Integrated Project

Priority 2.4.7

Semantic based knowledge systems

Information Society Technologies

The Social Semantic Desktop

NEPOMUK

Interactive Semantic Wikis

Deliverable D1.1

Version 1.0

08.01.2007

Dissemination level: PU

Nature	Prototype
Due date	31.12.2006
Lead contractor	Cognium Systems SA
Start date of project	01.01.2006
Duration	36 months

Authors

COG: Mikhail Kotelnikov, Alexander Polonsky
DFKI: Malte Kiesel
FZI: Max Völkel, Heiko Haller
IBM: Mikhail Sogrin
KTH: Pär Lannerö
NUIG: Brian Davis

Mentors

DFKI: Ansgar Bernardi
DFKI: Thomas Roth-Berghofer

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Erwin-Schroedinger-Strasse (Building 57)
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM FUER KUENSTLICHE INTELLIGENZ GMBH (DFKI)

IBM IRELAND PRODUCT DISTRIBUTION LIMITED (IBM)

SAP AG (SAP)

HEWLETT PACKARD GALWAY LTD (HPGL)

THALES S.A. (TRT)

PRC GROUP - THE MANAGEMENT HOUSE S.A. (PRC)

EDGE-IT S.A.R.L (EDG)

COGNIUM SYSTEMS S.A. (COG)

NATIONAL UNIVERSITY OF IRELAND, GALWAY (NTUA)

ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (EPFL)

FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE (FZI)

UNIVERSITAET HANNOVER (L3S)

INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS (ICCS)

KUNGLIGA TEKNISKA HOEGSKOLAN (KTH)

UNIVERSITA DELLA SVIZZERA ITALIANA (USI)

IRION MANAGEMENT CONSULTING GMBH (IMC)

Versions

Version	Date	Reason
0.1	23.11.06	Outline
0.2	23.11.06	Added Integration parts
0.3	24.11.06	Integrated info from wiki and LaTeX files; marked open questions
0.4	29.11.06	Added draft texts for introduction + semantic wiki state of the art
0.5	29.11.06	Added language processing and semantic annotation parts
0.6	30.11.06	Updated Chapters 1, 2, and 3
0.7	01.12.06	Executive Summary, integrated Ch 4 & 6, and a part of Ch 3, modified Ch 2
0.9	07.12.06	1st assembled draft, sent to the mentors
0.10	20.12.06	Updates, reaction to mentor comments, typo correction and additional comments.
0.11	22.12.06	2nd assembled draft, sent to mentors
0.13	03.01.07	Final version sent to mentors and Markus
1.0	08.01.07	Finalised for EC by M. Junker

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for NEPOMUK partners

Executive Summary

Knowledge articulation, both individual and collaborative, is an integral part of a social semantic desktop. In Nepomuk, this articulation is mediated by a semantic wiki developed in the course of the project and integrated with the Nepomuk platform.

Wikis have simplified authoring and sharing of unstructured content. Semantic wikis aim at semantically annotating wiki content and deriving benefits from this annotation. The challenge is to improve information retrieval and management without sacrificing the trademark wiki flexibility and ease of use.

We have analyzed the state-of-the-art of both traditional wikis and semantic wiki prototypes in order to identify successful design solutions and popularity factors (Chapter 2). Wherever possible, the results have been applied in the ongoing development of the three Nepomuk wiki prototypes (described in Chapter 3). The links to test the prototypes are provided in the Appendix.

Just as there are many traditional wiki engines, several approaches are possible to building semantic wikis. However, semantics offers new opportunities for communication between different wiki engines. We have developed proposals and prototypes of semantics-based and other integration methods (Chapter 4).

Language analysis algorithms can greatly simplify formal content annotation, although they often require human input to reduce error. Some of the Nepomuk wiki prototypes rpovide features for an effective human-computer cooperation in content structuring (Chapter 5).

We have evaluated the Nepomuk semantic wiki prototypes based on the requirements derived in the four Nepomuk case studies (Chapter 6). We have found that the prototypes already satisfy several important requirements, although much still remains to be done.

Based on the gathered information, a roadmap for the subsequent Nepomuk semantic wiki development has been determined (Chapter 7).

Table of contents

1. Introduction

According to Wikipedia, a wiki is "a type of Web site that allows the visitors themselves to easily add, remove, and otherwise edit and change some available content". It is thanks to this ease of use that wiki popularity has surged and that Wikipedia itself has become the largest encyclopedia in the world within the 5 years of its existence [1]. By making Web site creation and modification simple and collaborative, wikis promote the original vision of the Web as a collaborative environment accessible to everyone.

At present, wiki engines (i.e., the software for wiki creation) count in the hundreds while the number of wiki contributors and wiki pages are in the millions. Google Trends analysis indicates that interest in wikis (i.e., people searching for the word "wiki" on Google) continuous to grow exponentially. Finally, "internet research firm Gartner Group predicts that wikis will become mainstream collaboration tools in at least 50% of companies by 2009" [2].

However, information accumulated in the wiki systems is unstructured, and hence, poses problems for knowledge management and productivity: "Gartner says the explosion of unstructured data is negatively affecting the productivity of individuals and the overall competitiveness of enterprises" [4]. The classic approach for dealing with this issue is for the information to be entered into structured forms or templates thereby becoming much clearer to humans and more accessible to computer-aided operations, such as structure-based search, data integration, and data analysis. Although this approach has worked very well for certain type of data (structured data), it has not worked well for all data (unstructured data). Indeed, a lot of information entered in a document format is difficult to input into a form. The same is true for information represented as a network, image, or sound.

The semantic systems are based on the inverse paradigm: as opposed to forcing information into a given form, information can be recorded in a free document format and then labelled with the corresponding semantic tags. This approach allows to combine flexibility with semantic organization during document authoring (e.g., wiki page authoring).

The goal of the "semantic wiki" development is to add to wiki systems the possibility of formal knowledge organization and the functionality to benefit from the knowledge formalism, while at the same time preserving the flexibility and ease of use of the traditional wikis. The potential benefits of semantic wikis relative to the traditional ones have been well summarized in a recent Project10X report [5]:

- "Concept-based rather than language-based searching: queries span vocabularies, languages, and search engines
- Question answering rather than simple retrieval. Also, overlay ontologies and knowledge bases can integrate with major web searching engines
- More richly structured content navigation, including multiple perspectives, multiple levels of abstraction, dependency/contingency relationships, etc.
- Easy visualization of content structure (categories, taxonomies, semantic nets, etc.). Direct editing of content structure.
- Mining of semantic relationships in content.

- Wiki content linked to dynamic models, simulations, visualizations.
- Wiki content linked with external repositories, file systems (e.g. personal desktop, enterprise servers, web sources, semantic-enabled feeds [e.g. RSS])
- Richer user access/rights models, including reputation systems."

At present, there are 22 prototypes of semantic wikis posted on the dedicated Web page at www.ontoworld.org [6]. These prototypes are based on different technological platforms and programming languages. But more importantly, they adopt different approaches to striking the balance between flexibility, simplicity, and semantic organization. These approaches are well represented in the semantic wiki prototypes developed in Nepomuk. The semantic wiki developed in Nepomuk will take into account the lessons learned from both internal (i.e., Nepomuk-affiliated) and external semantic wiki prototypes.

2. Semantic wiki state of the art

2.1. Popular wiki features

The most basic explanations to the success of wikis are probably to be found in wiki father Ward Cunningham's ”Why wiki works”[1]. This page not only describes the social phenomena which make the wiki concept useful, but it is also a page of the first (and obviously very successful) wiki: http://c2.com/cgi/wiki.

The wiki syntax of that page is minimal. It employs CamelCase in order to tell the system what should be considered a link. Otherwise, at the time of this writing (November 22nd 2006), it's just plain text. At least that’s what it looks like to a first-time user. If you look closer at the TextFormattingRules[2] you notice that there are actually some conversions being made: "---" is rendered as a ruler, asterisk (”*”) character makes for bulleted lists and so on, but these are only some superficial typographic improvements.

In terms of user interface widgets, they are very few. There are no buttons, no graphs, no visualizations, no sliders, no drop-downs or pop-ups. Just two action links: EditText and FindPage. Actually, the original wiki is so minimalist that it does not even have automatic layout. But that obviously did not stop it from triggering the whole wiki movement.

The key points in success of wikis were:

- *Collaboration aspect*: all information became immediately available for everybody.
- *Simplicity* in information interlinking: to create a link between documents it is enough to put the name of the desired document. So creation of a complex graph became easy.
- *Simplicity of document creation*: if a target document does not exist then it is enough to click on the link to start the editing process - the cost of creation of a new document is just one click.
- *Openness for reading and editing*: the information is open to readers as well as to editors. It considerably increases the implication of readers in the process of the information creation and eliminates borders between document creators and their consumers. In addition, users became personally involved in the process of the information creation, which changed completely the psychological perception of the information.
- *Openness for experiments*: another factor contributing to the information editing is absence of fear to destroy existing

[1] http://www.c2.com/cgi/wiki?WhyWikiWorks

[2] http://www.c2.com/cgi/wiki?TextFormattingRules

documents. In wikis this archived by using modification histories for all the information.

- *Fine granularity of information* – the simplicity of document creation and interlinking encourages creation of individual pages for each topic, term, or word. In wikis this information can be easily linked and found in different contexts which increases the re-usability of information. This factor contributes considerably to the collaboration aspect of wiki – each editor can work on an individual piece of the same information patchwork without interference with other editors, without loosing at the same time the whole picture and the context of the edited document.
- *Refactoring*: simplicity in document creation and versioning encourage refactoring of the information; if a document became too big, it can be easily split into many smaller ones.
- *Immediate rewards*: each modification in documents becomes published and visible right away.
- *Data more important than presentation*: editors have to deal with different wiki syntaxes. And it is a big drawback for end-users. But, at the same time the usage of such a syntax force editors to concentrate efforts only on the data creation and data structuring, and not on their visual effects, which can be considered as a positive aspect of the wiki syntax.

2.1.1. Secrets of the most popular wiki engines

Using the collective intelligence of the web, Mr Cunningham has compiled a list of the Top Ten Wiki Engines[3]. That list is supposed to be based on popularity, feature set and purposefulness. It's not a very scientific measure, but at least the list is based on some sort of general consensus among people worldwide who care about the topic. Otherwise it would have been edited by someone. (Today, November 22nd 2006, the latest edit was made more than a month ago.)

The “top ten” page features the following nine (!) wiki engines:

1. MoinMoin - A Python Language wiki engine, features flexibility and modular design.
2. MediaWiki - Used by the WikiPedia project, which is one of the most popular wikis (PHP and MySql).
3. PhpWiki - A very popular Php Language Wiki based on UseModWiki, with many features added.
4. OddMuseWiki - Really popular descendant of UseModWiki (one big Perl script).
5. UseModWiki - A Perl Language wiki, based on Wards original WikiWiki.
6. TWiki (TwikiClone) - A powerful, skinnable, extensive Perl Language wiki, aimed at large corporate Intranets.

[3] http://www.c2.com/cgi/wiki?TopTenWikiEngines

7. TikiWiki - A has-everything content management system with a powerful Wiki (PHP).
8. PmWiki - A popular Php Language Wiki, easy installation, simple design, nice feature list.
9. Best WakkaWiki fork - (Which WakkaWiki fork is the best?) (PHP/MySQL)

Another way of ranking wiki engines is to make Google searches to see how many pages are being served (and indexed by Google) from the different wiki engines. In order for this to work, the wiki pages must contain some sort of trace, which indicates what engine is being used. This is probably almost the case, but you cannot know for sure. Such a survey was conducted in August 2006, and is presented at http://www.wikicreole.org/wiki/WikiPopularity:

1. MediaWiki +
2. Twiki +
3. Confluence
4. TikiWiki +
5. PukiWiki
6. JotSpot
7. MoinMoin +
8. PBwiki
9. PmWiki +
10. DokuWiki

Engines with a plus sign appear in the manually constructed list above, so as you can see the overlap is about 50%.

2.1.1.1 Comparing wiki engines

The Wiki Matrix[4] is a tool for comparing wiki engines. It consists of a searchable database where a very large number of wiki engine characteristics have been recorded, for each of the most common engines. There are no evaluations – just listings of features, technical details and other plain facts. A comparison of those of the above wiki engines that appear in the Wiki Matrix gives the following insights:

Common properties

- Most are free and open source
- Most use PHP or Perl for programming
- Most run on several different operating systems

Common features

- All have a page history function

[4] http://www.wikimatrix.org

- All have a preview function
- All have unicode support
- All support full text search
- All support more than 10 languages
- Despite being wiki engines, all of them can handle page permissions
- All but one support internal comments and CSS
- All export RSS feeds

All feature a "recent changes" page

Note: These features may have appeared because of engine popularity, and are not necessarily the explanation for the engine being popular. But still, these features seem to be quite important, since somebody bothered to implement them in all popular engines.

Features which seem to be irrelevant for success

- WYSIWYG editing
- There's no correlation between popularity and storage method
- The CamelCase notation is not the only successful linking style. Some use [square brackets] or similar constructs instead.
- HTML tag support
- Math formula support
- FAQ tags (note: this can be seen as a light-weight semantic wiki feature)
- Scripting
- Feed aggregation
- Section editing

Multimedia editing features

Interestingly, the most popular of the commercial wiki engines - Confluence - differs from the top free systems in that it uses Java and not, as virtually all other popular engines, an interpreted language such as PHP and Perl.

2.1.2. User interface features in popular wiki engines

The above feature review gives *some* insight into good user interface design in wikis. But in order to better understand how popular wiki user interfaces are designed, we have taken a closer look at some of them.

2.1.2.1 Confluence

The most popular commercial wiki engine is Confluence. Confluence UI features include:

- Breadcrumbs indicating where you are in the wiki (works if there is any hierarchy at all)
- Rich text editing interface, showing some code
- Wiki-syntax editing interface, showing quite advanced code

- Tagging (with ajax-based autocompletion)
- Embedded email client
- Nice and clean default layout

2.1.2.2 Pbwiki

Pbwiki, which has both commercial and for-free options and has been extensively used by KTH employees, features:

- Templates
- HTML code editing
- Few steps needed to setup new wiki. Wiki owners don't host the wiki engine, instead they borrow or rent wiki space on a server provided by the maker of Pbwiki.

2.1.2.3 Media Wiki

Media Wiki, which is probably the most widely *read* wiki system (due to Wikipedia's popularity) can also provide some inspiration:

- Quite a lot of code may appear in editor, but you don't need to use much more than plain text
- Editor provides buttons which help in inserting formatting (and other) code in the wikitext
- Nice multi-language feature
- Nice and clean default layout

2.1.2.4 Conclusions for Nepomuk

From what it seems, the popular wiki systems have cleanliness as a premier common characteristic. Users don't seem to be deterred by some odd syntax here and there, but it should also be possible to just enter some plain text and press *publish*. There's really no need to have advanced multimedia editing features (but this may change later on).

Common features such as page history, recent changes et cetera, must be implemented at some stage if a future Nepomuk wiki engine is supposed to become widely used. But it might not be necessary to implement all of it from scratch. If the merits are enough to attract some developers, they will probably help implementing any missing base functionality. Whether or not external developers actually do this could be seen as a way of measuring the impact of the wiki engine.

2.2. Semantic wikis

While plain, non-semantic wikis have become very widely used, much due to their simplicity, they certainly do have many limitations.

One limitation is inherent in the foundational wiki concept "anybody can edit anything": A basic wiki has very limited rights management features. This limitation has been addressed by many wiki engines, by more or less sophisticated methods of identifying users, blocking malicious users by IP

number, blocking spam-bots by requiring the entry of non-machine readable data (typically displayed in a picture requiring manual interpretation) in a “password” field and so on.

Another area where traditional wikis are very limited is that of structure and semantics. There’s no way for a traditional wiki engine to provide users with business logic functionality above the level of free-text search:

- You cannot extract all descriptions of female physicians from a traditional wiki, unless those descriptions contain exactly the words “female physician” and the same phrase does not appear in any other part of the wiki.
- You cannot sort all persons described in a wiki by age.
- You cannot import structured data from a database into the wiki (e.g. for the sake of making it easy to edit collaboratively) and then make queries on the data as if it was still in the database.
- You cannot expose or export arbitrary selections of the wiki content to another software.
- The most basic kind of reasoning, such as “a Volvo is a kind of car, therefore a wiki page about a Volvo is a wiki page about a car” cannot be automatically performed inside a traditional wiki, even though the information is often present (but only in a machine readable format).
- Pages cannot be automatically formatted with respect to what kind of information they contain.

Semantic wikis attempt to address these limitations using methods and tools from the Semantic web, such as ontologies, metadata and tags.

2.2.1. Existing semantic wikis, an overview

In most traditional wikis, the idea of metadata typically only appears in a very technical way. For example, in JSPWiki, metadata is added directly into the wiki text using special tags, and mostly serves the purpose of implementing access control. In SnipSnap, labels may get attached to wiki pages, serving mainly as a categorization scheme.

2.2.1.1 Platypus

The semantic wiki Platypus adds RDF(S) and OWL metadata to wiki pages. Metadata has to be entered separately from wiki text and relates a wiki page to another resource; thus, metadata can be transformed into a list of related pages that can be shown along with the actual wiki page.

2.2.1.2 Semantic MediaWiki

The Semantic MediaWiki is an extension of MediaWiki, the software used by Wikipedia. Again, metadata associated to a wiki page may point to other resources, but here, literals are allowed, too. Also, metadata is entered directly into the wiki text, and does not have to adhere to a schema. A nice feature of this implementation is its support for multiple data types such as coordinates and temperatures, along with conversion between different unit scales.

2.2.1.3 Rhizome

Rhizome [7] builds on a framework that adapts techniques such as XSLT and XUpdate to RDF. In essence, RDF is used throughout the framework for almost everything, and RxSLT (an XSLT variant adapted for RDF) is used for transforming query results to HTML or other output formats. Page metadata has to be entered separately from the page. While the approach is very interesting from a technical point of view, the current implementation requires a lot practice with the underlying techniques.

2.2.1.4 IkeWiki

IkeWiki [8] is a rather new wiki supporting OWL ontologies. It supports inferencing when typing links and relies on JavaScript-based features for supporting the user, which helps quite a lot when adding semantic information.

2.2.1.5 OpenRecord

OpenRecord is a kind of database/spreadsheet wiki. It focuses on enabling the user to enter structured data using tables. It heavily uses JavaScript, providing almost the feeling of a standalone application. However, currently it is in alpha stage only.

2.2.2. Current R&D issues in semantic wikis

While the log-term R&D plans for semantic wikis are still unclear, several immediate issues in this domain are apparent.

2.2.2.1 Interoperability

While one of the main points of semantic web standards is interoperability, there seems to be no semantic wiki that allows import of RDF data. Some wikis allow usage of ontologies (in OWL or RDFS language), but integration into the wiki concepts seems to be amendable. For example, ontologies loaded typically do not show up in the wiki since they are loaded into a separate repository. Thus, ontologies are deemed to remain static and cannot be edited by users of the wiki.

2.2.2.2 Is it a resource, or a wiki page *about* a resource?

Several (but not all) existing semantic wikis take the simple approach of supposing that the URI of a page about concept C can be regarded as the URI of C. The simplification is reasonable while working with wiki based encyclopedias such as Wikipedia. Typically, such wikis do have one page per concept. The simplification makes it almost as easy to express semantic statements about a concept as it is to make a link to a wiki page. It also has the nice effect that if the user does not explicitly state what resource his statement is about, it can be assumed that it refers to the concept of the page where the statement appears.

However, it follows that when using RDFS, wiki pages must be both of the type wiki:page and of the type the resource the wiki page is

supposed to describe. This has two drawbacks: First, from a knowledge engineer's point of view, existence of an entity that is both a text (a wiki page) and, for example, a person, is not desirable. Second, while the approach can be handy for generating RDF data of “shallow'' ontologies with few classes and many relations, we think that it reaches its limits as soon as more elaborate ontologies and structures are used.

For example, imagine a large table that lists 100 products along with a short description and price. In order to express this in a semantic wiki that identifies a page with a resource, one gets forced to create 100 wiki pages, one for each row of the table, both cluttering title index and recent changes pages.

Furthermore, the simplified model puts a restriction on the level of granularity of possible annotations: If you can only refer to “the concept of the page”, how do you refer to a specific fragment of the page?

So there are both advantages and drawbacks to the simple approach. Preferably, a future superwiki should be able to encompass both the simple case where each page is about a concept, and the concept can be referred to by the same URI as the page, and the more general case where a page can contain an arbitrary number of statements not necessarily related to the concept of the page (if there is any).

2.2.2.3 Annotation support

While some existing semantic wikis allow addition of semantic features to existing content (for example, by typing previously untyped links in the wiki), no wiki seems to provide features to assist the user when extracting further semantic features from (imported) plain text.

2.2.2.4 Queries

The only means of querying semantic information is either very simple queries built with a user interface (such as “Show a list of all publications to me'') or complex queries entered manually in a query language such as SPARQL [9].

While current research and development holds promise of some interesting query interfaces (such as faceted browsing), the potential of today’s semantic wikis as independent search systems is still rather difficult to harness. However, they can be of great immediate value when integrated in larger systems, such as the Nepomuk platform, that provide the semantic search functionality.

3. Nepomuk Semantic Wikis

3.1. Overview

In order to explore problems and advantages of different semantic wiki approaches, we developed several independent semantic wiki prototypes which will be described in this chapter.

The first one, Semantic Pad already unites two different approaches: it can be used remotely through a web browser like usual wikis but it has also been integrated into the Eclipse Rich Client Platform for better integration with other future RCP-based Nepomuk components. For reasons of clarity and ease of use, it follows the one-page-per-concept paradigm where each concept has a corresponding wiki page that holds all statements about this concept. One of its specialties is a templating feature, which allows to view the same information in different customized layouts.

The second semantic wiki prototype is called Kaukolu Wiki ("kaukolu" being Hawaiian for "triple"). It is implemented as an extension to JSP-Wiki, an already existing browser based Wiki engine. Other than the above, it allows arbitrary statements on any wiki page, even about concepts outside the wiki. Furthermore it supports semi-automatic annotations with an ontology-based auto-completion feature.

The third semantic wiki described here is Semantic MediaWiki. It has been collaboratively developed in the projects SEKT (www.sekt-project.com) and NEPOMUK. While it was primarily targeted to enhance Wikipedia with semantic capabilities, it offers a set of features that are also useful for personal use: E.g. it provides several largely customizable ways to embed the results of complex queries into a page. One of them is an interactive timeline visualisation. Semantic MediaWiki is currently probably the most wide-spread and most mature semantic wiki in use.

A functional comparison between the three Nepomuk wiki prototypes is summarized in the table below:

Functionality	Semantic Pad	Kaukolu	Semantic MediaWiki
Arbitrary statements	-	X	-
Literal statements	X	X	X
Typed references with other pages	X	-	X
Back links grouped by properties	X	-	-
Annotations that words to resources	-	X	X
Automatically generated lists	-	X	X
Semantic search	-	-	X
Inter-language consistency	-	-	X
Type specific views	X	-	-
Template-based editing	X	-	X
Autocompletion	-	X	-

Timeline view	-	-	X
RDF(S) import	-	X	-
RDF(S) export	-	X	X
External reuse	X	X	X

Table 1 Functional comparison of the wiki prototypes.

3.2. Semantic Pad

The main idea behind Semantic Pad is to create a system conforming to the following criteria:

- Standards-based implementation (osgi, jcr-compatible, ...)
- When possible, re-use existing industrially established technologies
- Extensibility and easy adaptability for future re-use
- Adapt the key success factors of wikis in the field of Semantic Web
- Individual components of this application should be useful at the server side as well as at the client side.

The Semantic Pad was engineered as a personal wiki engine. But at the same time it keeps doors open for further extensions in the direction of a server-side application.

The Semantic Pad can work in two different modes: as a simple web-based application and as an application tightly integrated with the Eclipse Rich Client Platform [http://www.eclipse.org/rcp/]. Eclipse RCP provides important features not available for simple web-based applications such as tight integration with the operating system, integration with Eclipse/SWT components, and the possibility to edit the wiki content in specialized Eclipse-based editors.

3.2.1. Functionality

At the present, the following semantics-based functionalities have been implemented:

- Each wiki document is equivalent to an RDF resource. So each page can be used as a statement subject, predicate or object
- Each document can have *formatted literal statements* (paragraphs with styled text, headers, tables, ...) as well as typed references to other pages.
- *Facet document representation based on templates*: document visualization depends on the type of the current document. If for a specified document type there is no particular template, the default template is used. Facet templates define document layout of formatted document sections and references. These templates can manipulate with direct references (from the current page), typed back-references (to the current page), and results of complex structured queries (SPARQL) that represent complex relations of the current page with other resources. Content

structuring becomes immediately rewarding – the user sees immediate benefits of automatic structure-based formatting. The facet views are implemented using a flexible template system based on a widely used FreeMarker template engine [http://www.freemarker.org/]. To implement a new specific view (for example: “Person” or “Organization” view) it is necessary to define the real layout of the document's properties and typed direct or inverse links within the corresponding template. It is important to note that each wiki page can define multiply view types (facets) at the same time. For example the same document can be shown or edited as “Person”, “Friend”, “Collaborator” and “Taskforce Leader” at the same time. Each facet reflects only one particular aspect of the information defined in the page itself or in related resources (by direct or inverse properties).
An important aspect from the point of view of their system dissemination is that the Semantic Pad's template system shifts a part of work from developers to integrators. Integrators can easily create facet templates, thereby considerably enhancing the usability of the system at whole. Hence, it is easy to adapt the system to a domain-focused distribution that takes into account the concrete needs and preferences of the domain users.

- *Template-based document editing*. When user clicks on the "edit" button the system proposes to choose a template from the list of templates (facets) associated with the document.
- *Automatic wiki page annotation*. Semantic Pad has been integrated with the IBM LanguageWare technology. Based on a given ontology, LanguageWare suggests to the Semantic Pad users relevant disambiguated keywords for wiki page annotation (see Chapter 5 for details).

3.2.2. Architecture

Semantic Pad has two different configurations – the first one is a pure web-based application, the second one is integrated in the Eclipse Rich Client platform and communicates with the rich client components. In both versions Semantic Pad uses the same technological basis – the OSGi [http://www.osgi.org] platform which is the basis of the Eclipse plug-in system. The entire Semantic Pad application was designed as a set of plug-ins which are then re-packaged in the Rich Client version. This shows the high degree of re-usability of the components. The Rich Client Semantic Pad contains additional components providing integration with the Eclipse Rich Client Platform (e.g., embedded web-browser and the workspace explorer).

The main Semantic Pad components are:

1. Hierarchical Finite State Machine (HFSM)
2. Template API
3. Semantic Pad Data Storage System
4. The wiki engine

All components mentioned above are packaged as OSGi bundles.

3.2.2.1 Hierarchical Finite State Machine (HFSM)

The Semantic Pad uses Hierarchical Finite State Machine (HFSM) [http://www.quantum-leaps.com/resources/glossary.htm#HSM] as the main internal engine for executing the application. It provides the possibility to formally define the logic of the whole application, as opposed to just a sequence of transitions between web pages visualizing the data, as it is done in other web frameworks (e.g., Jboss Seam, Java Page Flow, Struts, Spring WebFlow).

The main ideas behind this approach are:

The system always exists in a state and changes the state during an event

Each state can have its own sub-states; all transitions between sub-states are defined in the parent state

The kernel of the system executing transition between states notifies user-defined state handlers about each stage of the process (stored as XML files): when the system goes into a state or when it leaves the state. All application-specific operations (e.g., data loading, access rights checking, data handling, visualization) are performed by these user's handlers. In the terminology of Model-View-Control (MVC) pattern our HFSM implementation realizes pure control functions.

The advantages of this approach are:

- The application logic becomes easily testable - the logic of state transitions can be tested in unit tests independently from user interfaces or application data. Only once the application is tested, specific java handlers can be added to define the specific behaviour in these states.
- The whole process can be temporarily stopped in any place to perform additional operations such as client-server communication. These feature is especially important in the context of web applications – it enables the server to show visualized user interfaces, to receive user's response, and to continue the same process.
- The state of the current user's session is explicitly managed on the server.
- Added security - the HFSM guaranties that the system can not arrive in a specific state without passing through the previous states. For example, we can be sure that the system arrives to the editing state only after the authorization state. It gives the possibility to create a simple application and add access rights verifications later.
- Well-defined application behaviour - application behaviour is defined not by URLs but by the internal state. The URLs are used only to identify resources, and not the operations performed on them.

The HFSM operations are executed in the following order (Fig. 1):

1. FSM servlet registered in the embedded OSGi HTTP Service receives user requests and asks the FSM Kernel to create a new FSM process corresponding to the URL.
2. FSM Kernel is used as a registry of all available FSM process descriptions. It creates new process instances based on descriptions and launches them. The Kernel notifies the FSM

Processor about the process stages, i.e. when a process enters or leaves a state.

3. FSM Processor locates the description of a user-defined State Handler, instantiates it, and calls activate() or deactivate() methods depending on the process stage. Usually, each state is associated with a state handler. By defining various handlers developers can perform actions specific to each logical state defined in the state charts.

3.2.2.2 Java Wiki Model

Semantic Pad is based on the Java Wiki Model developed within Nepomuk (see section 4.7.). The Java Wiki Model can be used as a comprehensive solution to parse, validate and re-construct well-formed wiki documents containing semantic information. It has been integrated into Xwiki – a commercial open-source wiki application – to be able to add semantic aspects to this application.

3.2.2.3 Template-based visualization

A simple API for template-based content visualization has been defined in order to abstract from specific template engines. Right now there are two different implementations of this API – one uses Velocity [http://jakarta.apache.org/velocity/] and the second is based on the Free Marker template engine [http://www.freemarker.org/

3.2.2.4 Semantic Pad Data Storage System

Semantic Pad re-uses the Eclipse Workspace to store the generated information. The Eclipse code build system activates “build process” when an object on the disk is modified. This code builder system is used to extract the semantic information from wiki pages and store it in the underlying RDF storage.

3.2.2.5 The wiki engine

The wiki engine defines the application logic in the form of FSM state charts and corresponding handlers executing specific actions for each state. There are two types of state handlers: action and view. State handlers of the view type are responsible for the data visualization of the internal resources (wiki pages and binary data) using corresponding templates. The name of the used template is defined by the view action itself.

The diagram below summarizes the Semantic Pad's architecture:

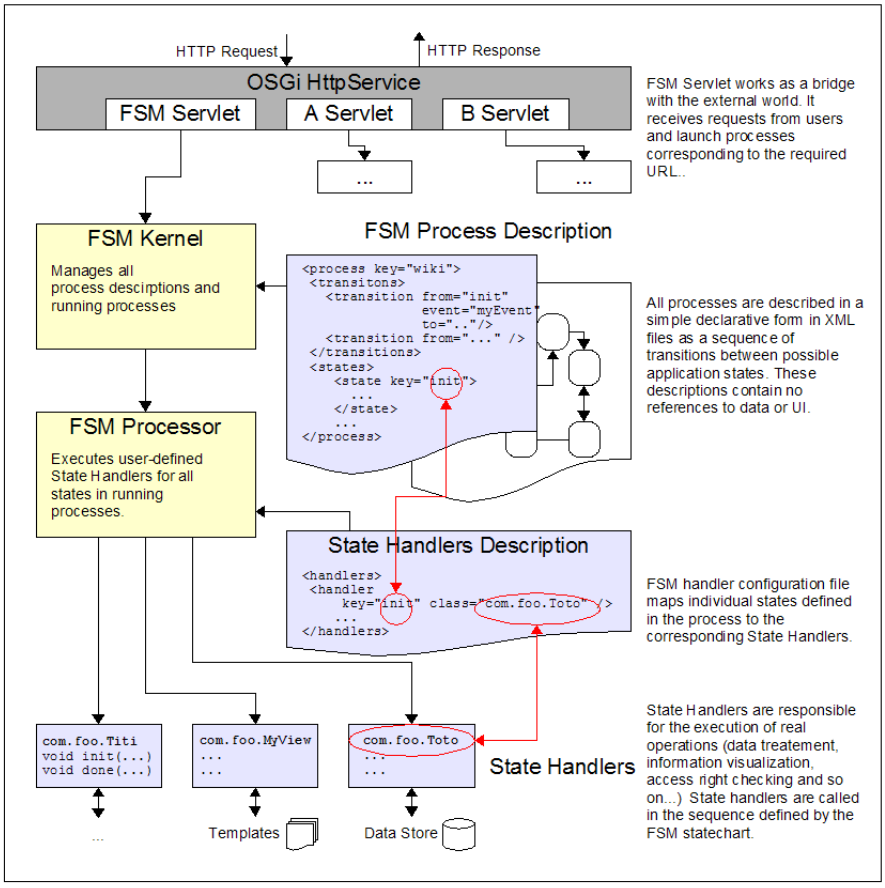

Figure 1. Semantic Pad architecture overview.

3.2.3. Screenshot illustrations

The screenshots below illustrate the usage of semantic document properties in Semantic Pad.

3.2.3.1 Authoring and annotation

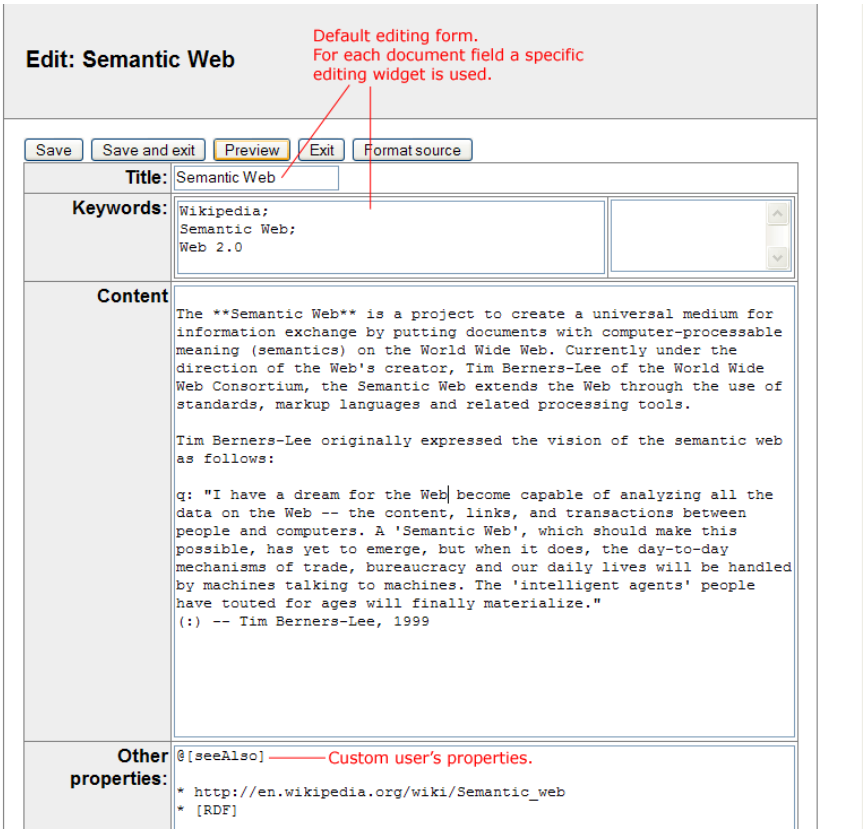

Figure 2. Template-based document editing

3.2.3.2 Automatic annotation

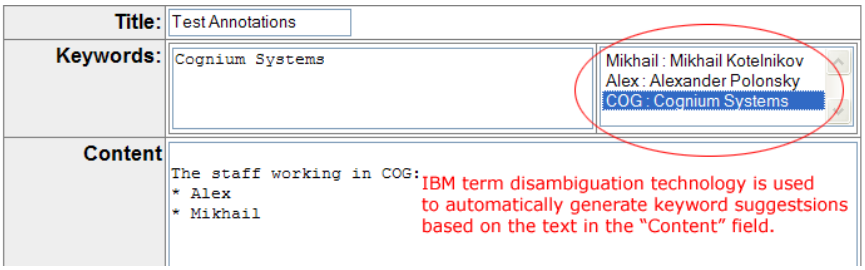

Figure 3. Automatic annotation by disambiguated keywords is based on the text in the "Content" field:

3.2.3.3 Template-defined views

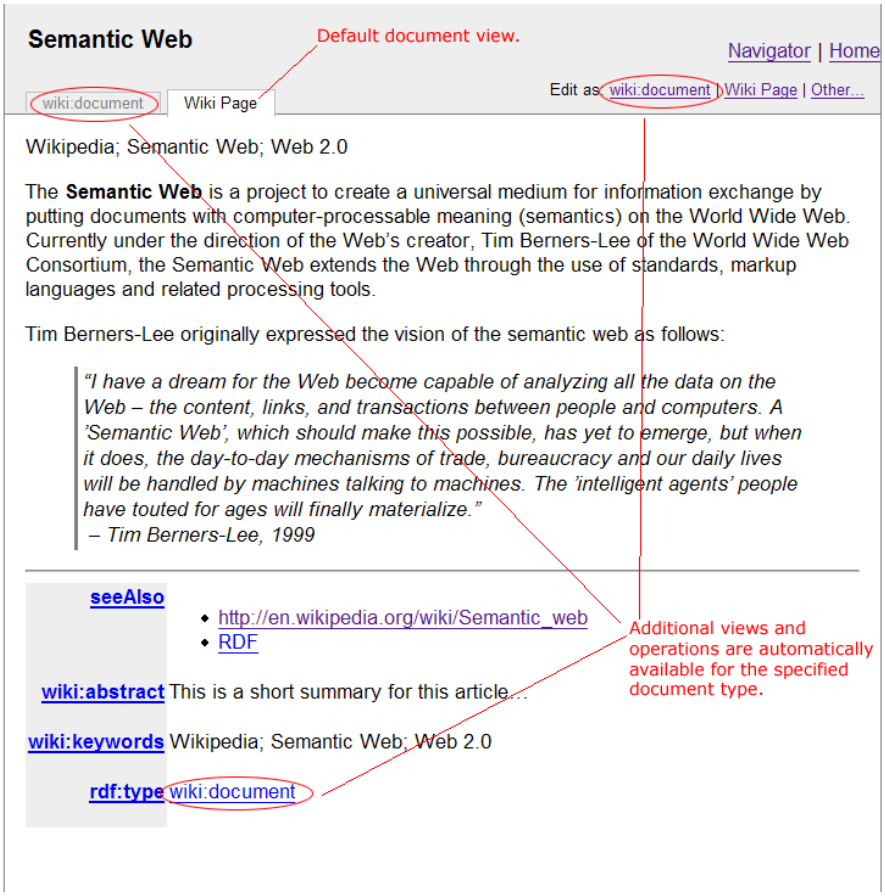

Figure 4. Default wiki page visualization

Figure 5. A template-based visualization specific for the type "wiki:document"

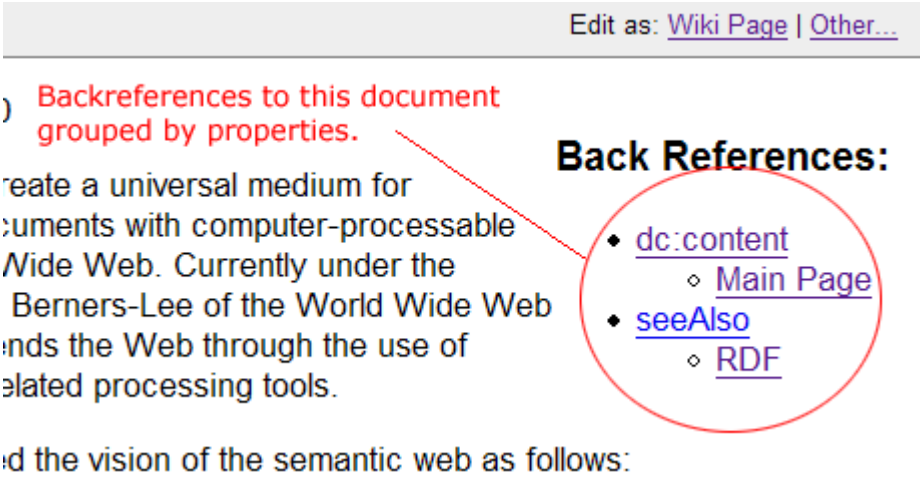

Figure 6. Back-references (i.e., inverse links)

3.3. Kaukolu wiki

Kaukolu differs from other semantic wikis insofar as it does not try to map wiki elements (pages, paragraphs) directly to RDF resources. In contrast, it sees a wiki page as a container that can be used to store arbitrary RDF [10]. The reasoning behind this is that while for the use cases that are encyclopedia-like it is fine to treat a wiki page as an RDF resource and assume that any statements in the wiki page's text describe the resource the wiki page is associated with, most non-encyclopedian

wiki sites such as intranet wikis or project wikis differ from this notion. There, a wiki page is just a container for text, often exhibiting characteristics of information collections such as brainstorming pages or item lists. For these use cases, more freedom when generating RDF is desirable if one does not want to restart the wiki from scratch when moving from a standard wiki to a semantic wiki engine [11].

3.3.1. Functionality

Kaukolu provides two modes of operation: *Standalone* and *embedded in gnowsis* – the Semantic Desktop environment used in Nepomuk (http://www.gnowsis.org/ [12]). The modes serve different purposes.

3.3.1.1 Standalone and embedded mode

Standalone Mode: Kaukolu is launched from a servlet container such as Jetty or Tomcat. Kaukolu uses its own embedded triple store and allows usage of notification and RDF(S) import plugins. This mode is intended for use in intranets without any additional components necessary on the end-user side. Typically, multiple users share one Kaukolu installation in this setup.

Embedded Mode: In embedded mode, Kaukolu runs as part of gnowsis, sharing the RDF store. It is possible to annotate and relate between pimo:Things (concepts in gnowsis' Personal Information Management Ontology) using Kaukolu in this mode. Since gnowsis takes care of ontology import, the respective Kaukolu plugins are not necessary. Since users runs individual gnowsis instances on their desktops, Kaukolu becomes a personal semantic wiki.

3.3.1.2 Features

Kaukolu differs from other semantic wikis in several respects. Unless indicated otherwise, the following features apply to standalone mode only - some of these features are also implemented by gnowsis.

No Restrictions on RDF Triples

Kaukolu allows to formulate arbitrary RDF on any wiki page using a slightly extended wiki syntax. Subjects of RDF triples are not required to represent the URI of the page where the triple is located. This allows for more complex RDF data that is compliant with any RDFS ontology regardless of its complexity without the need of setting up one wiki page for every RDF resource.

In the embedded mode, every pimo:Thing gets assigned one wiki page. Therefore, wiki pages are about a resource. While it is not strictly necessary to formulate only triples related to this resource, the interface design strongly encourages this. The main focus of arbitrary triples is that this way people can create lists of things/statements on one page, which mostly occurs in the intranet/standalone scenario.

RDF(S) Import and Export

Being able to associate arbitrary RDF with a wiki page not only works when formulating RDF but also allows to import RDF. In fact, since RDF Schema is also represented in RDF, one can even import RDFS ontologies to Kaukolu using this method. Imported RDFS ontologies can be used in

various ways within Kaukolu, first and foremost by the autocompletion feature (see below).

A direct benefit of RDFS ontologies being stored on wiki pages is that this way users are able to edit and extend the ontologies used by the wiki in a straightforward way, using all features a wiki provides (versioning, collaborative authoring, viewing diffs, ...). However, one has to say that currently changing RDFS using this approach is quite difficult as one has to directly work on RDFS without any tool support. However, it is possible to write plugins that can act as a simple ontology editor.

It has to be noted that while imported ontologies can be edited inside Kaukolu, there is no checking of existing RDF instances against changed schemas. This means that, for example, predicates of triples are not changed if the corresponding property definition is edited. As ontology evolution is a field of research in its own, this was out of scope of Kaukolu development.

In the embedded mode, importing schemas is done from within gnowsis, so there is no need to rely on Kaukolu functionality in this case.

Aliases Replacing namespace:localname URIs

In contrast to most existing semantic wikis, users of Kaukolu are not required to use localnames, labels, or namespaces of RDFS properties in order to express RDF triples using these predicates. For example, typically the user has to write *dc:author* "Author Name" if he wants to express that the current wiki page has a Dublin Core author property. In Kaukolu, we allow an intermediate step: every RDF instance or RDFS property may be associated to arbitrary strings (*aliases*) that can be used instead of the URI/label of the respective property or instance. Aliases are defined using the *hasSubjectURI*/*hasPredicateURI* keyword. This not only relieves the user from having to remember namespaces or localnames but also facilitates internationalization by usage of ontology metainformation.

In embedded mode, pimo labels are used as aliases instead of relying on explicit alias definition using *hasSubjectURI*/*hasPredicateURI.*

Autocompletion for Both Semantic and Non-Semantic Content

Despite wiki syntax and aliases for properties and instances, entering RDF triples is a tedious task. Without further support, the user would need to keep the documentation of the ontologies always at hand, typing mistakes would introduce severe errors, and the user would have to remember the URIs of all RDF instances created in the wiki. In Kaukolu, there is ontology-based autocompletion support, which proposes aliases based on RDFS range and domains. For example, when typing "Dirk knows", with *Dirk* being a *foaf:person*, and *knows* being associated to *foaf:knows*, the system automatically proposes a list of *foaf:persons* defined in the wiki to complete the RDF triple, as only *foaf:persons* are allowed as range of *foaf:knows*, even without any prefix typed. If a prefix has been typed already, it is used to narrow down the list of suggestions. Autocompletion works for predicates, too. In case no alias is found in the typed text, Kaukolu assumes that the user does not intend to write triples, and simply proposes names of wiki pages as autocompletion suggestions, based on the prefix already typed. So if you type "InfoOn", and there are "InfoOnDirk" and "InfoOnNasim" pages in the wiki, both of those page names are suggested.

In the embedded mode, ontologies and aliases defined in gnowsis are used for the autocompletion feature.

In the standalone mode, in addition to the ontology-based autocompletion suggestions, a natural language processing module has been integrated in the autocompletion feature. At the present, for the purpose of a proof-of-concept, the module provides disambiguation suggestions for a number of terms of the NEPOMUK project domain.

All Standard Wiki Features are Implemented

Most other semantic wikis have been rewritten from scratch and therefore miss several standard wiki features such as file attachments, access control, plug-in support, or support for multiple backends. Kaukolu is based on JSPWiki, an established wiki that is quite feature-complete.

3.3.2. Architecture

Kaukolu is an extension to JSPWiki, a well-known Java-based Wiki with an active community. Therefore, Kaukolu provides all features known from standard wikis. However, this is not without costs: The JSPWiki architecture is quite big, keeping in sync with JSPWiki development is sometimes not trivial, and major changes in functionality that is implemented in JSPWiki and used in Kaukolu are not possible without essentially setting up a separate fork. Therefore, Kaukolu does not redefine the wiki experience but merely tries to act as a toolbox for RDF-enabled plugins.

Kaukolu basically consists of a set of JSPWiki plugins, a modified frontend template, and some backend changes. These backend changes have been necessary mostly due to the RDF-based features Kaukolu provides. For example, the wiki page storage backend (*page provider* in JSPWiki terminology) of JSPWiki has been extended with means to access an RDF store (Sesame2 in this case). Multiple providers have been implemented - in addition to a page provider that allows Kaukolu to run with an embedded RDF repository, there is a provider which lets Kaukolu use the RDF repository of Gnowsis which allows to browse pimo:Things in Kaukolu easily.

In addition to generic RDF extensions, there are also some more traditional plugins such as an enhanced mail notification plug-in, an LDAP authentication plug-in, and a page hierarchy plug-in. All of these functionalities are known to be very useful in intranet settings such as the TMI usage case (Nepomuk Workpackage 9). The plugins can be used with a non-modified JSPWiki engine. They have been announced on the JSPWiki website and mailing list and are known to be in use in the community.

3.3.3. Screenshot illustrations

Below we demonstrate Kaukolu's gnowsis integration. We will take a look at a pimo:Person as displayed within gnowsis, then add an annotation from within Kaukolu using its autocompletion feature, and then show the new annotation in gnowsis.

For this example, two personas, Dirk of SAP and Nasim of TMI, are used.

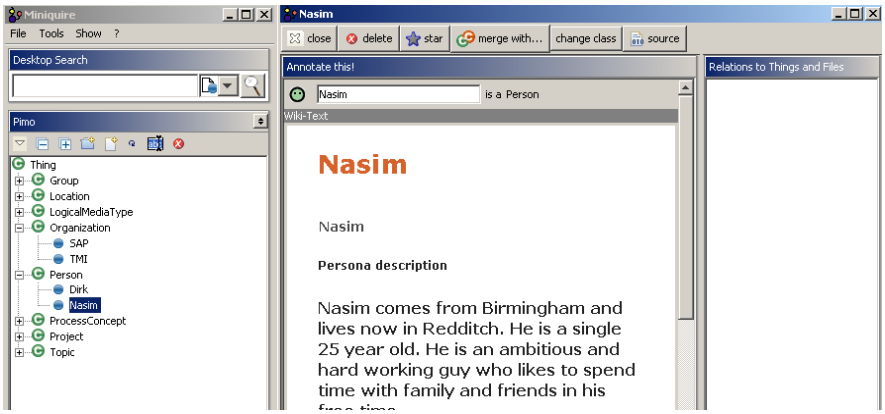

Figure 7. The persona Nasim as seen in gnowsis

In Figure 7, both personas can be seen as instances of the Person class (left window). The wiki text in the center window has been entered and rendered by Kaukolu. Note that there are no relations available at this stage (right window).

Using the “edit wiki” button in gnowsis, we open the Nasim wiki page in Kaukolu and edit it using Kaukolu’s editor.

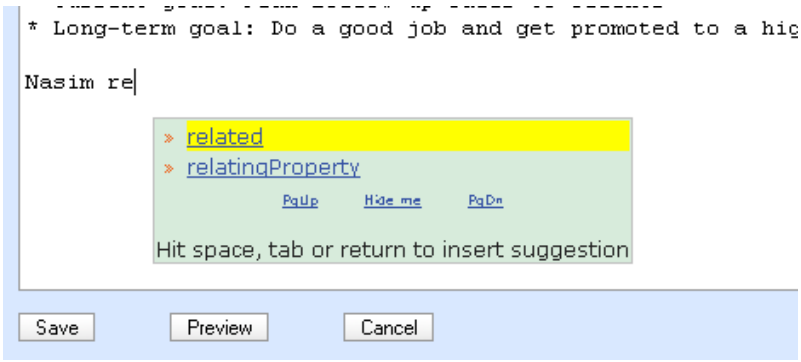

Figure 8. Editing the Nasim wiki page

In Figure 8, we add a new relation to Nasim using Kaukolu’s autocompletion feature. Note that “Nasim” is an alias associated with the Nasim person in gnowsis. We add the statement “Nasim - related - TMI”. With customized ontologies, using a more specific relation is also possible.

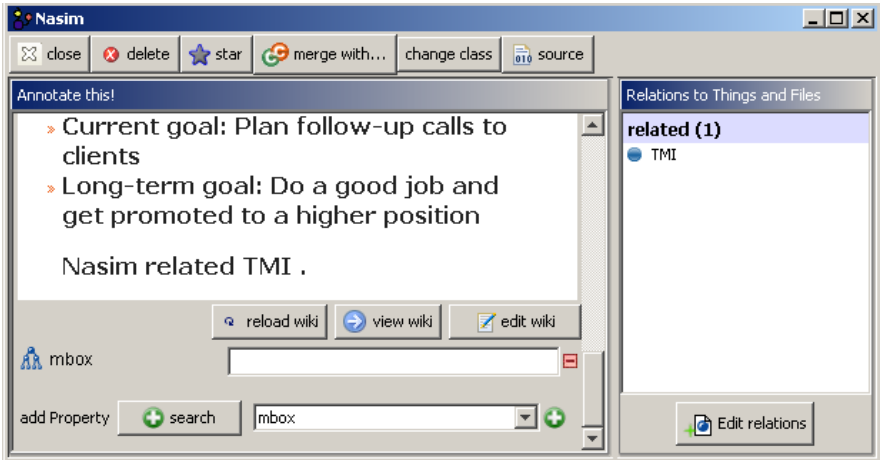

Figure 9. The result as seen in gnowsis

In Figure 9, the result in gnowsis can be seen. Note that the new relation defined in Kaukolu is also visible in gnowsis. It can be used to navigate through the user's personal information space or for querying.

3.4. Semantic Media Wiki

Semantic MediaWiki is an extension of MediaWiki – a widely used wiki-engine that also powers Wikipedia.

It has been developed in Karlsruhe at AIFB and FZI, financed partly by the European Projects SEKT and NEPOMUK.

It makes semantic technologies available to a broad community by smoothly integrating them with the established usage of MediaWiki. While its main target is to establish the "Semantic Wikipedia", it is already in use in numerous publicly accessible installations. It is also reported to be successfully used as a personal wiki by some users.

3.4.1. Functionality

In the Semantic MediaWiki paradigm, each article (wiki page) describes a concept.

While classic wikis only know unspecific Hyperlinks between pages (light blue arrows in following figure), in Semantic MediaWiki Semantic Statements can be made by freely assigning link types to these hyperlinks (thick black arrows). Additionally, attributes of any concept can be specified in a syntax similar to the link syntax – instead of referring to another concept, they simply refer to a value (e.g. area and population in the following figure).

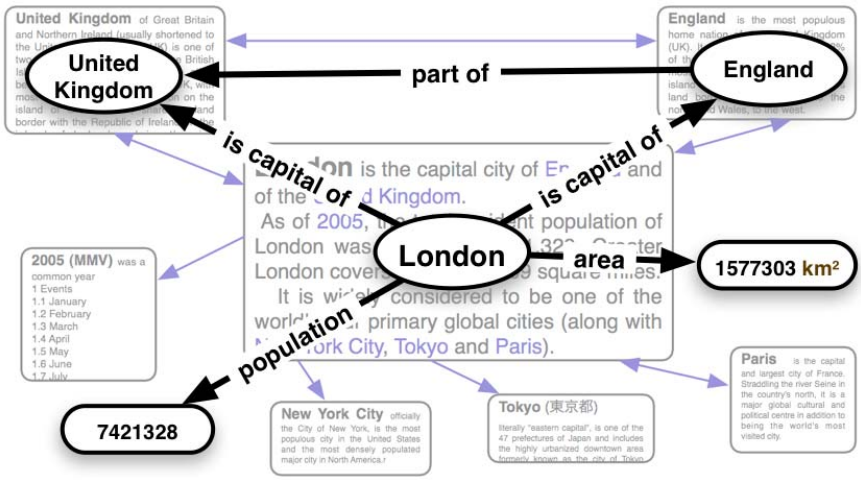

Figure 10.

Semantic contents of the wiki are made available in several ways:

- via RDF export (per page or in full)
- listed at the bottom of each page in a navigable way
- querying with a semantic search form (see below)
- providinig a SPARQL end point
- using inline queries, that can be embedded in any wiki page and that display the actualised results in a customizable table each time the embedding page is displayed (see below).
- via plugins, like the timeline (see below)

Apart from this, Semantic MediaWiki is also capable of importing OWL/RDFS Ontologies that can be used as background knowledge. Concepts and relations specified in the wiki can also be mapped to those of the imported ontologies.

Details on the functionality of Semantic MediaWiki can be found in the above mentioned article or, with the most recent actualisations on http://ontoworld.org/, a reference implementation and Semantic Web community Wiki.

3.4.2. Architecture

Semantic MediaWiki is a extension that plugs in to the MediaWiki Engine. Like MediaWiki, it is entirely written in php and JavaScipt.

Detailed descriptions on its architecture can be found in the WWW conference paper “Semantic Wikipedia”:

http://www2006.org/programme/item.php?id=4039

3.4.3. Screenshot illustrations

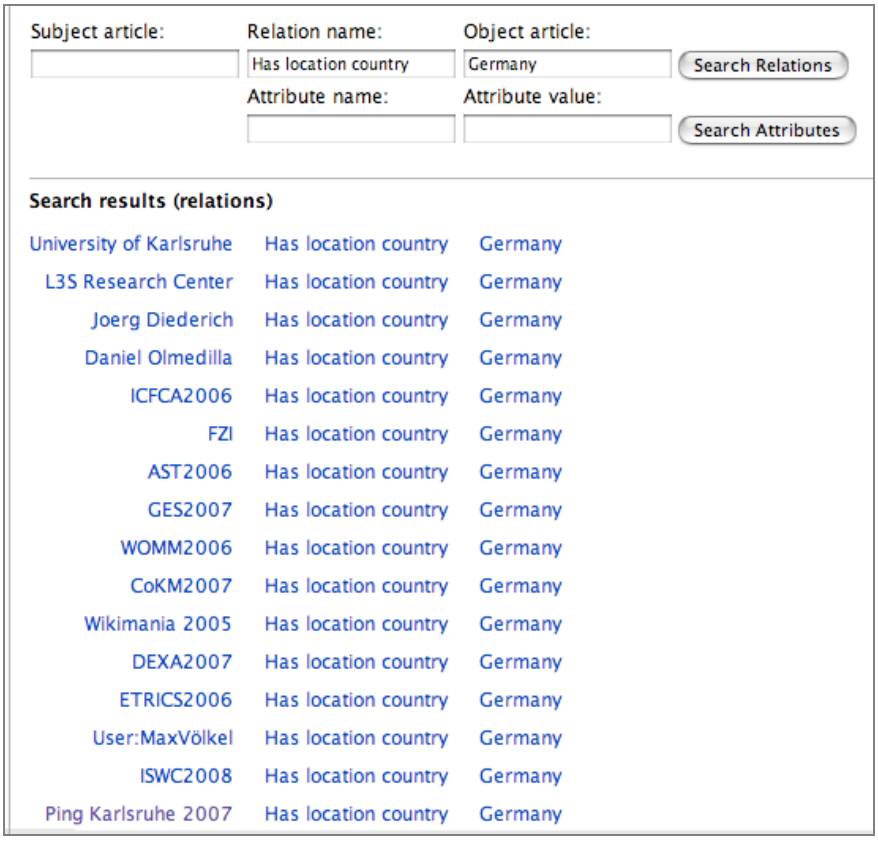

Figure 11. Semantic Search Form

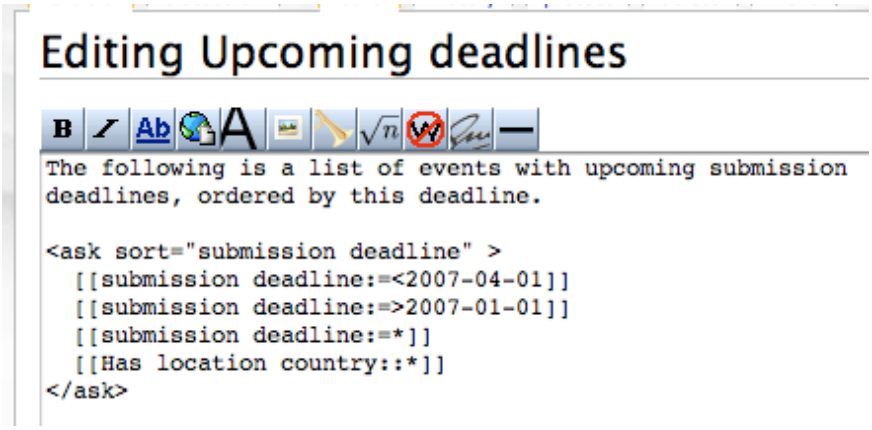

Figure 12. Inline query (source)

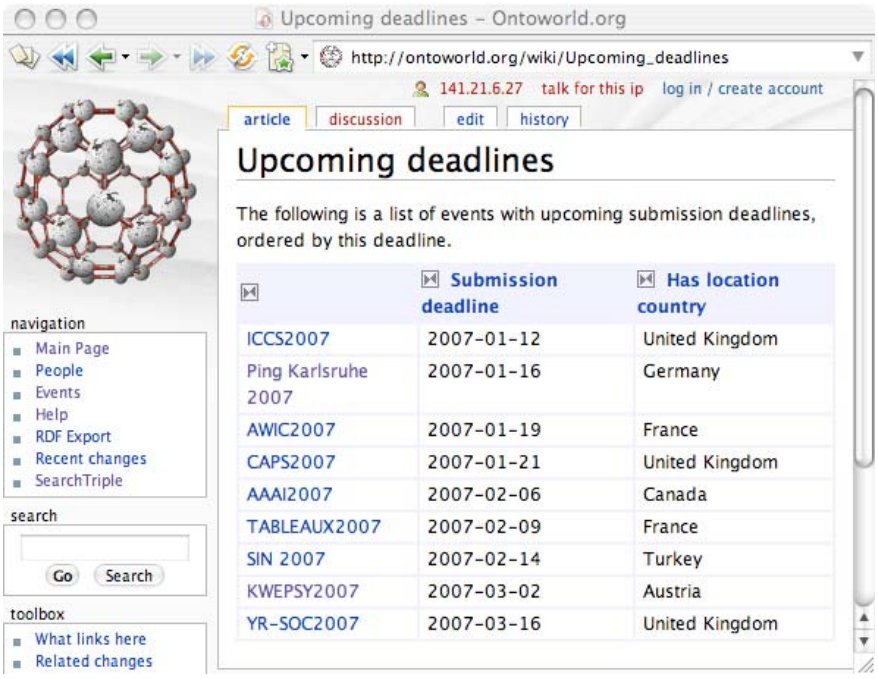

Figure 13. Inline query (result)

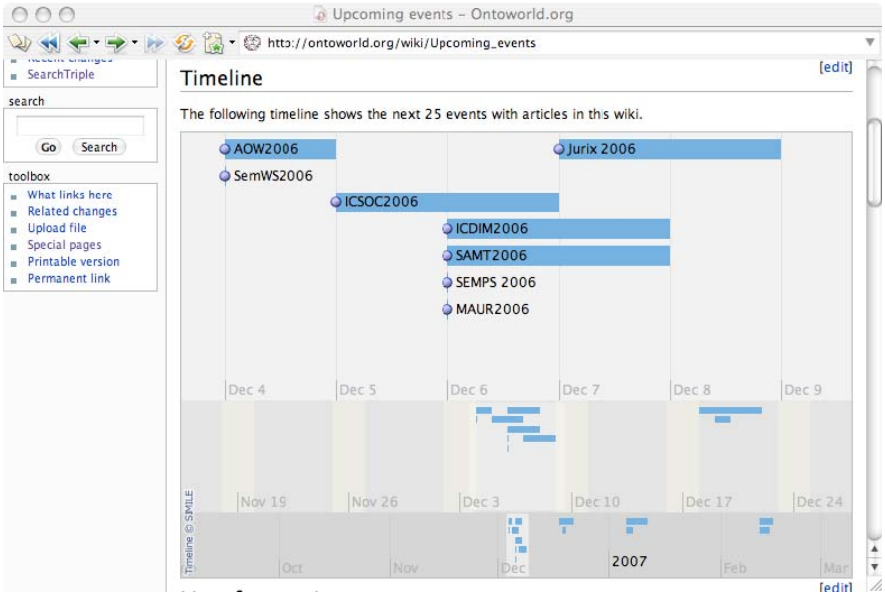

Figure 14. The interactive timeline plug-in (http://simile.mit.edu/timeline/) displays the results of an upcoming events query.

4. Wiki integration

4.1. Summary

With the rising popularity of wikis – and now also semantic wikis – a rising demand for integration has appeared. People want to connect one wiki with another or a wiki with an external application.

In this chapter we explain wiki integration scenarios (migration, backup, import, export, sync), explain problems (different syntaxes, no API) and describe the solutions we found: Wiki Interchange Format (WIF), Common Semantic Wiki API (CSWA), WikiModel, and a Wiki Metadata Ontology (WMO).

The CSWA is an API intended specifically for integration of semantic wikis in a semantic desktop. WikiModel is a framework for semantic wikis in Java. WIF is a generic interchange format for semantic and non-semantic wikis. The WMO is used both by WIF and by CSWA to ensure proper exchange of wiki metadata. In this respect, the WMO is the “common wiki data model” described in the proposal.

4.2. Problem Description

Both traditional and semantic wiki engines provide a wide range of features and approaches that suit best one or another situation. Hence, there is no "perfect" wiki for everyone. However, whatever a user's preference may be, it is important for the the differnet wikis to be able to "talk to each other", so that information from one wiki is accessible and editable in another wiki.

There are many conceptual ways to create semantic wikis. The most popular semantic wiki types are:

- **Page-centric**: Each page models one concept, wiki syntax is used to annotate this concept. Content and semantic content form one “body of knowledge”.
- **RDF-centric:** Each page is a container for RDF statements, expressed in a convenient wiki syntax. Often the page itself can also easily be addressed via the syntax.
- **Annotation-centric:** Users can use additional parts of the user interface which are not present in normal wikis to annotate wiki pages semantically. The annotations have no representation in wiki syntax and are e.g. lost when the page wiki source code is e-mailed.

Semantic wikis also vary along other dimensions:

- Is wiki syntax used to make semantic statements? What is exposed as semantic data?
- Is the integration with a different engine or application or with another instance of the same engine? The latter can e.g. be solved with specific APIs.

- How is the wiki deployed and used? As multi-user collaboration portal on a central server or as a single-user authoring tool on each desktop?
- What is the type of application? A web-application or a desktop application? This influences the offered API: Only HTTP or also a more convenient API for programmatic access, e.g. a Java API.

Page name, URL and URI. All semantic wikis have to solve the problem of naming. Since wikis usually use a web interface, each page has a URL. Additionally, each wiki page has a wiki name, which is used within the wiki to create links to other pages, e.g. by putting square brackets around a page name. Finally, in a semantic wiki, the concept described by the page has a URI which is used in RDF as an identifier. All these three ways of naming have to co-exists and in particular the page URL and the concept URI should not be confused.

In the section below we describe some concrete scenarios and specify the functionalities required to carry out these scenarios.

4.2.1. Integration Scenarios

"Dirk" is the name of the hypothetical user in the scenarios below.

4.2.1.1 Content aggregation

Background: There is a single client application which can use transparently different wiki engines at the same time. This is accomplished by using the same API from different wikis.

- Dirk opens a client application showing an aggregated view of the content of two different wiki engines.
- At the same time Dirk can open pages from either one wiki engine.

Required functionality: Content export

4.2.1.2 Semantic content migration

- Dirk opens a wiki page in wiki A
- He creates a test page ("MyTest"). This page can contain some semantic information.
- He goes to the "export" page and selects "export all" option and click "ok" button.
- The system proposes to download a file containing the whole content of the wiki A and to store it on the local storage. Dirk chooses a temporary folder and stores the content on the disk.
- Dirk opens wiki B
- Dirk tries to open "MyTest" page and the engine says that there is no page with such a name.
- Dirk goes to the "import" page and uploads the locally stored file with the content of wiki A

- Dirk tries to open "MyTest" page again. And now he/she can see that this page exists and it contains the whole information created using wiki A.

Required functionality: Semantic content export, Semantic content import

4.2.1.3 Semantic Desktop Integration

- Dirk opens the project overview in his semantic desktop
- Dirk clicks on "view wiki"
- Wiki A opens with the corresponding wiki page
- Dirk adds some comments and relations to other people
- He goes back to his semantic desktop
- Semantic desktop should show the new relations
- Dirk configures wiki B as his wiki engine
- Again, Dirk clicks on "view wiki"
- Wiki B opens with the corresponding wiki page (which might look different from the page shown in wiki A)
- Dirk adds some comments and relations to other people
- He goes back to his semantic desktop
- Semantic desktop should show the new relations

Required wiki functionality: Semantic data export / import

4.2.2. Wiki and Semantic Wiki Architecture

Wiki Architecture

In order to explain how wikis can be integrated we have to look how wikis are implemented. Traditional wikis are implemented as follows:

- The back-end stores a set of text files in wiki syntax
- When a users edits, the changed file is stored on the disk
- When a user views a page, the file is loaded from the hard disk and the content is post-processes via a carefully crafted set of regular expression (text re-writing patterns) into HTML syntax.

The important fact is, most wiki engines have therefore no internal representation of the page's structure. One could say "the wiki doesn't know what it knows". The often encountered back-links, a very important navigation feature of wikis, is often implemented via a direct full-text search in the text files.

Semantic wiki architecture

In this case, the classic model no longer scales. At least when the RDF export is triggered, the wiki has to render the page source code into RDF. This could still be implemented in the classic way by using regular expressions. But a soon as the user wants to pose queries to the wiki, the wiki engine has to know all RDF data at the same time. Therefore, most semantic wikis are implemented entirely different from classic wikis. A semantic wiki typically:

- Parses the wiki syntax edited by the user
- Stores the content in a structured way

- Stores RDF data separate from the page content
- Is able to answer RDF queries (e.g. in SPARQL)
- Computes back-links by fast RDF queries, not by time-consuming full-text searches.

4.3. Requirements

A semantic wiki has *three* assets of information that can be integrated with other applications. First, the pure content, like in traditional wikis. Such content can be modelled as XHTML.

Second, the semantic data, which can be exported as RDF. However, semantic data in a semantic wiki consist of two sub-parts: 1) Metadata about the page, e.g. the author, the modification date, or the previous version of a page. Such data is usually created automatically by the system; 2) Semantic data created by the page author via syntax constructs explicitly, e.g. this person is a member of this project or this city is the destination of my trip in August.

Third, the two pieces of interwoven information can be exported/imported as a single unit. There is currently no content format defined for this, but there some approaches are in development.

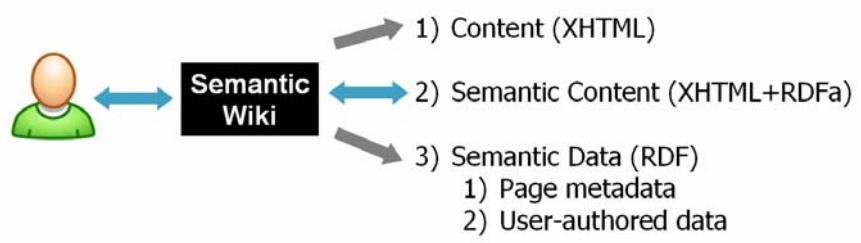

Figure 15. Semantic Wiki Integration Architecture

Based on the layering depicted in Fig. 15, we can classify the possible integration use cases as follows:

4.3.1. Content

Content is the visible part of a wiki page. It contains text, tables, images, etc.

Export

- Export the content of a wiki page or several wiki pages to another content format, e.g. MS Word, OpenOffice, HTML or another wiki syntax.
- Email the content of a page to a colleague
- Print the content of a wiki page with page numbers
- Import
- Import a single wiki page content from e.g. an MS Word file or another (semantic) wiki
- Import an email into a wiki page or as a new wiki page

Note that semantic data is lost, when page content is overwritten with a non-annotated version.

4.3.2. Semantic Data

Semantic data is composed of two kinds of data:

- Page metadata – this is controlled by the application (the wiki engine) and can never be changed by the user. It contains data such as the author of page, the change data or the version number.
- Authored semantic data – this is expressed using special syntax elements in semantic wikis. Such data can be change by the user.

Export

Export a page or all pages' RDF data, and import into an RDF store. Combine it with other data. This can e.g. be used to let the wiki be an ontology editor.

Import

There are two kinds of import:

- Import annotations about the page made by external tools as semantic data
- Import metadata created e.g. by NLP tools

Note that this feature can in some wikis only be implemented by creating wiki syntax that states the same RDF as given. Overwriting the page with such a generated syntax will delete the page content. Alternatively, one could append the generated wiki syntax to the existing page content.

4.3.3. Semantic Content

Semantic content models the wiki content plus the semantic statements derived from it, without loosing the connection what was generated from where. E.g. the page content contains the name "Paul", which is also used to generate a triple that (Paul, foaf:knows, Alice). In semantic content one would still have the link which HTML-part refers to which RDF-resource.

Export

Export a semantic page with content and metadata to another semantic wiki

Import

Import a semantic page with content and metadata from another semantic wiki

Note: Only importing content and semantic data as one data chunk makes updates to semantic wiki pages possible without overwriting content or semantic data.

4.4. Solutions

To sum up, we have three kinds of players in our integration scenarios:

- Wikis,
- Semantic wikis, and
- Semantic desktop applications

We present four solutions to the problem of wiki integration.

First, we present an ontology to describe the metadata of wikis and semantic wikis. This ontology is shared among all integration players (wikis ,semantic wikis and desktop applications).

Second, we present the Common Semantic Wiki API, which is use din NEPOMUK Milestone 1 to integrate three different wikis with he semantic desktop middleware. This is tailored for integration with semantic desktop applications.

Third, we present WikiModel, a re-usable Java component for parsing (semantic) wiki syntax and modelling (semantic) wiki content. This is usable, but not limited to, integration semantic desktop applications.

Fourth, we present the Wiki Interchange Format (WIF), which can be used to exchange data between wikis and semantic wikis. This is mainly useful for integration of non-semantic wikis or external semantic wikis with local desktop applications - or with other non-semantic or external semantic wikis.

4.5. Solution: Wiki Metadata Ontology

The wiki metadata ontology describes how to encode the essential characteristics of wiki pages into RDF. The page content is intentionally left out, as this ontology models only the metadata.

The WMO is used by WIF and CSWA, which are described in later sections. The ontology is available online at [http://wif.ontoware.org/2005/04/index.html].

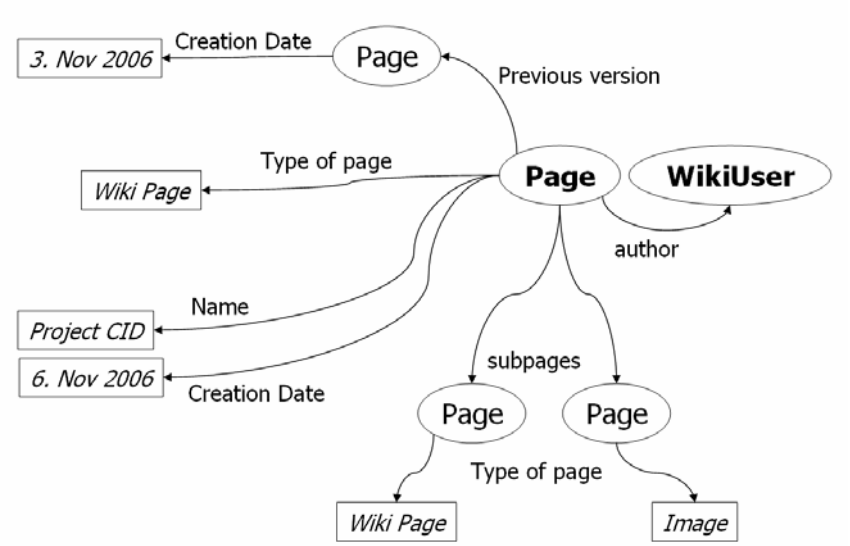

Figure 16. High-level view on wiki pages, users and relations

As you can see in Figure 16, we model a wiki as a set of Pages. Each page has a number of relations and attributes. Each page has a previous version. Each page has an author, the name of the page, and a creation date. When a page is change, we consider it as a new page, pointing to it's previous version.

We also model the type of page, which can be wiki-page, image or any other media type, as we model binary files also as "pages". Attachments to pages and images are modelled as sub-pages. The page type distinguishes the different possibilities.

A refined view on the *complete* WMO can be seen in Figure 17.

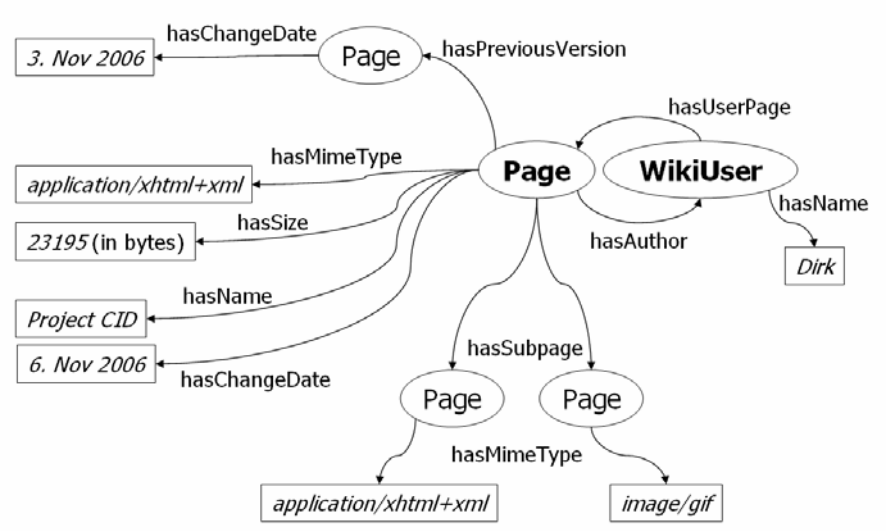

Figure 17. The complete Wiki Metadata Ontology

4.5.1. Ontology Classes

class WikiUser — A person/agent who authored items

- URI: http://wif.ontoware.org/2005/04#WikiUser

class Page — The basic building block of wikis. models wiki pages, images and attachments

- URI: http://wif.ontoware.org/2005/04#Page

4.5.2. Ontology Properties

property hasUserPage — Models users in a wiki. Typically each registered user in a wiki has a dedicated wiki page.

- URI: http://wif.ontoware.org/2005/04#hasUserPage
- domain: WikiUser
- range: Page

property hasMimeType — Assigns a mime-type to a Page. This allows to model wiki pages, images and attachments with the same class. The mime type tells an application how to deal with the content. The same approach is used by the WWW.

- URI: http://wif.ontoware.org/2005/04#hasMimeType
- domain: Page

property hasName — Assigns a unique name to pages and users. This property is mapped to pimos:hasWikiname.

- superproperties: pimos:hasWikiname, pimos:hasTerm, dc:title
- subproperties: pimos:hasWikiname
- URI: http://wif.ontoware.org/2005/04#hasName
- domain: Page, WikiUser

property hasSize — The size in bytes

- URI: http://wif.ontoware.org/2005/04#hasSize
- Label: size
- domain: Page

property has Author — Each wiki page has an author, which is a wiki user

- URI: http://wif.ontoware.org/2005/04#hasAuthor
- Label: author
- domain: Page
- range: WikiUser

property hasPreviousVersion

Minimal versioning assigns each page it's predecessor. Transitive usage of this property allows to reconstruct the whole versioning history of a page.

- URI: http://wif.ontoware.org/2005/04#hasPreviousVersion
- Label: previousversion
- domain: Page
- range: Page

property hasSubpage — Links a page to images, sub-pages and attachments

- URI: http://wif.ontoware.org/2005/04#hasSubpage
- Label: subpages
- domain: Page
- range: Page

property hasChangeDate — As each change to a wiki page results in a new version, this property models the creation date of a version.

- URI: http://wif.ontoware.org/2005/04#hasChangeDate
- Label: changedate
- domain: Page
- range: xsd:date

4.6. Solution: Common Semantic Wiki API (CSWA)

The semantic wiki API is a first step towards an API targeted for

- Semantic wikis on a desktop
- Accessed via a programmatic API
- Integrated with other semantic applications

Thus the CSWA has special requirements, different from a completely generic (semantic) wiki interchange.

We developed an API that allows to solve scenario 3 (see Sec. 4.2.1).

4.6.1. RDF operations

String getURIforPageWithName(String pagename);

- Return the URI which represents in RDF the concept described on the page. E. g. for a page named "Berlin", we have a URL http:/example.com/Berlin, which describes the wiki page Berlin, and we have a URI, e.g. http://example.com/id/Berlin which is the concept Berlin. Note: An http-range-14-compliant wiki may not return HTTP 200 OK for an HTTP GET on http://example.com/id/Berlin. Instead, an 303 redirect to http://example.com/Berlin should be used.
- There might be multiple concepts described with a page. But then, we have no longer a clear mapping from document to concept, so then one should rather.
- This method should return null, if the wikipage does not describe exactly one single concept.

String[] getPagesRelatedToResource(String uri);

Return the URIs of all pages that represent/contain statements about a certain resource. In Semantic Pad, this is simply the page URI itself. In Kaukolu, this is realised with a semantic query.

INepomukTriple[] getStatementsOnPage(String pagename);

Note: The INepomukTriple means the triple type supplied by the NEPOMUK middleware. At the time of writing, the exact binding to the Java language was not clear.

This method returns statements contained in a page as an arrayThe metadata of the page should be attached to the URI returned by getPageURI. The metadata should be described according to the WMO.

For a wikipage named "SandBox" with the url "http://example.com/wiki/SandBox" and the URI for the concept of a sandbox "http://example.com/concept/SandBox", the following triples should be in the statement array as well. Note how we distinguish the concept of the wiki page and the wiki page as a document:

```
<http://example.com/concept/SandBox>
 rdf:type pimos:Thing .
 pimos:hasWikiname “SandBox”
 foaf:page <http://example.com/wiki/SandBox> .

<http://example.com/wiki/SandBox>
 rdf:type wif:WikiPage .
 wif:hasAuthor <URI-of-author> .
 wif:hasChangedDate “2006-04-12”^xsd:date .

INepomukTriple[] findStatements(
 String subjectURI, String predicateURI, String
objectURI);
```

Search statements in the wiki, i.e. treat all RDF of the wiki like one triple store.

4.6.2. Content Management System operations

`boolean pageExists(String pagename);`

Check whether page exists.

`void deletePage(String pagename);`

Remove an entire page from the repository.

4.6.3. User Interface Integration

`String getEditUrl(String pagename);`

Get the url to edit the wiki page.

`String getViewUrl(String pagename);`

Get the url to view the wiki page.

`String getMinimalViewUrl(String pagename);`

Get the url for a minimal (embedable) view the wiki page (no sidebar, logo, etc.). This function is used by semantic desktop applications that show the wiki with an embedded browser widget.

`String getPageMarkup(String pagename);`

Get wiki markup of a page.

`void putPageText(String pagename, String text);`

Save page markup. Throws an exception if page does not exist.

4.7. Solution: WikiModel – a re-usable Semantic Wiki component

A first step towards semantic wiki integration is the creation of an API which abstracts away from specific syntax features. *The WikiModel consist of a configurable parser which parses wiki syntax into a stream of wiki events (wiki event model: WEM) which can be used to build an in-memory representation of wiki content in the wiki object model (WOM).* This goes beyond the integration scenarios and will serve as a building block for future semantic wikis.

The Java WikiModel framework consists of:

DataModel – defines the grammar of wiki documents. The main idea is that each document consists of a sequence of marked blocks named sections. This notion of sections can be easily mapped to property definition in RDF. So the semantic model, as it defined in the current version of the WikiModel defines each wiki document as a subject of statements, labelled sections corresponds to RDF predicates of statements and the content of sections are statement objects.
The wiki data model defines that each document section in turn can contain block elements (like headers, paragraphs, tables, list...). Each block element contains a set of formatted in-line elements (text, images, references...). The formatting is defined using styles aggregating different characteristics applied for the text (like bold, italic, subscript, superscript and so on).

Wiki Event Model (WEM) is an event-based API defining the interface of event listeners used to notify about individual structural elements in wiki pages (ex: beginParagraph(...)/endParagraph(...)). This API was inspired by the standard Java SAX API [http://www.saxproject.org/].

Wiki Object Model (WOM) - defines interfaces of in-memory object representation of individual structural elements

Syntax Definition – a particular wiki syntax. This syntax was developed taking into account results of analysis of various popular wiki syntaxes. Some features of this syntax:

- The syntax natively includes definition of semantic elements in wiki pages.
- This syntax is build using a set of simple rules: structural elements in wiki pages are defined using a two-letter sequence of special characters.
- Only the easy available special characters existing in most (in all) keyboard layouts are used to define commonly occurred structural elements (like headers, lists, tables). This is very important point ignored by most wiki syntaxes. For example the commonly used symbol "|" for tables is not available in many eastern keyboard layouts. In our syntax the sequence of ";;" and "::" is used instead. (But the "|" symbol should be recognized as well)
- Each document can contain "embedded documents". Each embedded document can contain their own block and inline elements. For example, it allows users to create tables containing lists, headers or another tables using only this syntax. This feature is unique for the JavaWikiModel. This functionality is not available in other wiki engines.
- Almost all structural elements in wiki pages can have their own parameters defined in the form of key/value pairs. These parameters can be used by developers to generate specific formatting as well to include additional (for example – semantic or security) meta-information to all parts of wiki pages.

Parser – a real JavaCC-based parser (see [https://javacc.dev.java.net/]) recognizing the JavaWikiModel syntax and generating a well-formed sequence of events notifying a WikiEventModel (WEM) listeners about individual structural elements of parsed wiki pages.

WikiSyntaxSerializer used to generate a well-formed wiki document from a sequence of events

Utility tools:

- WEM to WOM (and WOM to WEM) transformers used to translate a sequence of events to the object representation of a wiki document and vice versa
- An XML serializer and de-serializer used to generate and read XML documents containing the exact structure of wiki documents.
- An HTML generator used to create HTML from a sequence of events (WEM)
- An HTML importer used to transform an arbitrary HTML document to a well-formed sequence of events (WEM)

The Java Wiki Model can be used as a complete solution to parse, validate and re-construct well-formed wiki documents containing semantic information.

At present, the WikiModel is used as one of the main blocks in the Semantic Pad and it was integrated in XWiki to add semantic aspects to this application.

In the future, WikiModel is intended to be WIF-compliant, which is described in the next section.

4.8. Solution: Wiki Interchange Format

A more generic format for wiki interchange is the *Wiki Interchange Format* (WIF). WIF is neither restricted to semantic wikis, nor to a particular platform. The current status of WIF can be found on a dedicated WIF homepage at http://wif.ontoware.org.

WIF is intended to work for classic and semantic wikis alike. It consists of two levels:

- On level 1, the wiki page content is modelled and can be exchanged. This is basically an XHTML subset. A subset, because transforming this via XSLT to another wiki syntax becomes much easier. The full HTML spec contains many elements not used by any wiki syntax.
- Level 2 uses RDFa5 to embed semantic data into the content. The two levels together allow full round-tripping.

The main idea of WIF is to represent the semantic content (content + semantic data + page metadata) in a single information chunk, which can be exported. In a nutshell, WIF uses

- A subset of XHTML to represent the content. A subset is chosen, because this allows to write XSLTs easily which can render the page content in other wiki syntaxes or content formats. The whole XHTML structure contains many elements not needed by wiki engines and most of them are also not used. WIF contains the most-often used elements only.
- RDF to express page metadata and user-stated semantic data. In order to express page metadata, WIF uses the Wiki Metadata Ontology described in the next section.
- RDFa is an emerging W3C recommendation to glue XHTML and RDF together. RDF is represented as attributes (hence the name RDFa) of an XML file.

WIF also defines a serialisation format, i.e. how to store a single WIF file or a set of WIF files (called the Wiki Archive Format, WAF) on disk or send over an HTTP connection. In the future, an HTTP-based API for WIF is also planned.

[5] See http://RDFa.info – RDFa is a W3C proposal for embedding RDF into arbitrary XML, by encoding RDF triples as XML attributes. XML elements can play a double roles as XMLLiterals in RDF.

Currently, an Java API for WIF is developed at FZI. Adapters for MediaWiki and JSPWiki are initially planned to disseminate to the classic wiki community. In the future, Semantic MediaWiki, Kaukolu and Semantic Pad adapters are planned.

5. Language processing and semantic annotation

Semantic annotation is the key functionality that differentiates semantic wikis from the non-semantic ones. Although such annotation provides clear benefits to users (e.g., improved information retrieval, analysis, and sharing) it also requires an additional effort on their part. Reducing this effort is just as important for the semantic wiki success as increasing the benefits. Language processing algorythms can be of significant help in automating (at least, partially) the annotation process.

5.1. The Language Processor API

Due to architecture requirements the decision was made to consolidate the design of the semi-automatic semantic analysis component (WP1) and the text mining and semantic extractor component (WP2) into one Text Miner and Semantic Analysis component. This component complies with architecture and APIs as outlined in D1.1, D2.1 and D6.1. WP2 will access this new combined component for its text mining functions through the middleware layer in accordance to the architected APIs.

This component will provide the following services:

- Language processing — linguistic processing of natural language text documents and extraction of information from such documents.
- Semantic processing — analysis of topic and context of documents in relation to personal or domain-specific ontologies.

5.2. The Language Processor API

The main goal of the Language Processor API is to provide optional linguistic analysis of text documents for the Semantic Processor by mapping lexical expressions to concepts.

In addition, the following specialised linguistic processing and language generation functionalities are available as services:

- Identification of the most important keywords in a document.
- Identification of speech acts within a given text. Speech acts consist of questions, statements, orders, requests for permission, requests for information or an action, etc.
- Identification of main semantic entity types in text without using explicit semantic knowledge bases.
- Semantic annotation of text using Controlled Natural Language (unambiguous machine processable natural language — in this case a form of simplified English).
- Ontology authoring and instance population using Controlled Natural Language.
- Text generation from ontologies using Controlled Natural Language.

5.2.1. Description

Optional Linguistic Analysis can be provided for the Semantic Processor using the following GATE [13] NLP Framework processing resources or IBM LanguageWare [14] lexical analyzer:

- A Tokenizer for splitting sentences into tokens such as words, numbers, punctuation.
- A Sentence Splitter, which segments text into sentences.
- A Part of Speech (Pos) Tagger, which assigns a linguistic category to each word.
- A Semantic Tagger, which when targeted to a given ontology will identify ontological entities within the text. This is achieved through the use of Shallow Parsing and word list or gazetteer lookup.
- Finally, a semantically annotated text can be passed to the Semantic Processor for further disambiguation and semantic processing.

The above processing resources are each reused at stages within the following services provided by the Language Processor, specifically 5.2.2, 5.2.3, 5.2.5, 5.2.6 and 5.2.7.

5.2.2. Keyword Extraction

There are three aspects of this functionality, all based on the same basic approach. Given a string of text, the service performs tokenization, sentence splitting and part of speech tagging. Once completed, stop word removal is performed (*stop words* are those words which are so common that they are useless to index or use in search engines, i.e. *a*, *the*, *in*, *of*). The Porter Stemmer algorithm (see [14]) is also applied. Stemming involves the reduction of a word to its base form, i.e. *cats* ≤ *cat*}.

From this point the data is manipulated in three different ways:[6]

- The service computes the ten most frequently used terms in the string. In this approach a term is considered to be a single word.
- The service also performs noun phrase chunking on the string. Noun phrase chunking deals with extracting the noun phrases from a sentence. It then finds the ten most frequently used noun phrases in the string.
- Based on the approach outlined in [15], collocation analysis using Pearson's Chi squared [16] is performed on the document (within the area of corpus linguistics, collocation is defined as a sequence of words or terms which co-occur more often than would be expected by chance). The algorithm attempts to reject the hypothesis that two noun phrases which commonly occur

[6] This service has been provided by Hewlett-Packard Galway.

together are good keywords. This approach outputs the ten most probable keywords.

5.2.3. Speech Act identification

For a given text, this service will return a set of Speech Act Annotations. Text is annotated at the sentential level. Shallow Parsing methods are used, given a set of hand-coded grammars to annotate sentences within the text as instances of speech acts. In addition a collection of hand written gazetteers or lists of words is used to aid the parsing process [17, 18].

5.2.4. WebLearn

WebLearn identifies entity candidates in a given text and extracts for each the main semantic type based on the information available on the Web, without use of explicit semantic knowledge base, i.e. gazetteer lists, grammar rules etc. The procedure is implemented along the lines of [19].

Examples:

entity: Texas ⇇ type: state

entity: George Bush ⇇ type: president

Procedure:

- identification of instance candidates (entities) in text based on shallow NLP (patterns over part-of-speech tags).
- generation of exact queries with instance candidates based on Hearst-Patterns [20] (“George Bush is a *”, “* such as George Bush”, etc).
- with these queries, a Web search is invoked, based on the search engines Yahoo and Google Web services, harvesting returned snippets.
- shallow annotation (part-of-speech tags, lemmatization) of relevant context of harvested snippets.
- statistical analysis of extracted candidates, probability ranking and proposal of the best candidate as the semantic type.

5.2.5. Semantic Annotation using Controlled Natural Language

A form of Simplified English or Controlled (Natural) Language or CL is used to annotated instances of concepts with free text given an ontology. When editing a Wiki page the user will be capable, upon encountering an instance of a concept in an ontology, escape into an annotation mode using a special reserved character within the Wiki edit syntax. Then the user will then use CL to annotate the instance left most to the escape character.

```
Brian [is a Person] is a researcher at Galway
University.
```

The CL text “Brian is a person” will then be translated into semantic data, guided by the underlying ontology. The semantic data or *metadata* is

anchored to the position in the text — consequently it is a semantic annotation. The CL syntax analyzer in conjunction with the ontology will guide the user with respect to acceptable CL input. Upon given a text with CL comments or annotations on input, the user will receive a semantically annotated text in return [21].

5.2.6. Ontology authoring using Controlled Natural Language

A form of Simplified English or Controlled (Natural) Language (CL) is used to create and author an ontology. In addition, instances of concepts can be described to populate the ontology using CL. The CL text will then be translated into the targeted underlying ontology. The input will consist of CL text input only and the type of ontology to be targeted. The output will consist of a reference to newly created and populated ontology. The CL syntax analyzer in conjunction with the in vivo ontology will guide the user with respect to valid CL input [22].

5.2.7. Text generation of Ontologies using Controlled Natural Language

Given a reference to an ontology the inverse of 5.2.6 will be applied. Thus, a text summary of a given ontology generated with CL and its populated instances will be returned to the user. This will involve: 1) traversing the ontology and generating CL text given a selection of XML templates; and 2) the resultant text will contain snippets of CL that will, given a concept or instance, combine into a CL sentence and finally a CL based textual summary of the ontology. This process in conjunction with 5.2.5 can be used to edit a CL textual summary of an existing ontology in order to regenerate or amend/edit an existing ontology repeatedly to achieve the desired output. The process of merging 5.2.6 and 5.2.7 is called round-trip ontology authoring [22].

5.3. The Semantic Processor API

The Semantic Processor is used for processing text documents by finding their topics in relation to given domain ontologies and disambiguating words based on their usage context and the ontologies.

The lexico-semantic processing combines linguistic analysis and mapping between lexical expressions and concepts based on semantic information about concepts and their relations.

The analysis is performed in relation to an ontology that is represented as a graph. Each node in a graph describes a unique concept or entity, either abstract (e.g., category or type) or specific (e.g., particular person or place). Edges in the graph represent relations between concepts.

The semantic analysis involves mapping between lexical expressions (words, phrases) that appear in natural text and the corresponding concepts. Thus, it can handle high level of ambiguity found in natural languages and some areas of knowledge, when concepts may have alternative names (including names in different languages), or names of different concepts (or even unrelated words) have identical spelling.

The semantic processor provides the following functionality:

- *Find a topic or focus of an analyzed text document*. The focus is a set of concepts from an ontology which are found to be the

most central to the document. The focal concepts may not be directly referred in the text, but instead implied from context. The focus of a document may be used to perform classification or tagging of documents, or locate the most important keywords related to the document (even those keywords which did not appear in the text).

- *Perform automatic disambiguation of the mentioned terms*. The result is the mapping from words or phrases in the text to the corresponding concepts in the ontology. In GUI applications, this functionality may be used to provide hyper-links to descriptions of disambiguated words.

6. Evaluation

The most direct way to evaluate semantic wikis with respect to what they could do for case study users would be to ask representative users from the four groups to try out the wiki engines (and apply traditional HCI evaluation methods while they do so). However, there are a number of problems with that approach:

- Our semantic wikis are only prototype level tools. This means users would likely experience a lot of problems with the software – making it hard for users to appreciate and for us to evaluate the semantic features.
- There's no semantic wiki instance known to us, which contains data relevant to any of the case study settings. Either we would need to put a lot of effort into the creation of such a knowledge base, or users would be required to perform tasks in a domain not familiar to them.
- Some features of semantic wikis (such as writing complex queries) require some education before they can be appreciated or even used in a proper way.

The initial evaluation was therefore conducted by Nepomuk researchers themselves, although not those directly involved in the wiki development. We have based the evaluation on the information about the case study users that has been accumulated both in the minds of Nepomuk researchers and in external documentation in the form of personas and scenarios (see Nepomuk deliverables D8.1, D9.1, D10.1, and D11.1). The scenarios used in the evaluation are described in the next section.

6.1. Evaluation Scenarios

The scenarios below are based on the scenarios developed during in-depth field studies of the four Nepomuk case environments. Note: The names ("André", "Kim", "Karen", "Claudia" etc.) are not names of real persons, but names of fictional *personas* which have been created in the case studies, based on our interviews with and observations of several users. For more information on the case study methodology, see deliverables from Nepomuk workpackages 8 to 11.

The scenarios reflect needs of the users within the target audiences - knowledge workers in four different domains. However, they have been chosen to be relevant to the semantic wiki technology. For each scenario below, the relevant functional requirements are specified in addition to the description. These requirements have also been identified in the course of the case studies.

6.1.1. Scenario 1: Advanced querying of Semantic Helpdesk

From Mandriva case

Scenario description: André is pretty sure that the answer to his current problem (how to install his new graphics card) is available somewhere on the wiki, but he cannot find it using the traditional wiki tools. Either he finds too much information (when only providing some keywords related to his problem) or he finds no information at all (when providing

keywords describing his system configuration, his problem, his situation). With the semantic wiki, however, André can query the helpdesk with formal constraints on system configuration and situation description and find the most relevant information.

Relevant functional requirements as expressed in the case studies:

semantic search

6.1.2. Scenario 2: Ontology enhanced search in Semantic Helpdesk

From Mandriva case

Scenario description, version A (synonyms): Kim searches for an answer to his current problem (how to install his new graphics card). Unfortunately, the name of the maker of the graphics card changed recently, and all items on the wiki that contain the answer to Kims question mention the old name. Fortunately, in the semantic wiki, somebody entered a page describing the maker (under the new name), and on this page there is a semantic link (SameAs: Oldname) which tells the system that the two names are actually referring to the same company. Therefore, the search function can provide Kim with the answer he needs, even though the search term was not present in the page.

Scenario description, version B (subconcepts): Kim searches for an answer to his current problem (how to install his new graphics card). Unfortunately, since the graphics card is brand new, nobody has yet written any problem report or how-to document on the wiki containing the name of the new graphics card. Fortunately, in the semantic wiki, somebody has entered a page describing the new graphics card. This page includes a semantic statement saying that the card is a kind of "superCard" (isA: superCard). There are many how-to documents in the wiki describing peculiarities of superCards. Therefore, the search function can provide Kim with the answer he needs, even though the search term was not present in the page.

Relevant functional requirements as expressed in the case studies:

- semantic annotation
- semantic navigation
- semantic search

6.1.3. Scenario 3: Wiki with timeline view to support report writing

From SAP case

Scenario description: As a project leader Claudia is responsible for the deliverables in many projects. The deadline is approaching in one of the projects and she needs to get things done. Claudia can see in her task list that the deliverable is only two weeks away. To get an overview of all relevant information to be reported in the deliverable, Claudia searches the system for anything related to the project, and switches to the timeline view. Here she can see what documents have been created when, what meetings have been held, and what is being planned in the future. This is a convenient way of finding relevant documents and dates which should appear in the report.

Relevant functional requirements as expressed in the case studies:

- perspectives
- versioning / differentiation
- semantic annotation
- semi-structured forms
- semantic navigation
- semantic search
- templates (e.g., task creation)

6.1.4. Scenario 4: Annotate resources for sharing

From TMI case

Scenario description: Karen is in the office on a Friday and it is time to take care of some administration, attend meetings and such, and like any other day this day is really busy. She has just finished a successful project that she believes should be shared with her colleagues. She opens the course material and marks it “Shared” and adds a few keywords to make sure that people interested in fitting anecdotes for IT and Telecom companies can find the material. Now everyone at TMI – especially those who have chosen to focus on IT and Telecom – become aware that the material exists, can learn from her experience and re-use her material.

Relevant functional requirements as expressed in the case studies:

- semantic annotation
- semantic navigation

6.1.5. Scenario 5: Project browsing

From Institut Pasteur case

Scenario description: When looking through a project Marie can browse the information in a variety of ways. She can look at lists of project participants, experiments composing the project, problems found in the project, urgent tasks, etc. When she looks at the urgent tasks list, she opens one task and she can add herself as a participant to the task (with a link to her wiki page). Her name gets automatically formatted into a list according to the task template. She then clicks on her name and looks at the back-links of type "InvolvedInTask" on the page devoted to her - there she finds all the tasks in which she is involved.

Relevant functional requirements as expressed in the case studies:

- perspectives
- semantic annotation
- semantic navigation

templates (e.g., task creation)

6.2. Evaluation results

The focus of the evaluations has been from the perspective of the end users, although the system administration and integration aspects have

also been considered. The developed prototypes offer interesting architectural solutions and already at the early stage satisfy a number of important user requirements collected in the Nepomuk case studies. Nevertheless, much work still remains to be done.

6.2.1. Semantic Pad

6.2.1.1 General impressions

Semantic Pad was easy to install, which is remarkable for prototype software.

Two features of Semantic Pad immediately stand out:

- The clean design.
- The idea of making templates and customized views.

The first feature goes well with the conclusions of section 2.1 (wiki success factors). The second feature also has some interesting potential. One of the goals of semantic web applications is to provide end users with advanced functionalities tailored to specific situations, without the need for expensive and not-so-flexible development of business systems. Giving wiki users the ability to make templates and views suitable to their specific needs, if successful, is halfway to that goal. The Semantic web can be considered a revolution if an end user can create semantic wiki templates and views (which could include business logic) instead of hiring a consultant to adapt a commercial business system upwards of €100.000!

Right now, Semantic Pad is not directly useful for a normal end user. Creating templates requires programming skills, and the README file accompanying the distribution (not inlined in the default wiki startpage) is not yet easy to understand. However, in-house or external system integrators can easily configure Semantic Pad into a user-friendly application that satisfies user needs in a given knowledge domain.

The Semantic Pad differs in its basic functionality from ordinary non-semantic wiki systems. For example, there is a "navigator" which displays a table of system resources, the purpose of which is not clear. This is a drawback, but with some instructions added into the editing page it will likely not be a big problem.

6.2.1.2 Ability to be used in case study scenarios

Scenario 1: Advanced querying of Semantic Helpdesk: Since there is not yet (November 2006), any interface for advanced queries, this scenario cannot be realized with Semantic Pad.

Scenario 2: Ontology enhanced search in Semantic Helpdesk: Since there is not, as of today, any search function at all, these scenarios cannot be realized with Semantic Pad.

Scenario 3: Wiki with timeline view to support report writing: This scenario cannot be realized today, since it requires a timeline widget, desktop integration and a query interface. Desktop integration should be available rather soon, though, and the other requirements may not be very far away either.

Scenario 4: Annotate resources for sharing: There's no support for sharing in today's version of Semantic Pad, but it doesn't seem like a difficult feature to implement in any semantic wiki system.

Scenario 5: Project browsing: As has been already stated, the specialized views feature is one of the strengths of Semantic Pad. It is possible for an advanced user or a systems integrator to set up Semantic Pad so that this scenario can be realized.

6.2.2. Kaukolu

6.2.2.1 General impressions

Kaukolu wiki has been tested in two different modes:

1. standalone semantic wiki system
2. tightly integrated with Gnowsis 0.9.0 (and briefly 0.9.3).

In both modes, installation was extremely simple.

In both modes, editing non-semantic data was easy. If the user can edit any normal wiki, then he or she should be able to edit Kaukolu wiki pages as well. There are instructions on how to write wiki code, and the welcoming page even features a link to a one-minute guide for users who are not familiar with the wiki concept.

Editing semantic data, on the other hand, is not that easy and requires some training. There are some instructions on how to edit triples, how to import ontologies, and how to use autocompletion for triples, but there is no guide for the user who does not know what a triple is.

In the standalone Kaukolu wiki there is no obvious instant reward for adding semantic statements to the wiki. The Kaukolu wiki integrated with Gnowsis functions as the concept editor for Gnowsis (i.e., semantic efforts in Kaukolu may be rewarding in the bigger context of the semantic desktop).

The autocompletion function, while definitely a very desirable function, does not work well in any of the browsers used during testing. Also, since there is no Kaukolu instance available containing any volume of information except for the SandBox and some introductory pages, it has been hard to get a feeling for Kaukolu's semantic features. The general experience of using Kaukolu is that it is a tool primarily developed for the purpose of researchers exploring their theoretical ideas, not to be widely used in everyday life by, for example the knowledge workers of Nepomuk case studies.

6.2.2.2 Ability to be used in case study scenarios

Scenario 1: Advanced querying of Semantic Helpdesk: Since there is not yet (November 2006) any interface for advanced queries, this scenario cannot be realized with Kaukolu wiki.

Scenario 2: Ontology enhanced search in Semantic Helpdesk: Since there is, as of today, no semantic search function, these scenarios cannot be realized with Kaukolu. Once there is, it shouldn't be very hard to implement. All the necessary relations can be expressed in Kaukolu.

Scenario 3: Wiki with timeline view to support report writing: Kaukolu is the only wiki with some desktop integration today. It is not a gigantic task to implement a wiki view of desktop data (e.g., file system) with Kaukolu. However, Kaukolu does not feature a timeline widget, and it does not yet have the query functionality, which would also be needed for this scenario.

Scenario 4: Annotate resources for sharing: Possibly, in the case where Kaukolu is integrated with Gnowsis, a user can setup a search for content with a "Shared" tag, which belongs to a certain person. Otherwise, since search functionality is limited, Kaukolu does not support this scenario today, but the necessary functionality should not be difficult to implement.

Scenario 5: Project browsing: Kaukolu does not offer a specialized "task view" and the search functionality is limited to traditional free text search. Querying based on semantic data is within reach, but having different views is not the current focus of Kaukolu development.

6.2.3. Semantic MediaWiki

6.2.3.1 General impressions

Contrary to the above semantic wiki systems, SMW thus has the appeal of being an actively used tool where we have been able to look at some real world data being used for interesting purposes. For this reason we did not install Semantic MediaWiki (SMW), but instead used the real world active installation publicly available at http://ontoworld.org/wiki.

Just like Kaukolu, Semantic MediaWiki can be used as any ordinary wiki, ignoring the semantic features. We see this as a great advantage, since it lowers the barrier of entry, which increases the chances of the system being used by people other than researchers and people with a technical interest in the system itself.

Adding semantic statements is simpler than with Kaukolu, but at the cost of expressivity: each page in SMW is an RDF resource (same as in Semantic Pad) and RDF statements can refer only to internal pages.

There is a timeline widget (borrowed from the Simile project), which utilizes explicit date-time semantics in wiki content on Ontoworld. See for example http://ontoworld.org/wiki/Upcoming_events. This, of course, makes it very interesting for users to add semantic statements concerning dates and times to any pages they edit on topics which have a time aspect.

On the whole, SMW is, at this stage, the most attractive semantic wiki engine from the perspective of a novice end-user. The reason for this is of course that the other engines are even more prototypes than products.

6.2.3.2 Ability to be used in case study scenarios

Scenario 1: Advanced querying of Semantic Helpdesk: The query interfaces available in SMW are not easy to use, but there is something called "inline queries" which turns out to be very useful. Creating an inline query means that the user edits a wiki page, and in the wiki code of that page enters a query. The result of this query is displayed

whenever the page is requested. The language of these queries seems to be a custom language for SMW, but the beauty of it is that queries can be copied and modified from other pages. That way, a user does not have to learn the query language in order to be able to perform queries similar to other queries, which they see on the wiki. Copying source code to generate similar results was what every web master did when the worldwide web became a global success. Perhaps the semantic web can follow a similar pattern?

The fact that there are a number of advanced semantic queries stored in pages makes it potentially instantly rewarding (and definitely inspiring) for users to add semantic statements.

Scenario 2: Ontology enhanced search in Semantic Helpdesk: There is currently no way that SMW can do the reasoning required to perform scenarios 2 a and b. But it should not be very hard to implement. All the necessary relations can be expressed in SMW.

Scenario 3: Wiki with timeline view to support report writing: SMW is not integrated with any desktop system, so project overview can only be achieved if all project data is in the wiki. With this limitation, though, the scenario can be realized with SMW.

Scenario 4: Annotate resources for sharing: In SMW any user can setup an inline query, which will always display the search results of, for example, pages with a “Shared” property. There is, however, no method for notification, so the user must regularly check the page with the inline query in order to become aware of Claudias recently shared documents.

Scenario 5: Project browsing, In a way specialized views can be achieved by using the inline query feature, and some of its formatting options. However, this is fairly limited in expressivity. For example, there’s no possibility of using templates.

6.2.4. Results summary

	Kaukolu	Semantic Pad	Semantic MediaWiki
Advanced querying of semantic helpdesk	No	No	Yes
Ontology enhanced search in Semantic helpdesk	No	No	No
Timeline to support report writing	No	No	Yes with wiki data
Annotate resources for sharing	Possibly	No	Yes in a limited way
Project browsing	No	Yes	Yes in a limited way

7. Development roadmap

The next step in NEPOMUK is the creation of a new, CDS-based wiki, taking into account what we learned from the prototypes. CDS stands for *Conceptual Data Structures* [23], a top-level ontology for relations. The CDS-based wiki will shift the focus from wiki documents to smaller chunks of information, exploiting the implicit structure of wiki mark-up.

In this section, we first show two wiki usage scenarios, which illustrate how semantic wikis are used in NEPOMUK. Then we analyse functional requirements for semantic wikis in depth, based on the requirements obtained in the course of the Nepomuk case studies.

Some requirements have already been implemented in wiki prototypes, while others are expected to be added by the CDS-based tools, namely a semantic wiki (*SemWiki*) and a graphical, zoom-able concept-map-like tool (*iMapping*).

Finally, we list functional requirements to be realised by other NEPOMUK services, which are vital for the CDS-based tools. To round up our discussion we also list some functional requirements not needed or not likely to be realised in NEPOMUK and explain why.

7.1. Scenarios

From the Nepomuk case studies, a large number of scenarios have emerged. Some of these scenarios are *as-is*, meaning that they describe situations in which we believe semantic desktop technology may provide improvements. Other scenarios are *to-be*, meaning that they describe imaginary future situations in which semantic desktop technology is being used and appreciated.

Some of these scenarios are already possible to realize using features present in today's semantic wiki engines (see section 6.1.). Below we describe two to-be scenarios that are relevant to the semantic wiki technology.

7.1.1. Semantic wiki being used as a personal, networked Customer Relationship Management system

7.1.1.1 Scenario description

Alistair is preparing for a sales pitch at a big telecom company with offices in Birmingham. It is a new client with whom a sales manager booked a meeting. Alistair needs to know more about the company before the meeting and he also needs to prepare a presentation of TMI's track record within the same branch. Alistair first goes to his personal semantic wiki system for information. He instructs his wiki to synchronize all content in the "TMI Sales"-group via the P2P network. Then he starts to browse. When he knows enough about the company he starts to search for information about what TMI has done for telecom companies in the past, he searches for "telecom" and gets a view of the results, which he can view from different angles such as timeline, company size

and location. He chooses to view clients that are approximately the same size and cases that are relatively fresh.

7.1.1.2 Implementation comments

This would be possible if postings in the wiki had semantic links to:

- Relevant industry (Possibly from UN:s ISICv3 ontology. In this case: ”Telecom”.)
- Size of customer organizations
- Geographic location of customer organizations
- Timestamps

Naturally, the timeline part would require a timeline widget in the interface, and a geographic view would do well with a map interface. A faceted browsing interface would also be interesting for this scenario.

7.1.2. Semantic wiki as an interface to desktop data

Scenario description: As a project leader Claudia is responsible for the deliverables in many projects. The deadline is approaching in one of the projects and she needs to get things done. Claudia can see in her task list that the deliverable is only two weeks away. To get an overview of all relevant information to be reported in the deliverable, Claudia looks at the task in the project view. Here she can see what has been done, by whom and easily retrieve all this information to compile the report.

7.1.2.1 Implementation comments:

This would be possible if wiki could access all data on the desktop having been produced by the project, and had semantic links to:

- Timestamps
- Author identities
- Project affiliation

It would also require specialized views (task view, project view). Such views could be similar to Semantic Media Wikis inline queries, or the template views of Semantic Pad.

A simplified version of this scenario (with no desktop integration) was described in section 6.1. .

7.2. Functional Requirements

As a part of the case study analysis, functional requirements from the case studies have been integrated into a manageable number of mid-level requirements. In this section we summarize the functional requirements related to semantic wikis in NEPOMUK. First, we look into functional requirements that are already realised in the prototypes. We plan to integrate most (or all) of them into the CDS-based tools. Further feedback from the case studies will be needed to prioritize the requirements.

7.2.1. Functional requirements realized in NEPOMUK semantic wiki prototypes

- Automatic completion support: Helping users enter tag properties and property values by giving them suggestions for possible word completions, as well as filling out standard properties automatically. The semantic wiki editor should provide user assistance in proposing text completion while the user types some text, in a manner that is similar to what modern code editors propose.
- Identify changes between two revisions of the same document: A product developer or a trainer wants to identify the gap between two revisions of the same document in order to understand the differences and decide which one to use.
- Semantic annotation of phrases inside documents: Assigning meta-data to words, phrases, or document sections, either restricted by a pre-existing domain ontology or open for creation of ad-hoc properties.
- Semantic navigation: Browsing the information based on its semantic structure.
- Semantic search: Searching for information based on both keywords and the semantic structure. (Note: query execution provided by the Nepomuk core middleware).
- Wiki support for ontology design: The wiki should have built-in support for collaborative ontology design.
- RDF/OWL ontology import: The semantic wiki engine should be able to import an existing RDF/OWL ontology and to create the corresponding wiki pages for each class and property of the ontology.
- RDF/OWL export: The semantic wiki engine should be able to export the domain ontologies in RDF or OWL, as well as the entire semantic graph.
- Semantic annotation: Assigning meta-data to an object, either restricted by a pre-existing domain ontology or open for creation of ad-hoc properties.

7.2.2. Functional requirements to be realised in CDS-based tools

First we briefly explain the vision of CDS-based tools, and then list functional requirements, which should be implemented in the business logic of the CDS components.

Vision

A central result of Nepomuk workpackage 1 is a component for knowledge articulation and visualisation. In order to explore visual and textual knowledge interaction metaphors in parallel, we are developing a data model, the Conceptual Data Structures (CDS).

CDS can be seen as the next step after wiki and concept maps, fusing the two concepts and shifting the focus from documents to knowledge models. We plan two interfaces, a visual one (iMapping) and a wiki-like interface (SemWiki, as a joke sometimes called “The SuperWiki”). SemWiki will not be like an ordinary semantic wiki, as it has no notion of “document”. Instead, it puts the focus more on micro-content and its

relations. As a direct result, searches should not return long wiki documents, but short fragments of text with its relations to other parts.

Functional requirements to be realised by CDS-based tools

- Composition of documents using fragments of different documents: Create a new document based on fragments of different documents of previous work.
- Ontology refactoring: The ontology properties and types should be refactorable.
- Template system: The system should let users define advanced rules defining the way resources should be displayed, using templates.
- Visual refactoring: Users with appropriate rights should be able to refactor the contents visually, changing their names, their status, their meta-data.

Summary

Compared to the semantic wiki prototypes, the CDS-based tools aim for

- A tighter integration with desktop data
- Intuitive, efficient graphical user interface (iMapping)
- A switch at the user interface level from RDF to CDS, to reduce cognitive load for knowledge articulation. RDF will be use behind the scenes for implementation purposes only.
- Overcome segmentation of knowledge into pages – instead in CDS knowledge is represented as a highly networked set of small items.

7.2.3. Functional requirements exposed by CDS-based tools

We now list functional requirements that were not realised in semantic wiki prototypes, but are expected to be realised by middleware services. Here we list the services, which should be exposed to the end-user via the user interface of the CDS-based tools.

- Task related requirements such as task creation, delegation, planning, management (status, estimation), and support for providing task patterns, may use a semantic wiki, but much of the backend functionality will be provided by the Task Manager developed in the work package 3.
- Support of diverse access rights with fine granularity: Access rights for the following operations: read, read meta-data only, write, comment, search.
- Access rights visualization: Users can make mistakes in access right settings.
- Audit trail: For each modification on the tag level, it is needed to know who modified it, and when.
- SPARQL queries should be handled by the middleware and the result should be rendered in the wiki editor.
- Notification requirements: Users should be able to express the notification they want to receive using advanced rules.

- Offline reading, change, and synchronization: Modifying information in environments without internet connection.
- Ranking search results based on user rating of resources: The user should be able to give a personal score to the results brought by the system.
- Federated relevance-based search: Taking advantage of content's semantic structure for locating and aggregating the relevant information from various data sources.
- Semantic search in P2P: The ability to search across public information spaces of individuals must be integrated seamlessly.

7.2.4. Functional requirements not to be realised by CDS-based tools

All other requirements have been excluded from the development road map of the CDS-based tools. We list some of them and explain why. Of course, the overall reason is a lack of resources, we had to cut down somewhere.

- Real-time features: Chat among community members, Real-time collaborative editing
  - Real-time communication is not a typical, essential wiki feature.
- General content management requirements: The system should provide following standard ECM features (Enterprise Content Management): content versioning, conformance to content management standards (such as JCR) facilitating import/export of contents, content locking feature.
  - A personal semantic wiki is not an enterprise content management system.
- Group creation, Support for joining or leaving a group/community, News board post and view
  - Access rights and group membership are expected to be part of other NEPOMUK services. The wiki may offer support for using these services in the user interface.
- Mobile device support: Availability of information on mobile devices such as a Palm pilot or a mobile phone (both for reading and modification).
  - A relevant requirement, but this requires a completely different UI design strategy. We don't have resources in Nepomuk do to this, but will allow others to do it.
- Resource sharing: Team members store documents within a document repository.
  - Sharing is part of NEPOMUK, and should be exposed by the wiki, but binary documents are not the core of a wiki.

7.3. Outlook

We show now some early mock-up screenshots, to illustrate the ideas of the CDS-based tools further. Two CDS-based tools are planned: SemWiki and iMapping. As SemWiki is more wiki-like, we show only mockups of SemWiki.

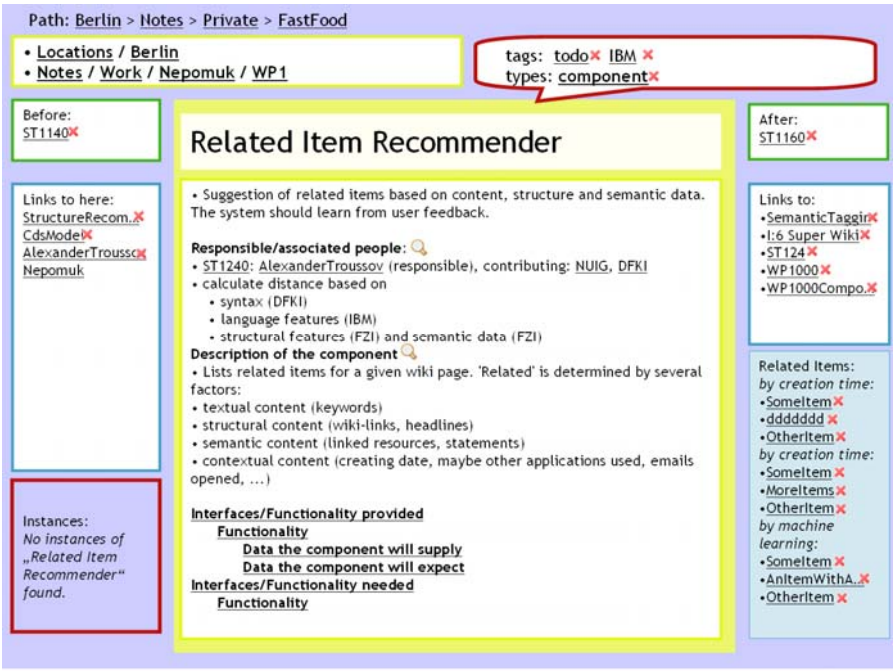

Figure 18. Mock-up Home screen for SemWiki

The middle pane shows an Item (like a wiki page title). Underneath, in the main pane, a virtual wiki page, composed of many CDS-items is shown. To the left and right, in the panes “before” and “after”, we see CDS items which are in the relation cds:hasBefore/cds:hasAfter to the item “Related Item Recommender”. In the same way, “Links to here” shows items with a cds:hasTarget relation to “Related Item Recommender”, and “Links to” shows outgoing links from “Related Item Recommender” to other items. Outgoing links are not necessarily rooted in the page text, instead they are simply loaded from the underlying model.

The pane starting with “Location” and “Notes” shows two different paths using the cds:hasContext relation. “Instances” shows the instances of “Related Item Recommender” – but there are none. “Tags” and “Types” show the annotations of the current item, linked via cds:has Annotation.

The “Path” shows a simple history of last-visited pages. “Related Items” shows items similar to the current one, computed by the “Related item Recommender” component.

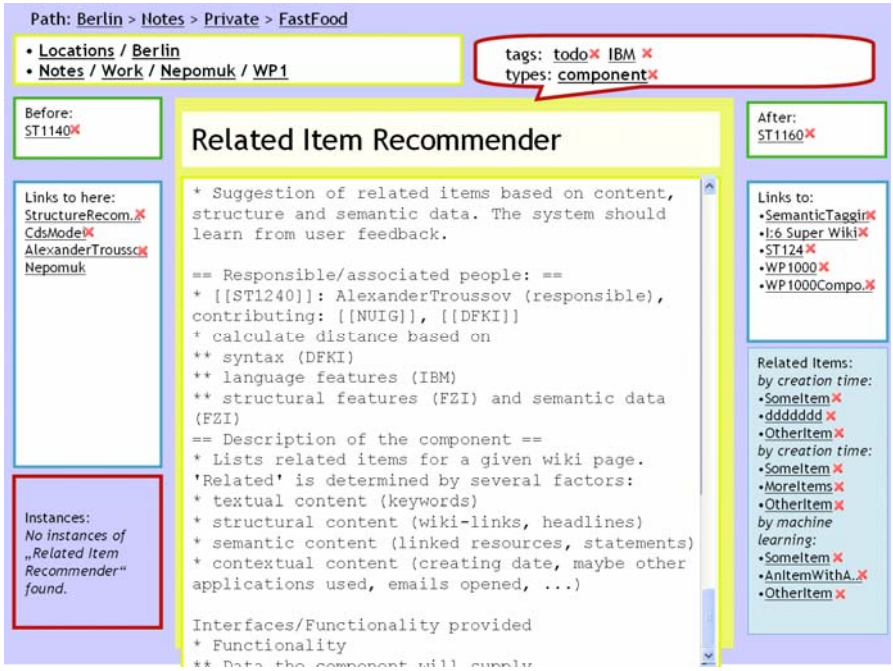

Figure 19. Mock-up wiki edit view for SemWiki. The headlines and bullet list represent the hierarchical nesting of items. In fact, the wiki edit view shows many items at once as a virtual wiki page.

On this mock-up a possible rendering of a hierarchy of CDS items as wiki syntax is shown. Editing the wiki syntax changes the CDS model.

8. Conclusion

Nepomuk is well-positioned to develop a successful Semantic Wiki implementation based on both the lessons learned from the existing prototypes and the analysis of the user requirements from the case studies.

Since Nepomuk is based on the OSGI plug-in platform, it is easy for individual components to have a life of their own. Given wiki popularity, Nepomuk semantic wiki is likely to be one of such components. In other words, the Nepomuk semantic wiki may be disseminated as a part of the Nepomuk's Social Semantic Desktop platform, as a separate application, or as individual wiki components, which can be easily plugged into and integrated with the Nepomuk platform if desired.

Java WIkiModel is a Nepomuk wiki component that has already been integrated into Xwiki, a commercial wiki engine, which is important for the dissimination and exploitation of the Nepomuk platform.

Semantic wikis can be expected to replicate the explosive growth of their non-semantic counterparts. Not only is the wiki market far from being saturated [2], but semantic wikis stand to outperform traditional wikis in many areas (especially in enterprises and communities where there is a significant accumulation of information).

Project10X, a consultancy in semantic technologies, estimates that "Markets for semantic technology products and services will grow 10-fold from 2006 to 2010 to more than $50B worldwide. From 2010 to 2015 the semantic market is expected to grow nearly ten-fold again, fuelling trillion dollar economic expansions worldwide" [5]. The report positions semantic wikis as an integral part of this growth.

9. References

1. http://en.wikipedia.org/wiki/Wikipedia:Size_comparisons
2. E-Mail Is So Five Minutes Ago. BusinessWeek Online, November 28, 2005.
3. Toto
4. Building Blocks of an Enterprise Content Management Business Case for Life Sciences. First Consulting Group (2004).
5. M. Davis. Semantic Wave 2006 Part-1: Executive Guide to Billion Dollar Markets A Project10X Special Report January 2006.
6. http://wiki.ontoworld.org/index.php/Semantic_Wiki_State_Of_The_Art
7. Souzis, A. Rhizome Position Paper, 2004. http://rx4rdf.liminalzone.org/FOAFPaper.
8. Schaffert, S., Gruber, A., and Westenthaler, R. A Semantic Wiki for Collaborative Knowledge Formation. In Semantics (2005).
9. Prud'hommeaux, E., and Seaborne, A. SPARQL query language for RDF. World Wide Web Consortium, Working Draft WD-rdf-sparql-query-20060220, Feb. 2006.
10. Kiesel, M., and Sauermann, L. Towards Semantic Desktop Wikis. UPGRADE special issue on "The Semantic Web" (2005).
11. Malte Kiesel, Kaukolu: Hub of the Semantic Corporate Intranet, SemWiki Workshop, ESWC 2006.
12. Sauermann, L. Gnowsis semantic desktop ISWC 2004 demo. In Proceedings of the International Semantic Web Conference 2004 (2004).
13. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics, 2002.
14. Martin F. Porter. An Algorithm for Suffix Stripping. Program, 14(3):130-137, 1980.
15. Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information.
16. Christopher D. Manning and Hinrich Schütze. Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts, 1999.
17. Hamid Khosravi and Yorick Wilks. Routing email automatically by purpose not topic. Nat. Lang. Eng., 5(3):237{250, 1999.
18. Vitor Carvalho and William Cohen. On the collective classification of email “speech acts”. In Proceedings of SIGIR, 2005.
19. P. Cimiano, G. Ladwig, and S. Staab. Gimme’ the context: Context-driven automatic semantic annotation with c-pankow, 2005.
20. Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In Proceedings of the 14th conference on Computational linguistics, pages 539-545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

21. Brian Davis, Siegfried Handschuh, Hamish Cunningham, and Valentin Tablan. Further use of controlled natural language for semantic annotation of wikis. In SAAW (ISWC), November 2006.

22. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva. User-friendly ontology authoring using a controlled language. In 5th Language Resources and Evaluation Conference, 2006.

23. Max Völkel and Heiko Haller . Conceptual Data Structures (CDS) -- Towards an Ontology for Semi-Formal Articulation of Personal Knowledge. In Proc. of the 14th International Conference on Conceptual Structures 2006. Aalborg University - Denmark, July 2006.

Appendix 1 Links to the software

9.1.1. Semantic Pad

Installations are available from the internal Nepomuk SVN repository (login required):

Directory:
http://svn.nepomuk.semanticdesktop.org/repos/trunk/deliverable/D1-1/D1-1-SemanticPad/

SemanticPad-Web.zip - web-based wiki

SemanticPad-RichClient.zip - rich client wiki

9.1.2. Kakolu wiki

Standalone version (including LanguageWare component):
http://svn.nepomuk.semanticdesktop.org/repos/trunk/deliverable/D1-1/D1-1-Kaukolu/

Embedded version in gnowsis (excluding LanguageWare component):
http://www.gnowsis.org/

Public demo standalone installation (scheduled for January 2007):
http://kaukolu.nepomuk.semanticdesktop.org/

Public open source project website: http://kaukoluwiki.opendfki.de/

9.1.3. Semantic Media Wiki

Hosted demo site:
http://wiki.ontoworld.org/index.php/Semantic_MediaWiki

Web-based wiki installation: http://wiki.ontoworld.org/wiki/Help:Installation