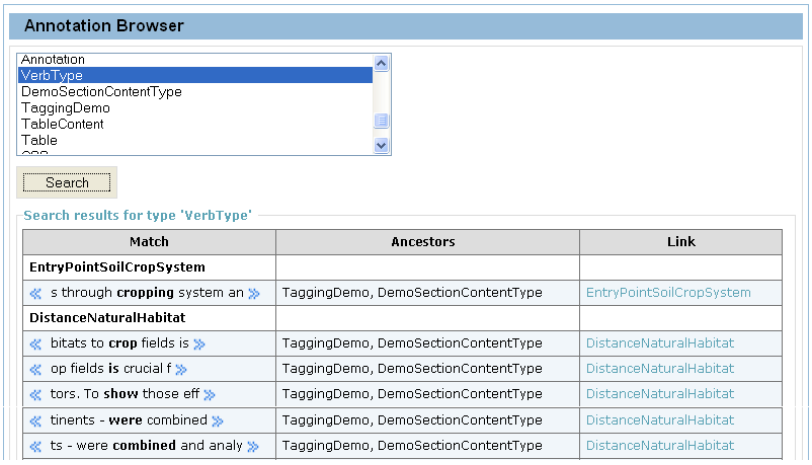


Fig. 8. Annotation browser listing all existing annotations in the wiki for a selected type

The semantic renaming tool The semantic annotation given to words by the POS-Tagger can be used for text refactoring. The semantic renaming tool is basically a standard global search and replace tool but additionally enabling semantic type-filtering. Thus, we can globally rename a string, but only if the string has a certain "role" in a particular markup, that is *VerbType* in this example. One use case which is enabled by this simple single-dimensional extension (Formalization) is that one could start some disambiguation efforts on the given content (e.g., for further processing by other NLP-techniques). Since this article is about crop fields we might like to rename the verb "crop" to a synonym like "cut" or "truncate" ensuring no confusion between the verb "crop" and the noun "crop" can arise. The renaming tool allows for sorting and filtering the replacements of the searched string by annotation-type and by articles. One can search for the string "crop" and all occurrences in the wiki are listed with annotation-context. Thus, one can select to replace all occurrences that are annotated as

VerbType in a subset of Wiki articles. In this way all nouns of "crop" stay untouched - this of course only works assuming that the POS-Tagger successfully separated the verbs from the nouns.

This semantic renaming tool working on all installed formalization extensions of a KnowWE system forms a basic refactoring tool for knowledge at different degrees of formality tagged by various formalization techniques.

4 Discussion

In this paper we motivated and discussed the concept of an extensible semantic wiki architecture. As an example implementation we introduced our extensible semantic wiki KnowWE and presented some of its current extensions. In our work we focus on supporting complex markups and the interconnection of formalized and textual content. The support of refactoring methods on the semi-formalized contents is the subject of our current research. The probably most popular semantic wiki Semantic MediaWiki shows numerous extensions on formalization and presentation. It is employing the "Decoration" pattern in different applications by the use of extensions like *semantic templates*, *semantic forms* and *semantic queries*. We think, that project-oriented customization of the tools towards the needs of the targeted domain and user community will become a more and more important challenge in the future. To demonstrate and evaluate our approach on this task we presented some case studies. Beside the HermesWiki project we are also applying this approach to the *BIOLOG* [8] project in order to optimally support the management of biological knowledge collected there. In this project we plan to use KnowWE to develop methods for semi-automated knowledge formalization using the agile employment of ontology learning methods. Considered from the other side, for researchers developing such ontology learning methods semantic wikis are attractive as an evaluation platform if simple integration of these methods is possible. We hope, that extensibility in general will reduce the setup-costs of semantic wiki solutions and therefore help to further establish this technology.

We will drive the KnowWE implementation towards even more flexibility and stability. For the latest news we refer to the project page of KnowWE on sourceforge[9].

References

1. Schaffert, S., Gruber, A., Westenthaler, R.: A semantic wiki for collaborative knowledge formation. In: Proceedings of SEMANTICS 2005 Conference, Trauner Verlag (2006)
2. Krötzsch, M., Vrandecić, D., Völkel, M.: Semantic MediaWiki. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273, Berlin, Springer (2006) 935–942

[8] www.biolog-europe.org

[9] http://sourceforge.net/projects/knowwe/

3. Schaffert, S.: IkeWiki: A semantic wiki for collaborative knowledge management. In: STICA'06: 1st International Workshop on Semantic Technologies in Collaborative Applications, Manchester, UK (2006)
4. Buffa, M., Gandon, F., Ereteo, G., Sander, P., Faron, C.: : A semantic wiki. Web Semantics **8**(1) (2008) 84–97
5. Auer, S., Dietzold, S., Riechert, T.: OntoWiki – A Tool for Social, Semantic Collaboration. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, Berlin, Springer (2006) 736–749
6. Kuhn, T.: Combining Semantic Wikis and Controlled Natural Language. In Bizer, C., Joshi, A., eds.: Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008). Volume 401., CEUR Workshop Proceedings (2008)
7. Lange, C., Kohlhase, M.: A semantic wiki for mathematical knowledge management. In Völkel, M., Schaffert, S., eds.: Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics. Workshop on Semantic Wikis, ESWC2006 (June 2006)
8. Nalepa, G.J., Wojnicki, I.: Proposal of a prolog-based knowledge wiki. In Nalepa, G.J., Baumeister, J., eds.: KESE. Volume 425 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
9. Baumeister, J., Puppe, F.: Web-based Knowledge Engineering using Knowledge Wikis. In: Proceedings of Symbiotic Relationships between Semantic Web and Knowledge Engineering (AAAI 2008 Spring Symposium). (2008)
10. Wagner, C.: Breaking the knowledge acquisition bottleneck through conversational knowledge management. Information Resources Management Journal **19**(1) (2006) 70–83
11. Kaljurand, K.: ACE View — an ontology and rule editor based on Attempto Controlled English. In: 5th OWL Experiences and Directions Workshop (OWLED 2008), Karlsruhe, Germany (26–27 October 2008)
12. Krötzsch, M., Schaffert, S., Vrandecic, D.: Reasoning in semantic wikis. In: Reasoning Web. (2007) 310–329
13. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), Edinburgh, Scotland (August 2005)
14. Ma, Y., Hitzler, P., Lin, Z.: Algorithms for paraconsistent reasoning with owl. In Franconi, E., Kifer, M., May, W., eds.: The Semantic Web: Research and Applications. Proceedings of the 4th European Semantic Web Conference, ESWC2007, Innsbruck, Austria, June 2007. Volume 4519 of Lecture Notes in Computer Science., Springer (JUN 2007) 399–413
15. Klinov, P., Parsia, B.: Pronto: Probabilistic ontological modeling in the semantic web. In: Proceedings of the Poster and Demonstration Session at the 5th European Semantic Web Conference (ESWC2008). Volume 401 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
16. Baumeister, J., Reutelshoefer, J., Puppe, F.: Markups for Knowledge Wikis. In: SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop, Whistler, Canada (2007) 7–14
17. Reutelshoefer, J., Baumeister, J., Puppe, F.: Ad-hoc knowledge engineering with semantic knowledge wikis. In: SemWiki'08: Proceedings of 3rd Semantic Wiki workshop - The Wiki Way of Semantics (CEUR Proceedings 360). (2008)

Knowledge Formalization in the Age of Wikis

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Jochen Reutelshöfer, Joachim Baumeister, Frank Puppe

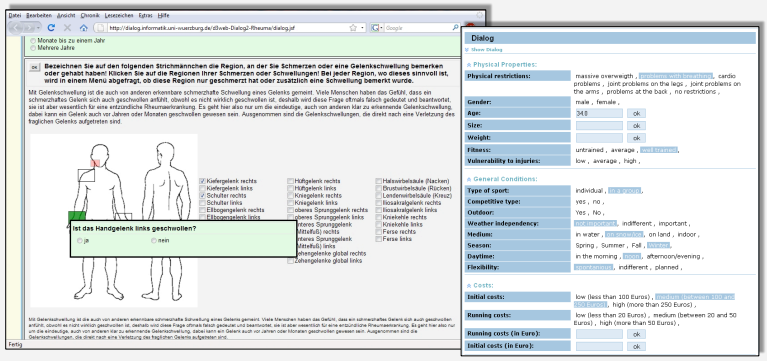
Overview

KnowWE is a semantic wiki designed for the integration of components for diagnostic problem-solving. To support the development of (diagnostic) knowledge systems, components for formalization (markup, editors), for reasoning (rules, decision-trees, fault models) and for executing/ testing have been implemented. The goal is to combine the strength of knowledge-based systems with the ease of Wikis to support collaborative knowledge engineering.

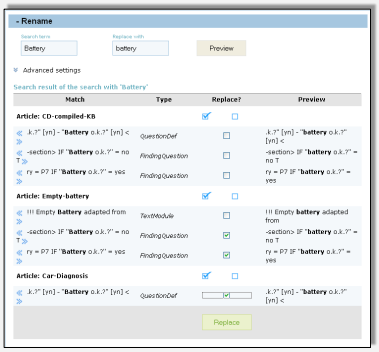
KnowWE

- Wiki-based workflow: collaborative fast and simple editing with immediate feedback.
- Combination of unformalized (startup documents) and (semi-) formalized knowledge.
- Problem-solving knowledge is instantly compiled into a knowledge base after saving a wiki page. It can be tested via starting test cases in user interviews.

User Interviews



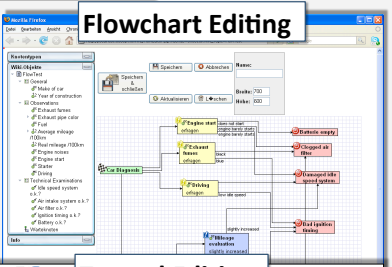
Refactoring



Markups and Editors

- Wiki editing (proprietary markup)
- Additional editors
  - Table/Inline-editing
  - Visual editors
- Syntax highlighting
- Verbose syntax-check
- Integrated refactoring tools
- Unit-test-cases
- Extensibility

Flowchart Editing



Textual Editing

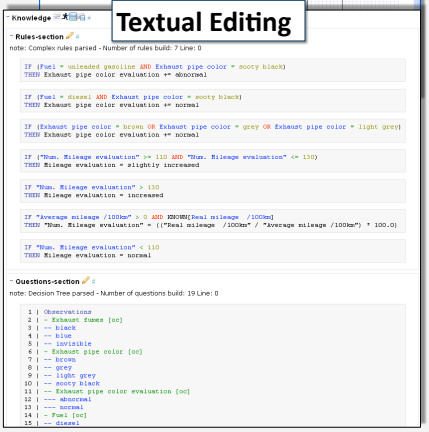
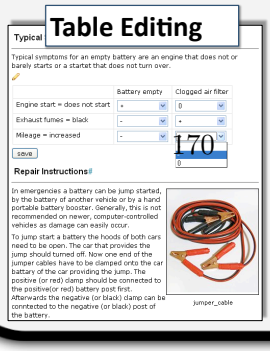


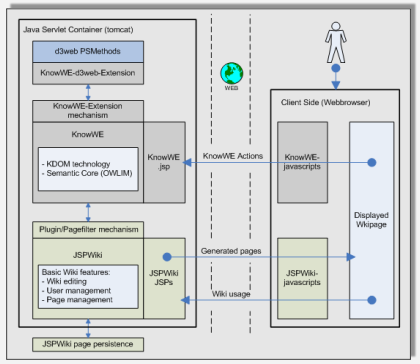
Table Editing



d3web Engine

- Open-source reasoning for diagnostic problem-solving, e.g. heuristic scoring rules, heuristic decision trees, set-covering fault models
- Implemented as KnowWE extension
- Compatible knowledge interchange with d3web.KnowME

Architecture



KiWi – A Platform for Semantic Social Software

Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, Stephanie Stroka

Salzburg Research Forschungsgesellschaft
Jakob Haringer Str. 5/II, A-5020 Salzburg, Austria
firstname.lastname@salzburgresearch.at

Abstract. Semantic Wikis have demonstrated the power of combining Wikis with Semantic Web technology. The KiWi system goes beyond Semantic Wikis by providing a flexible and adaptable platform for building different kinds of Social Semantic Software, powered by Semantic Web technology. This article describes the main functionalities and components of the KiWi system with respect to the user interface and to the system architecture. A particular focus is given to what we call "content versatility", i.e. the reuse of the same content in different kinds of social software applications. The article concludes with an overview of different applications we envision can be built on top of KiWi.

1 Motivation: From Semantic Wikis to KiWi

Semantic Wikis have been under investigation in the research community since 2005 (see e.g. [1,2]). Although there are many different systems with many different properties, a common trait between all Semantic Wikis is that they aim to combine ordinary wiki content and technology with Semantic Web technologies, either in order to provide better wikis ("Semantic Web for Wikis") or to ease the creation of Semantic Web data ("Wikis for the Semantic Web") [3]. Many also see Semantic Wikis as the "Semantic Web in a Nutshell", because – like Wikis show similar traits to the Web as a whole – Semantic Wikis share many properties and also problems with the envisioned Semantic Web.

The EU-funded project KiWi ("Knowledge in a Wiki")[1] takes the Semantic Wiki approach to the next level by providing a platform that allows to build many different kinds of Social Semantic Software, based on the conviction that most social software follows the "Wiki Principles". By "Wiki Principles", we mean that the term "Wiki" does not refer to technology alone. It is more a new philosophy of working with Web content influenced by ideas in the OpenSource community. These principles have revolutionised the way we work with content in the (Social) Web:

- **Wikis allow anyone to edit:** The core principle is that there are no access restrictions or strict hierarchies on the content of a wiki. Anyone can easily contribute his or her own knowledge, his or her own ideas, and his or her own content.

[1] http://www.kiwi-project.eu

– **Wikis are easy to use:** Anyone who is sufficiently familiar with the basic functionalities of word processing software (write, delete, save) has all the skills required to edit, correct and expand a wiki.
– **Wiki content is linkable:** By allowing users to create links between words and as such between concepts, wikis also allow for the creation of semantic relations, i.e. of meaning.
– **Wikis support versioning:** Never does information disappear on a wiki. If a page is edited, the previous version is still stored somewhere. This has an important psychological effect as it takes away the wiki writer's block: the fear that something might get lost through editing.
– **Wikis support all media:** Wikis are web-based. So whichever type of content you have, be it text, images, audio, spreadsheets, documents – anything that can be displayed in a web browser can be displayed in a wiki. And even if a file cannot be displayed in the browser itself, it can still be downloaded.

This same philosophy is underlying not only Wikis (technically-speaking), but also a large array of other Social Software applications: e.g., a weblog or social networking platform can be seen as just a different user interface (and different way of using), but is otherwise very similar concerning the underlying principles and also technology.

In the following, we describe how we realised the KiWi vision in the *KiWi System*, a generic Semantic Social Software platform based on the Wiki principles. We begin in Section 2 with introducing the concept of what we call "Content Versatility": content with flexible structures that can be reused in different applications. In Section 3, we then describe KiWi Core Applications: the Wiki, the Dashboard, TagIT, the KiWi Search, and the KiWi Inspector. Section 4 is dedicated to the more technical aspects of the KiWi platform: the system architecture, the data model, the KiWi entity manager, and façading. We conclude this article with an outlook and summary in Section 5.

2 Content Versatility: Same Content, Different Views

As we have already outlined in the introduction, what we call the "Wiki Principles" is actually applicable to many Social Software applications. It is therefore close at hand to build generic platforms for developing different kinds of Social Software. And indeed, there are several products already available that aim to deliver a platform that allows to combine wikis, weblogs, social networking, etc. Such platforms are for example Clearspace Community[2] from Jive Software, Community Server[3] from telligent, and Liferay Social Office[4] from Liferay.

However, all these systems only provide integration on the user interface level and still see wiki articles, blog posts, etc. as separate kinds of content that is visualised in a specific way. This has a number of serious disadvantages. For

[2] http://www.jivesoftware.com/products/clearspace-community
[3] http://communityserver.com/
[4] http://www.liferay.com/web/guest/products/social_office

instance, something that started as a wiki article can never become a blog post, it will never be possible to attach comments to a wiki article if not foreseen in the data model, and new kinds of content (like our TagIT locations, cf. Section 3.3) cannot be added easily without also doing fundamental changes to the system.

KiWi follows a completely different approach which we call "Content Versatility". The underlying principle is that every piece of information is a combination of human-readable content and associated metadata, and that the same piece of information can be presented to the user in many different forms: as a wiki page, as a blog post, as a comment to a blog, as a photo, or even in a bubble in a map-based application. The decision how the information is displayed is taken based on the metadata of the content, and the context of the content and the user (e.g. metadata, user preferences, different applications). Metadata is represented using RDF and thus does not require a-priori schema definitions, so the data model of the system can be extended even at runtime.

In KiWi, we call the smallest piece of information a "content item". A content item is identified by a URI and consists of human-readable content in the form of XHTML and associated metadata in the form of RDF. The KiWi core system provides means to store, update, search, and query content items, and offers automatic versioning of all updates (content and metadata). Whereas core properties of a content item (like the content, the author, and the creation date) are represented in XML and persisted in a relational database, all other properties can be flexibly defined using RDF properties or relations. The URI of a content item is generated in such a way that it is possible to make a KiWi system part of the Linked Open Data cloud [4]. This allows to easily integrate KiWi content with other services on the Semantic Web.

3 KiWi Core Applications

Content Versatility makes it possible to offer completely different views on the content inside the KiWi system without any modifications to the core system or data model. We call such views "KiWi Applications", and they are one kind of extension offered by the KiWi system (others are currently: KiWi Services, KiWi Widgets, KiWi Actions, and Exporters/Importers). In the following, we describe the set of applications that are part of the KiWi System to illustrate how the Wiki Principles and Content Versatility are realised in KiWi. The applications have been selected based on the requirements identified in the two KiWi use cases and accompanying SNML projects; additional applications are conceivable and reasonable. Additional applications will be offered as separate packages in later stages of the project.

KiWi applications share the same context, i.e. when the user browses a content item in the Wiki or Dashboard, she can switch to the TagIT application and view the same content item on a map or to the Inspector and get debugging information on the content item.

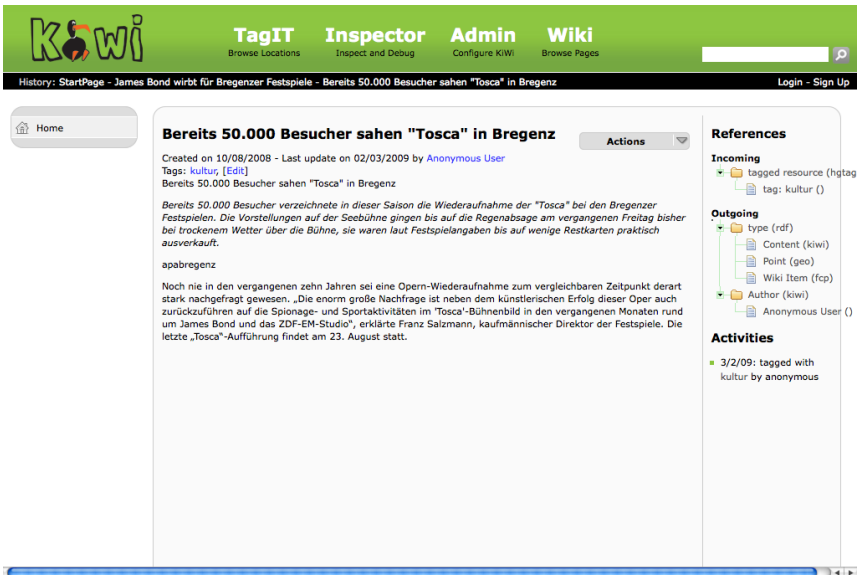


Fig. 1. The KiWi Wiki: A Semantic Wiki built on top of KiWi

3.1 The KiWi Wiki

The primary and most generic interface to the KiWi system is a Semantic Wiki. The layout and functionality (Figure 1) of the KiWi Wiki is inspired by its predecessor IkeWiki[5]: the left column offers navigation functionality, the centre column contains the main (human-readable) wiki content, and the right column contains dynamic widgets that display additional information about the content item based on its metadata (e.g. a map or incoming and outgoing links).

The centre column by default displays the content of the content item. The section below the page title contains maintenance information about the item as well as the list of tags that have been associated with it. By clicking on the "Action" menu in the top right corner, the user can select to edit the content, edit the metadata (i.e., OWL Datatype Properties), edit the relations (i.e., OWL Object Properties) to other content items, edit the list of tags, and access the history of the item's content and metadata as provided by the versioning subsystem. In principle, the KiWi Wiki interface can thus be used in the same way as IkeWiki. Additional actions can be registered using KiWi's extension mechanism (e.g. domain specific actions like "Geolocate" or "Share").

The widgets in the right column are visually part of the content box to emphasise that they contain additional information about the currently displayed item. Currently, the KiWi system offers to display the list of references (based on RDF relations with other items) and a "Stream of Activities" listing the

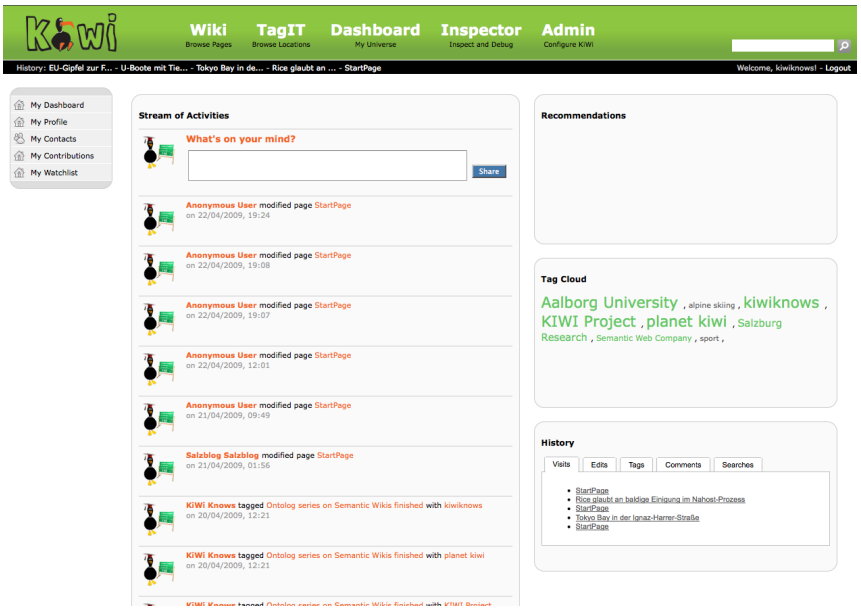


Fig. 2. KiWi Dashboard: personal user page with stream of activities, history, personal tag cloud, social networking, and recommendations

recent activities associated with the content item (e.g. modification, tagging, annotation).

3.2 The Dashboard

The Dashboard is a user's personal(ized) start page in the KiWi system. Figure 2 shows an early stage of its user interface, which is currently still under development. The Dashboard follows the same general layout as the Wiki, i.e. the left column provides generic navigation while the centre and right columns contain the actual content. While the look of the Dashboard is freely customisable by the user, the KiWi core system by default provides the following information:

- the *Stream of Activities* is the most important part of the Dashboard: it contains an aggregated list of activities that happened in the user's "universe", i.e. updates to content items that are either explicitly watched by the user or implicitly added to the user's universe e.g. because they have been edited by the user, because they have been edited or rated "good" by one of the "friends" of the user, or because they have been recommended to the user based on previous activites by the personalisation component of KiWi
- the *Recommendations* widget provides a list of additional content items that might be relevant to the user; different recommendation algorithms are investigated as part of the KiWi project [6]

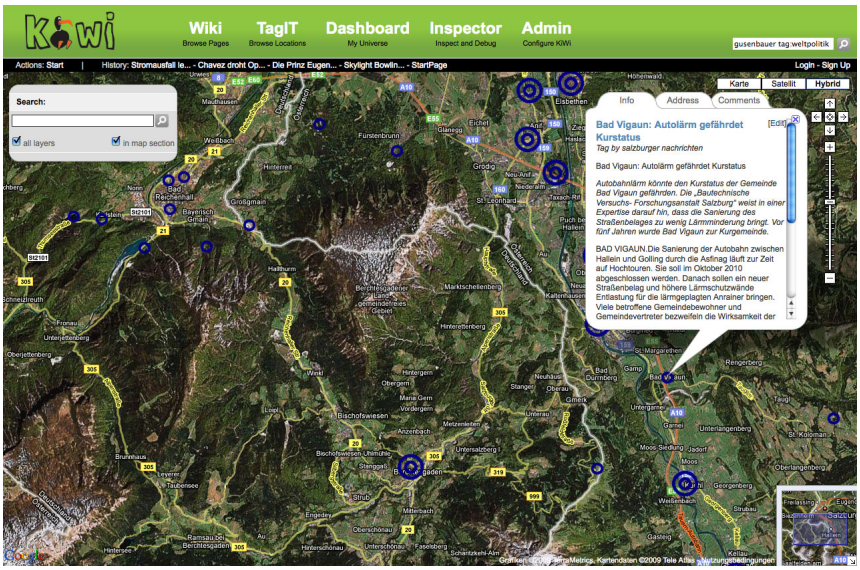


Fig. 3. TagIT: description of locations / geolocation of content; double circles indicate clusters of points, single circles single points of interest

- the *History* widget lists the content items the user visited or worked on recently to give quick access to the issues the user is currently concerned with
- the *Tags* widget lists the tags used by the user and is a quick and flexible means of structuring and accessing the content items that are relevant to the user; clicking on a tag redirects to the search interface described below

Besides the main view, the Dashboard is also the place where the user manages his own profile. The most important part of the user's profile is the list of friends, which is the primary way to use the social networking functionality of KiWi. The requirements analysis carried out as part of the KiWi use cases showed that social networking is a crucial aspect in a modern knowledge management system like KiWi as it helps define the context the user works in.

3.3 TagIT

TagIT is an application originally developed in a separate project with the goal to create the "Youth Atlas of Salzburg"[5], which has been running as a prototype successfully for over a year and has now been ported to the KiWi platform (Figure 3). In TagIT, users browse through a map (based on Google Maps) where they can "tag" locations with descriptions and provide interesting information

[5] http://tagit.salzburgresearch.at

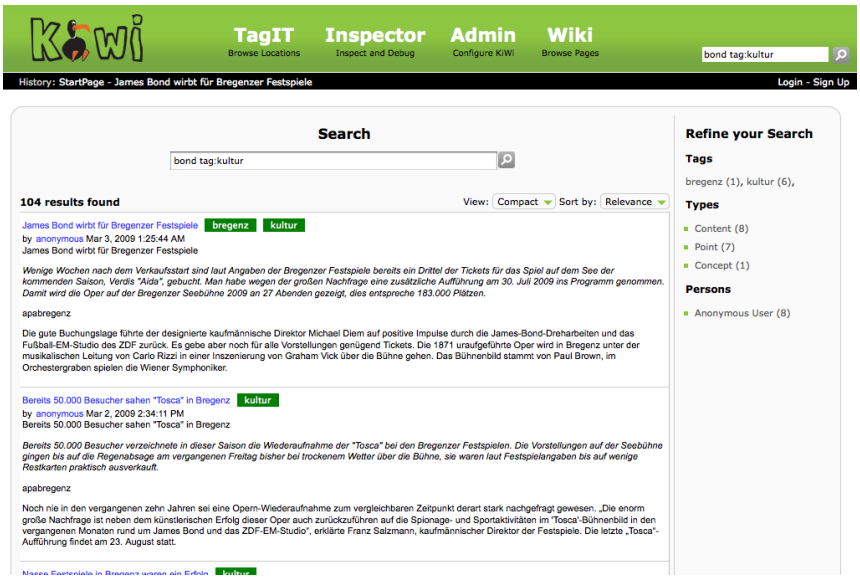


Fig. 4. KiWi Search: generic, facetted search over KiWi content

about them, e.g. cafés, bars, sports parks, hiking tours, etc. Such tags can be associated with categories from a SKOS thesaurus and with additional multimedia data like photos or videos. Other users can then browse or search for interesting locations using the same interface. In addition to the web-based platform, TagIT also offers a mobile client that can run on a GPS-enabled mobile phone. When visiting an interesting location, users can then start the TagIT application, take a picture of the location, add a short description and directly upload the "tag" using UMTS. The tag is automatically geolocated and immediately available for other users.

Although quite different on the user interface and in the way it is used, TagIT actually closely follows the wiki principles: everyone can add and edit tags, the system is easy to use, tags can be linked, tags are versioned, and different kinds of content are supported. On top of KiWi, tags are realised as content items and can thus be displayed in both the TagIT user interface and the previously described KiWi Wiki (in which case a small map widget is displayed in the right column showing the position).

3.4 KiWi Search

In addition to these different ways of presenting content, the KiWi core system also provides a generic search functionality accessible from within all KiWi applications. When a user switches to search and selects a content item, he is redirected back to the previous application afterwards. KiWi currently allows

a combination of full-text search, metadata search (tags, types, persons), and database search (date, title). A more sophisticated search language is currently under development [7].

The KiWi search interface implements a so-called "facetted search" (see Figure 4): the user starts with a keyword search, resulting in a list of content items ordered by relevance or time. In case the user is not satisfied with the results, he then has the option to refine his search using one or more of the facets offered in the right column of the search result box. Currently, the KiWi system offers the facets "tags", "types", and "persons", which we have identified as the core facets needed in any system. For each facet, only the criteria occurring in the currently displayed search results are listed, together with a count of the content items matching the criterion. Selecting one of the criteria narrows down the search.

All search facets are included in the full-text search box; this decision has been made to provide all search functionality in one place without confusing the user and to allow advanced users to directly search using the text field. Also, it makes it much simpler to bookmark searches or include them in a user's personal stream of activities on the Dashboard.

In later stages of the project, it is planned to make the set of widgets customisable to adapt it to different application domains. For example, in a biology scenario, an interesting facet could be different protein structures, in a music scenario it could be different instruments, and in a history scenario it could be different countries.

3.5 KiWi Inspector

The KiWi Inspector is an application developed for advanced users and developers. It provides a more technical insight into the current context. It currently provides the following functionalities:

- *Content Item Inspector*: displays the RDF data associated with the current content item as RDF/XML; the shown RDF data is the same as would be displayed to a Linked Open Data client when accessing the KiWi system
- *Tag Inspector*: displays a list of all tagging actions of the current content item and the RDF data generated by them as RDF/XML
- *User Inspector*: displays the RDF data associated with the currently logged in user as RDF/XML

In addition, the KiWi Inspector will later also provide more details about the revisions (particularly metadata revisions) of the current content item and additional debugging information as needed.

4 The KiWi Platform

The applications described in the previous section are built on top of the KiWi Platform, which provides all the core functionalities needed by most Semantic Social Software applications. In the following, we briefly describe architecture, core data model, and core services offered by the KiWi platform.

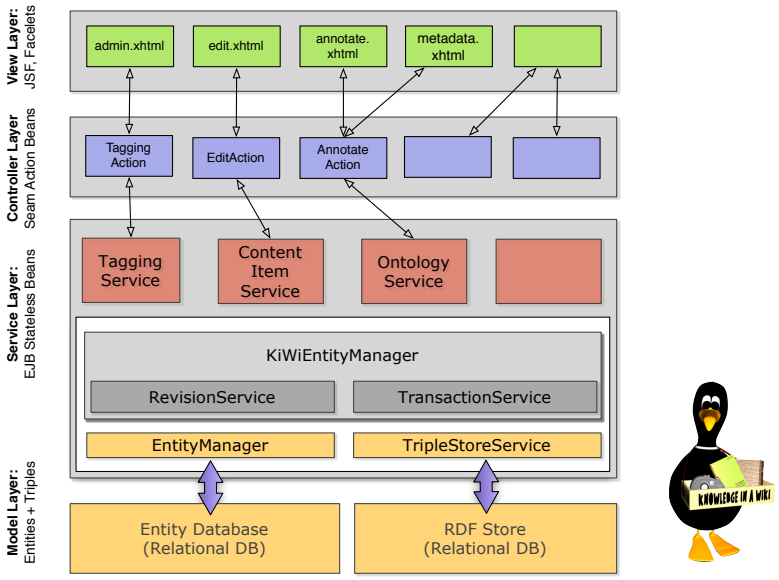


Fig. 5. KiWi Service-Oriented Architecture: Model Layer, Service Layer, Controller Layer, View Layer

4.1 Architecture: Service-Oriented and Component-Based

The KiWi system is implemented on top of JBoss Seam[6] and Java Enterprise Edition (Java EE 5), and thus follows a component- and service-oriented architecture. Figure 5 depicts the overall structure of the KiWi system:

The *model layer* comprises the KiWi data model (see below) and is represented in a relational database, a triple store, and a full-text index. Entities are persisted using the Hibernate framework[7], which maps Java objects to relational tables. The KiWi triple store is a custom implementation also based on the relational database, because existing triple store implementations provide insufficient support for features like versioning and additional metadata about triples that are needed by KiWi. The full-text index is implemented using Hibernate Search and currently allows to search over title, textual content, tags, authors, and RDF literals.

The *service layer* provides services to other components in the KiWi system. Of central importance is the KiWi Entity Manager, which provides unified access to content items and RDF metadata (see below). Further core services are the revision service – taking care of versioning, and the transaction service,

[6] http://www.seamframework.org
[7] http://www.hibernate.org

allowing to manage all updates to KiWi content in reliable transactions. Both services are heavily used internally by the KiWi Entity Manager and usually not used by further components. Besides these core services, the service layer may contain additional services that offer certain common functionalities. For example, the KiWi system currently offers an "ontology service" that provides convenient access to the triple store using higher-level concepts like "classes" and "properties", and a "content item service" that allows to easily access all functionalities associated with content items (creating, loading, updating).

The *controller layer* consists of action components that implement a specific functionality in the KiWi system. For example, the Semantic Wiki application contains a "view action", a "edit action", a "annotation action", and a "history action", and the TagIT application contains a "explorer action" and a "tagging action". Action components are usually very close to some functionality offered in the user interface, and they make use of service components to access the content in the KiWi system.

The *view layer* is implemented using Java Server Faces (JSF), which are used to generate the HTML presentation of the KiWi user interface and the user interaction with the system. JSF pages are linked with action components in the controller layer. Also part of the view layer are web services offered by KiWi. Currently, there are web services for accessing the triple store and SKOS thesauruses, and there is a "linked open data" service offering the content of the KiWi system to linked open data clients.

4.2 Data Model: ContentItems, Tags, and Triples

KiWi's core data model makes use of few concepts, namely Content Items, Tags, and Triples. Additional functionality is added by "KiWi Façades" (Section 4.3).

Content Items. The *content item* is the core concept of the KiWi system. It represents a "unit of information" in KiWi, e.g. a page about a certain topic, a user profile, etc. When a user accesses the KiWi system, he is always interacting with exactly one (primary) content item, the context content item. The context content item can be viewed, modified, and annotated by the user. Though changes might also affect other content items, the context content item is always the primary content item.

Each content item has both, a machine readable symbolic representation and a human readable textual or multimedia representation.

- content items are all different kinds of content and data items that are stored in the KiWi system, i.e. (wiki) pages, multimedia, users, roles, rule definitions, layout definitions, widgets, and possibly more. The KiWi system is not restricted a priori to specific content formats.
- URIs or blank nodes serve as machine-readable symbolic representations of resources, to be used in extended triples in the triple store. The URI is used to embed a resource in its context and provide machine-readable meaning, e.g. by annotation with formal annotations, reasoning, etc.

– the textual / media content related to a resource is meant for human consumption. The content resembles a wiki page, is easy to edit, supports linking, and is versioned. In the current design each wiki article in a specific language is represented as an own content item describing a single concept. The assumption for this design is that the content about a topic and its authors may differ from language to language. The definition of connections between content items with equivalent content but different languages can be accomplished with metadata relations.

These assumptions distinguish the KiWi data model from other wikis and content management systems. Most important, it treats all kinds of resources equally, leading to a very clean and simple model where every resource has both, a machine-readable and a human-readable description. A consequence of the direct relationship between content items and RDF resources is that every RDF resource, even those representing widgets, layouts, users, or even rules, can also be described in human readable form.

Textual content is represented internally as (structured) XML documents that can be queried and transformed to other representations like HTML or XSL-FO (for PDF and other print formats) using standard XML query languages (XQuery, XSLT) or using the rule-based reasoning language developed in KiWi. The XML format used for page representations resembles a subset of HTML, taking into account only core structuring elements.

Tags. Tagging is one of two ways of annotating content items in the KiWi system. In KiWi, tags serve many different purposes, for example associating content items with certain topics or grouping content items in "knowledge spaces". There are two kinds of tags: explicit tags are explicitly added to a content item by a user; implicit tags are created by the system, e.g. based on automatic mechanisms like information extraction from text or reasoning on existing tags.

Conceptually, an explicit tag is a 3-ary relation between two content items (the tagged content item and the tagging content item) and a user (the tagging user or tagger). An implicit tag is a binary relation between two content items, a tagged and a tagging one. Tagging content items are identified using one or more labels that are available for annotating content items. In case of ambiguous tag labels (i.e. the same tag label for different content items), the KiWi system asks the user to choose the appropriate content item. If the user enters a new label that is not yet used elsewhere, it is displayed like a wiki-link to a non-existing page; when the user clicks on it, he is given the choice to either associate the label with an existing content item or to create a new content item explaining this tag label. Internally, a tag is furthermore given maintenance information like creation time and date and a URI for uniquely identifying a tag.

For example, the content item that describes "Mickey Mouse" could be tagged with the label "Mouse", thereby associating it with the content item describing "Mouse" (the animal). The tagged content item would be "Mickey Mouse", the tagging content item would be "Mouse", and the tag label used for tagging would be "Mouse", which is a tag label of the content item "Mouse".

Inside the system, a tag is mapped to an RDF structure that can be used for deriving additional RDF metadata by means of reasoning. Also, tags can be "lifted" to taxonomy or ontology concepts by advanced users, e.g. by using the "meaning of a tag" (MOAT)[8] or "social semantic cloud of tags" (SCOT)[9] ontologies. In this case, more information about the meaning and context of a tag becomes available, e.g. for reasoning or querying.

Tags can be used by the KiWi system for many different purposes. For example, tags can help with searching by offering a facetted search interface or by offering tag clouds. Furthermore, it is possible to derive user preferences from the tags she has used or to identify users with similar interests via clustering. Similarly, tags can also be used for grouping related content items, e.g. for defining group work spaces or for clustering thematically related items. Beyond that, the way how tags are used is left to the application developers and users that implement a specific instance of the KiWi system.

Extended Triples. Machine-readable metadata is represented using what we call extended triples. Extended triples contain additional maintenance information that is used internally by the KiWi system for various tasks like versioning, transactions, associating a triple with a certain workspace, user, or group, or for reason maintenance (i.e. storing why a certain triple has been asserted). In principle, an extended triple can thus be seen as a "triple with attributes".

Note that these attributes could also be represented as a RDF subgraph using triples and reification. However, such a representation has several disadvantages compared to the extended triples proposed for KiWi:

- it requires reification, meaning that the original triple, which assumingly provides the most interesting information, is broken up into parts that have to be reassembled, and
- it mixes up several levels of abstraction, which is inconvenient not only for machines and reasoning, but also for the user
- it makes it difficult to filter out the information that is used for internal purposes and this not supposed to be exchanged with external systems

The implementation of extended triples is straightforward and fits easily with already existing tools and standards without disguising the original meaning. Triple attributes containing maintenance information are only represented programmatically inside the system, avoiding problematic situations. To the outside world, extended triples look like ordinary triples and can be exported into the usual Semantic Web formats like RDF.

In the current implementation, extended triples are represented in special tables in the relational database, and queries to the triple store are executed in Hibernate's object oriented query language HQL. We chose not to use one of the existing triple store implementations because they are restricted to simple triples

[8] http://moat-project.org/
[9] http://scot-project.org/scot/

without the possibility to add additional metadata, they have only basic transaction support [8], and they offer poor scalability if one wants to use reasoning. Building KiWi on top of these systems has proven to be extremely difficult and has been abandoned in favour of the more flexible database solution. Mapping SPARQL queries to Hibernate is currently under development.

4.3 KiWi Entity Manager and KiWi Façades: Content Versatility in Java EE

The KiWi system offers a number of core services that are needed in many situations. Of particular importance is the KiWi entity manager (named in analogy with the Java EE entity manager, as it provides the technological background for realising content versatility. In the following, we briefly introduce its functionality and then discuss a technique we call KiWi Façades.

KiWi Entity Manager. The KiWi entity manager is a central service providing unified access to data stored in the relational database, in the triple store, and in the fulltext index. It provides functionalities for storing, querying and searching entities (content items, triples, tags) in different languages:

- *Storing* of entities is handled through the methods `persist()` (for new entities) and `merge()` (for updated entities). Both methods take care of forking the data associated with a Java entity appropriately into relational database, triple store, and fulltext index.
- *Querying* of entities is handled by the method `createQuery()`, which takes as argument a query string in either HQL (Hibernate's object oriented query language) or SPARQL and returns a Query object that can be used in the same way as the ordinary Java EE Query object for retrieving results.
- *Searching* of entities is handled by the method `search()`, taking as argument a label-keyword search string and then delegates – depending on the label – to either the fulltext index, the relational database, or the triple store.

All KiWi Entity Manager methods support façading, described below. Additionally, all updates performed through the KiWi entity manager are automatically wrapped in appropriate transactions that support transaction isolation and commit/rollback functionality. Also, KiWi entity manager transactions are automatically versioned using KiWi's revision service and can be reverted individually. The KiWi transaction system is discussed in [8].

KiWi Façades. A particularly salient aspect of the KiWi system is its façading mechanism. Façading can be seen as a way of providing different application-dependent Java views on content items. They are thus a way of realising content versatility in Java in a way natural to developers. A KiWi Façade is specified as a Java interface, annotated with Java 5 annotations that map Java methods to RDF properties (see Figure 6). When calling an annotated method, an appropriate query to the triple store is issued instead of accessing the Java field.

```
@KiWiFacade
@RDFType( Constants.NS_GEO+"Point" )
public interface PointOfInterestFacade extends ContentItemI {

    @RDF(Constants.NS_GEO+"long")
    public double getLongitude();

    public void setLongitude(double longitude);

    @RDF(Constants.NS_GEO+"lat")
    public double getLatitude();

    public void setLatitude(double latitude);
}
```

Fig. 6. A Java interface annotated as KiWi Façade; the methods get/setLongitude() map to the RDF property `geo:long`, whereas get/setLatitude() map to `geo:lat`. A content item façaded with this interface is automatically assigned the `geo:Point` type.

For a Java developer working with the system, a façaded content item behaves exactly like an ordinary content item with the additional methods specified in the façade interface, and can be used in any context a content item can be used, e.g. as backing bean for user interface components. This functionality is realised using Java's dynamic proxy mechanism (implemented in the KiWi invocation handler). All methods in the KiWi entity manager may optionally take or return a KiWi Façade instead of a content item if they are passed a façade interface as additional argument.

5 Outlook and Conclusion

KiWi is an ongoing research project of which we have only demonstrated the first results. Many more salient aspects are still to come. Particularly, KiWi will be extended with "enabling technologies" in the following areas:

- **Reasoning and Reason Maintenance:** in this area, the KiWi system will be extended with a custom, rule-based querying and reasoning component where advanced users will be able to add custom inference rules to the knowledge base; reasoning will be based on content as well as metadata [9]. In addition, there will be a reason maintenance component that allows users to get explanations why a certain information has been inferred; reason maintenance is also beneficial for update performance and might even allow for different users having different rule sets.
- **Information Extraction:** the KiWi system will furthermore provide a component that semi-automatically extracts metadata from the content that is

stored in the knowledge base. Information Extraction will be performed in interaction with the user to minimise the number of errors.
- **Personalisation:** based on the metadata for a content item, a user model, and the rule-based reasoning, KiWi will also offer a personalisation component that allows to further customise the presentation of a content item. The personalisation component will further demonstrate the flexibility of the Content Versatility approach taken by KiWi.

TagIT is now further developed as part of the Salzburg NewMediaLab project "Future Content Platforms" together with our partner *Salzburger Nachrichten*. The goal is to integrate in the same interface not only wiki content and TagIT locations but also news articles, blog posts, and small advertisements from our partner. For the purpose of the demonstration, we have already imported 20.000 online news articles, but the system is designed to scale to hundreds of thousands.

Acknowledgement. This research has been partly funded by Salzburg New Media Lab and by the the European Commission within the 7th Framework Programme project KiWi - Knowledge in a Wiki (No. 211932). The latest KiWi source code is available as Open Source at the KiWi website http://www.kiwi-project.eu. The demonstration system is published from time to time at http://showcase.kiwi-project.eu.

References

1. Völkel, M., Schaffert, S., eds.: 1st Workshop "From Wiki to Semantics" (SemWiki'06) – colocated with ESWC'06, Budva, Montenegro (2006)
2. Lange, C., Schaffert, S., Skaf-Molli, H., Völkel, M., eds.: 3rd Workshop "From Wiki to Semantics" (SemWiki'08) – colocated with ESWC'08, Tenerife, Spain (2008)
3. Schaffert, S.: Semantic Social Software: Semantically Enabled Social Software or Socially Enabled Semantic Web? In: Semantics 2006, Vienna, Austria (2006)
4. Bizer, C., Heath, T., Ayers, D., Raimond, Y.: Interlinking Open Data on the Web. In: 4th European Semantic Web Conference (ESWC2007) – Posters Track, Innsbruck, Austria (2007)
5. Schaffert, S., Westenthaler, R., Gruber, A.: IkeWiki: A User-Friendly Semantic Wiki. In: Proceedings of the 3rd European Semantic Web Conference (ESWC06) – Demonstrations Track, Budva, Montenegro (2006)
6. Durao, F., Dolog, P.: Analysis of tag-based recommendation performance for a semantic wiki. In: Fourth Workshop on Semantic Wikis (SemWiki2009) in conjunction with the 6th Annual European Semantic Web Conference (ESWC2009), Heraklion, Greece, CEUR.org (2009)
7. Weiand, K., Furche, T., Bry, F.: Quo vadis, web queries. In: Proceedings of International Workshop on Semantic Web Technologies, Belgrade, Serbia (29th–30th September 2008). (2008)
8. Stroka, S.: Transaction management in federated, heterogeneous database systems for semantic social software applications. In: submitted to 20th Int. Conference on Database and Expert Systems Applications (DEXA'09), Linz, Austria (2009)
9. Bry, F., Kotowski, J.: Towards reasoning and explanations for social tagging. In: Proceedings of 3rd International Workshop on Explanation-aware Computing, Patras, Greece (21st–22nd July 2008). (2008)

VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis

Mariano Rico[1], David Camacho[1], Óscar Corcho[2]

[1] Computer Science Department, Universidad Autónoma de Madrid, Spain*
{mariano.rico, david.camacho}@uam.es
[2] Ontology Engineering Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es

Abstract. We describe VPOET templates, web templates to present or request semantic data to or from end users. These templates can be used by web developers with no competencies in Semantic Web by means of simple HTTP messages. Although its application to wiki environments is only pointed, a real application oriented to end users is shown. This experience shows that VPOET templates can be easily integrated in web applications written in any programming language.

1 Introduction

Besides semantic annotation [1], some general-purpose wikis also provide support to create semantic web applications [2,3]. VPOET [3] [4] is one of these semantic web application, aimed at web designers without knowledge in Semantic Web. It allows them to create web templates designed to display semantic data (output templates) or to request information from the user to convert it into semantic data (input templates).

All the information is stored as semantic data that can be recovered through simple HTTP messages, in a much simpler way than SPARQL endpoints (which require SPARQL language skills to create the HTTP message), aimed at lowering the adoption barrier for common web developers. An additional benefit of using HTTP messages, compared to traditional programming libraries, is that it can be exploited by developers in any programming language, expanding further the developers scope.

As a usage example for testing how a common developer can exploit the semantic information stored in VPOET, we have created a Google Gadget that allows users with minimal knowledge of Web technologies (end users) to incorporate semantic information in their web sites and in applications such as Google Pages, Google Desktop, etc. This example shows that VPOET templates can be

*This work has been partially funded under the projects HADA (TIN2007-64718), METEORIC (TIN2008-02081), and DEDICON (TIC-4425)

[3] See http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOET

easily integrated into any web application. In a wiki framework, a simple command (in "edition mode") with few arguments, can be used to render (in "view mode") semantic data, allowing users browse through the semantic relations.

2 Related work

The way in which templates can help users to homogenize semantic data presentation has been addressed before. Semantic Media Wiki allows users creating templates, but employs an intricate syntax and parsing functions [4]. Therefore, it is not easy to handle by a web designer, usually with lower competences in programming languages.

A template infrastructure for semantic data is Fresnel [5], used by Longwell, a faceted semantic web browser. However the Fresnel syntax [5] requires skills in semantic web technologies that cannot be accomplished by most web designers.

Rhizomer [6] is an infrastructure to browse and edit semantic data. The presentation of RDF data is achieved by means of XSLT. The competences required to create Rhizomer templates are not aimed at web designers either.

Therefore, the current state of the art does not provide web designers with authoring tools to create attractive an reusable templates for semantic data. There must be a balance between expressiveness, to address RDF features (e.g. multi-valued properties, properties with no value) and complexity, in order to reduce the required level of competencies. VPOET provides web designers with OMEMO[6], another application that generates simplified versions of a given ontology, specifically oriented to web designers. By reading the information provided by OMEMO, web designers can know the sub-components of a given ontology component, requiring only basics of semantic web technologies such as class, property, value or relation. Details such as restrictions, inverse functional properties, etc. are hidden on purpose.

3 VPOET as a templates source

VPOET has two faces, on the one hand it is a web application oriented to web designers ranging from amateur users to professional ones. Following a 20 min. online tutorial [7] is enough to start creating templates embedding simple macros in the client-side code (HTML, CSS, or Javascript) generated by the web designer favorite authoring tool (e.g. Dreamweaver). These macros are detailed in the aforementioned tutorial. Figure 1 shows an output template for `FOAF:Person`. Macros are framed within a thick rectangle (`OmemoGetP`, `OmemoConditionalVizFor`, among others), and reused templates are framed

[4] A template source code and usage example can be found at `http://en.wikipedia.org/wiki/Template:Infobox_Settlement`

[5] See an example at `http://www.w3.org/2005/04/fresnel-info/manual/`

[6] See `http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=OMEMO`

[7] See `http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOETTutorial`

Fig. 1. Template example showing macros (thick rectangles) and templates reuse (thin rectangles).

within a thin stroke. It must be noticed that, unlike other templates systems, the code is essentially client-side code, substituting programming issues by simple macros, easier to understand by web designers. These macros allow a variable number of arguments and can be evaluated in design time, warning web designers when some of the arguments is wrong.

On the other hand, it is a semantic data source fed by the templates created by a community of web designers sharing and reusing templates. This source can be exploited easily by common web developers, in any programming language, by means of HTTP messages (GET and POST) like "render the semantic data at URL Z by using the output template X created by designer Y", codified as a HTTP GET message by means of the following URL:

```
http://URL-to-servlet/VPoetRequestServlet?action=renderOutput
&designID=X&provider=Y&source=Z.
```

An additional argument can specify a given individual in the source. In this case, only the individual is rendered. The full syntax of these HTTP messages can be found in the aforementioned tutorial.

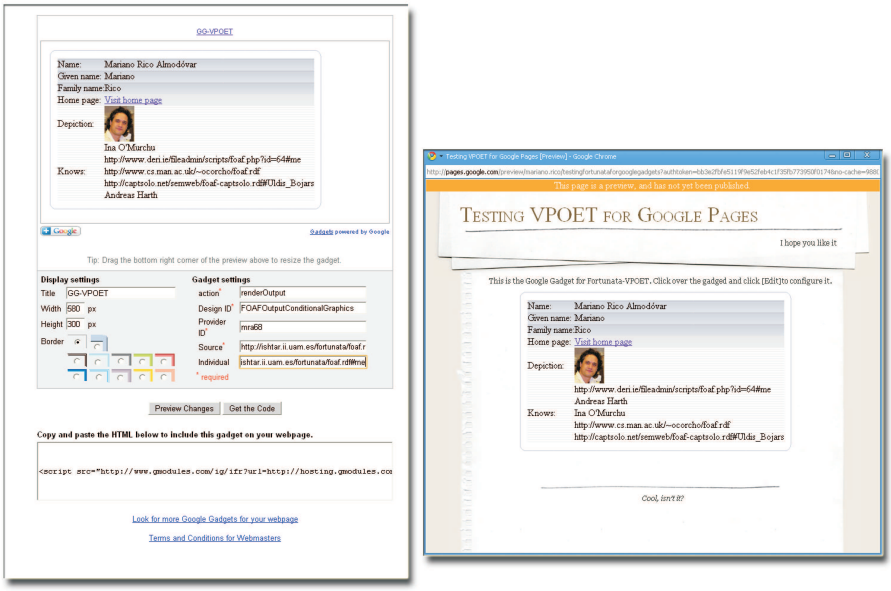


Fig. 2. Using GG-VPOET. Left: facilities to supply the HTML code that must be inserted by end users into a regular web page. Right: using this gadget in end user oriented application (Google Pages).

4 Using VPOET templates in wiki pages

Exploiting VPOET templates by means of the aforementioned HTTP messages is easy. As an usage example, a Google Gadget named GG-VPOET [8] has been created. By using this gadget, any end-user can render a semantic data source (by means of an output template) or provide a web interface to create semantic data (by means of an input template). GG-VPOET, as any other Google Gadget, can be inserted into a regular web page or in Google products such as iGoogle, Google Desktop or Google Pages. Figure 2 shows this gadget in action, rendering a given data source by means of the template shown in figure 1.

Following the same principles, the integration of VPOET templates in a given wiki framework (let us say JSPWiki) depends on the existence of a mechanism to add new tags in the wiki syntax and associate functionality to the tag. For example, JSPWIki implements these features by means of plugins, and getting the renderization shown in figure 2 (right side) is as easy as creating a JSPWiki plugin (let us name it VPOETPlugin) and use it like this in any JSPWiki page:

```
[{VPOETPlugin action='renderOutput' designID='X' provider='Y'
 source='Z' width='400' height='300'}]
```

[8] Available at the Google Gadgets Directory (look for GG-VPOET in `http://www.google.com/ig/directory`).

The implementation of `VPOETPlugin` reads the input parameters (`action`, `designID`, `provider` etc.), builds a HTTP message (a simple URL for HTTP GET messages like the one shown in section 3), and sends it to a running VPOET application (to a specific well-known VPOET servlet).

5 Conclusions and further work

The experience with GG-VPOET shows that the integration of VPOET templates into wikis could be easy in terms of wiki development effort and required competencies (basic skills in any programming language to create HTTP messages). This feature is specially remarkable in the wiki world in which wiki engines are written in most programming languages such as Java, Ruby, PHP or Perl. In any one of these languages it is easy to create and send a HTTP message. Additionally, wiki users benefit from professional web designs, providing attractive wiki pages which handle semantic data.

Our future work is to improve the features of VPOET in two directions: containers and user profiles. The former are intended to allow web designers create graphical containers (e.g. a tabs based agenda) to render multiple semantic individuals. The current implementation renders the individuals sequentially. The latter extends the HTTP messages to include a parameter pointing to a semantic description of the user profile. This profile can specify the user device (e.g. PC, handheld, TV) or the user interactive characteristics (e.g. color blindness, reduced visual sharpness). By providing this information, VPOET could return the "best"template for a given situation. An hypothetical wiki exploiting VPOET templates could request users to provide their user profile (URL) or provide an *ad hoc* user interface to create it and store it, providing wiki users with an interface adapted to their needs.

References

1. Oren, E., Delbru, R., Moller, K., Volkel, M., Handschuh, S.: Annotation and Navigation in Semantic Wikis. In Proc. SemWiki. CEUR-WS vol. 206, (2006)
2. García, R., Gimeno, J.M., Perdrix, F., Gil, R., Oliva, M.: The Rhizomer Semantic Content Management System. In Proc. WSKS. LNAI Series, vol. 5288, pp. 387–394. Springer (2008)
3. Oren, E., Haller, A., Mesnage, C., Hauswirth, M., Heitmann, B., Decker, S.: A Flexible Integration Framework for Semantic Web 2.0 Applications. IEEE Software **24**(5), pp. 64–71, (Sept-Oct 2007)
4. Rico, M., Camacho, D., Óscar Corcho: VPOET: Using a Distributed Collaborative Platform for Semantic Web Applications. In Proc. IDC2008. SCI Series vol. 162, 167–176. Springer (2008)
5. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In Proc. ISWC. LNCS 4273, pp. 158–171. Springer (2006)
6. García, R., Gil, R.: Improving Human–Semantic Web Interaction: The Rhizomer Experience. In Proc. SWAP'06. CEUR-WS vol. 201, pp. 57–64, (2006)