# Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics

edited by Max Völkel

May 15, 2006

# Preface

Dear Reader,

The community of Semantic Wiki researchers has probably first met at the dinner table of the Semantic Desktop Workshop, ISWC 2005 in Galway, Ireland. It was that very night, were the idea of the "First Workshop on Semantic Wikis" and a mailing list were born. Since then, much has happened.

The Topic of Semantic Wikis has evolved from a an obscure side-topic to one of interest for a broad community.

Our mailing list[1] has grown from twenty to over hundred subscribers. As the diversity of papers at this workshop shows, the field of Semantic Wiki research is quite diverse. We see papers on semantic wiki engines, a multitude of ways to combine wiki and semantic web ideas, and application of semantic wikis to bioscience, mathematics, e-learning, and multimedia.

Semantic Wikis are currently explored from two sides: Wikis augmented with Semantic Web technology and Semantic Web applications being wiki-fied. In essence, wikis are portals with an editing component. Semantic Wikis can close the "annotation bottleneck" of the Semantic Web – currently, we have many techniques and tools, but few data to apply them. We will change that.

We wish to thank *all* authors that spend their nights contributing to this topic and thereby made the workshop possible. The high number of good submissions made the work for the programm committee members even more difficult – thank you all for your work. Many thanks also to the ESWC organisation team, which decided in the last minute to grant us a full-day workshop. Let's continue to bring the wiki-spirit to the Semantic Web and enjoy reading the proceedings.

Karlsruhe, May 2006
Max Völkel, Sebastian Schaffert and Stefan Decker

[1]swikig@aifb.uni-karlsruhe.de

# Contents

# Program

**09:00 - 10:30 Session 1**

| | |
|---|---|
| 09:00 - 09:15 | Opening Ceremony<br>*Sebastian Schaffert and Max Völkel* |
| 09:15 - 09:45 | Graphingwiki - a Semantic Wiki extension for visualising and inferring protocol dependency<br>*Juhani Eronen and Juha Rning* |
| 09:45 - 10:15 | Reusing Ontological Background Knowledge in Semantic Wikis<br>*Denny Vrandečić and Markus Krötzsch* |
| 10:15 - 10:30 | Kaukolu: Hub of the Semantic Corporate Intranet<br>*Malte Kiesel* |
| 10:30 - 11:00 | **Coffee Break** |

**11:00 - 12:30 Session 2: Lightning Panels, 5 minutes per paper**

| | |
|---|---|
| 11:00 - 11:15 | **Semantic Wiki Engines**<br>Creating and using Semantic Web information with Makna<br>*Karsten Dello, Elena Paslaru Bontas Simperl, and Robert Tolksdorf*<br>Annotation and Navigation in Semantic Wikis<br>*Eyal Oren, Renaud Delbru, Knud Möller, Max Völkel, and Siegfried Handschuh*<br>SweetWiki: Semantic WEb Enabled Technologies in Wiki<br>*Michel Buffa, Gaël Crova, Fabien Gandon, Claire Lecompte, and Jeremy Passeron*<br>*–discussion–* |
| 11:20 - 11:35 | **Future of Semantic Wikis**<br>iMapping Wikis – Towards a Graphical Environment for Semantic Knowledge Management<br>*Heiko Haller, Felix Kugel, and Max Völkel*<br>The ABCDE Format Enabling Semantic Conference Proceedings<br>*Anita de Waard and Gerard Tel*<br>Learning with Semantic Wikis<br>*Sebastian Schaffert, Diana Bischof, Tobias Bürger, Andreas Gruber, Wolf Hilzensauer, and Sandra Schaffert*<br>*–discussion–* |

11:40 - 11:55 **From Wikipedia to Ontology**
Harvesting Wiki Consensus - Using Wikipedia Entries as Ontology Elements
*Martin Hepp, Daniel Bachlechner, Katharina Siorpaes*
From Wikipedia to Semantic Relationships: a Semi-automated Annotation Approach
*Maria Ruiz-Casado, Enrique Alfonseca, and Pablo Castells*
Extracting Semantics Relationships between Wikipedia Categories
*Sergey Chernov, Tereza Iofciu, Wolfgang Nejdl, and Xuan Zhou*
*–discussion–*

12:00 -12:15 **From Semantics to Wikis**
Wiki and Semantics: Panacea, Contradiction in Terms, Pressure for Innovation?
*Jean Rohmer*
A Wiki as an Extensible RDF Presentation Engine
*Axel Rauschmayer and Walter Christian Kammergruber*
*–discussion–*

12:15 - 12:30 **Wrap up**

12:30 - 14:00 **Lunch**

**14:00 - 15:30 Session 3: Demo/Poster Session**
A Semantic Wiki for Mathematical Knowledge Management
*Christoph Lange and Michael Kohlhase*
Towards a Semantic Wiki-Based Japanese Biodictionary
*Hendry Muljadi, Hideaki Takeda, Shoko Kawamoto, Satoshi Kobayashi, and Asao Fujiyama*
Ylvi - Multimedia-izing the Semantic Wiki
*Niko Popitsch, Bernhard Schandl, Arash Amiri, Stefan Leitich, and Wolfgang Jochum*
Automatic Deployment of Semantic Wikis: a Prototype
*Angelo Di Iorio, Marco Fabbri, Valentina Presutti, and Fabio Vitali*
Bringing the Wiki-Way to the Semantic Web with Rhizome
*Adam Souzis*
Towards a Wiki Interchange Format (WIF)
*Max Völkel and Eyal Oren*
A Collaborative Programming Environment for Web Interoperability
*Adam Cheyer and Joshua Levy*

*. . . and other demos/posters*

15:30 - 16:00 **Coffee Break**

**16:00 - 17:30 Session 4: Interactive**

|  |  |
|---|---|
| 16:00 - 17:00 | Teamwork |
| 17:00 - 17:45 | Teams present their results |
| 17:45 - 18:00 | Conclusion and Outlook |

**20:00 ESWC Demo Session**

**20:45 SemWiki 2006 Social Event**

## Organisation

- Stefan Decker
- Sebastian Schaffert
- Max Völkel

# Graphingwiki - a Semantic Wiki extension for visualising and inferring protocol dependency

Juhani Eronen and Juha Röning

Oulu University Secure Programming Group
Computer Engineering Laboratory, Linnanmaa BOX 4500
FIN-90014 University of Oulu, Finland
ouspg@ee.oulu.fi

**Abstract.** This paper introduces the Graphingwiki extension to MoinMoin Wiki. Graphingwiki enables the deepened analysis of the Wiki data by augmenting it with semantic data in a simple, practical and easy-to-use manner. Visualisation tools are used to clarify the resulting body of knowledge so that only the data essential for an usage scenario is displayed. Logic inference rules can be applied to the data to perform automated reasoning based on the data. Perceiving dependencies among network protocols presents an example use case of the framework. The use case was applied in practice in mapping effects of software vulnerabilities on critical infrastructures.

Keywords: semantic wiki, protocol dependency, visualisation, inference

## 1 Introduction

In recent years, Wikis and the semantic web have become the state of the art methods for the management of information. Wikis have proven to be an effective means for the collective gathering and editing of bodies of data ranging from encyclopaedia to bug tracking and journals. Semantic web is envisioned as a universal medium for data exchange and as a tool to manage the interconnection of information, enabling automated analysis of data. [5] [18] [24] [1]

Both of the technologies have strong selling points: Wikis enable collaborative, open, evolutionary, and easy modification of data, and the semantic web employs Resource Description Framework (RDF), a powerful yet relatively simple language for representing information about World Wide Web (WWW) resources. RDF consists of subject-predicate-object triples that are used to make statements about resources [12]. An RDF resource can basically be anything that has a Uniform Resource Identifier (URI), so it can be used to refer to any web resource. The triples describe either relationships between two resources, the subject and the object, or an aspect of the subject, the value of which is specified by the object. The predicate is a resource that the relationship or aspect describes. Integrating wikis with RDF could bestow it with the editing abilities necessary for efficient knowledge management.

Combining the approaches and techniques of Wikis and semantic web has met little success. The little support traditional Wikis offer for semantic data usually culminates in page categories and different kinds of comment tags. Semantic web tools are often single-user oriented and their operation frequently requires expert skills, which makes knowledge engineering challenging for domain experts. [7] [25] [18]

Wikis have the strength that they focus on the structure of the data instead of its presentation. Wiki users are accustomed to creating, linking and tagging content, which represent the bare minimum requirements for taking advantage of semantics. Adding semantic features to Wikis offers a smooth transition for exploiting different layers of knowledge. [23]

This paper introduces Graphingwiki, a Wiki extension that aims to enable knowledge engineering in Wikis by sidestepping the complexity of semantic technologies. The bare minimum functionality for semantic capabilities in a Wiki includes the implementation of a small but functional subset of RDF. This also follows the Wiki way of doing the simplest thing that could possibly work [6]. Users introduce semantic data into the Wiki by simply tagging pages and page links with words or phrases that sound suitable to them. RDF resources are represented on a Wiki page as tagged links and tagged page data. Together the page tags and the link tags create the RDF statements of the forms `<page> <tag> <linked page>`, `<page> <tag> <URI resource>` and `<page> <tag> <tag value>`.

The tags of represent a flat namespace and do not have a hierarchy of any kind. In a way, this method of adding semantic data resembles folksonomies such as del.icio.us[1]. Tagging is simple and unrestrained as it aims for easy diffusion in the user base. Existing mechanisms, such as different kinds of linking, category pages and macros, are utilised as much as possible. Users may freely select the tags they use, which thus sacrifices consistency for practicality. This approach can prove more useful than forcing any predefined tagging schema [17] [22].

A Wiki functions as its own ontology, formed by all the tags in the Wiki's pages [2]. Each descriptive tag is assigned a page of its own so that terms can be defined and refined in the Wiki itself. The resulting ontologies are expressive to humans but lack the complexity and formality required for elaborate machine-processable constraints on the page data. This does not present a hindrance for knowledge management — in fact, the most successful knowledge models tend to be very simple and specific [18].

Interactive visualisation is proposed as a method for understanding the relations of information on the Wiki pages. Visualisations can be used to navigate the Wiki, and they include facilities for filtering out non-relevant data. This enables the quick derivation of a general view on any desired topic or entity.

Furthermore, Graphingwiki includes some logic reasoning capabilities for refining specific knowledge from the Wiki tags. Wiki pages can include rules that lead to new conclusions about specific tags, and the resulting data can be queried for sets of pages and tags that fulfil the premises of the query. This presents a fine-grained method for discovering relations amongst the wealth of data.

The paper presents the methods in the context of a practical use case, fathoming interdependencies in communication protocols. It is also argued that a similar knowledge management approach would also be effective for other domain-specific tasks where an

[1] http://del.icio.us/

universal topical scope and some of the other stumbling blocks of semantic technologies are not an issue [18] [21].

The paper is structured as follows. In section 2, additions to traditional Wiki features are presented, which beget methods for gathering, visualising and reasoning on the data for the example case. Section 3 presents some results of an initial analysis of the use of Graphingwiki. Directions for future work are laid out in section 4. Finally, the work is summarised in section 5.

## 2 Methodology

The main methods used in Graphingwiki include additions to the MoinMoin[2] Wiki markup and plug-in tools that save the semantic data for later processing, visualise the semantic data and make logical reasoning based on it.

### 2.1 Implementation issues

MoinMoin was selected as the starting point for semantic Wiki development based on the criteria that it is open source, implemented in the Python[3] programming language, is mature and extendable, and uses a file database. Graphingwiki is implemented as a set of plugin *actions* to manipulate the page data, *macros*, and *formatters* to render the semantic data to the desired viewable or processable forms. The design strives to maximise backwards-compatibility and the use of existing MoinMoin features.

The semantic data in each Wiki page is stored into a file of its own, in a symmetrical manner with the page data storage in the MoinMoin Wiki. A general-purpose graph library was created for this purpose. Semantic data is interpreted with the help of existing and augmented Wiki markup, and serialised in the defined graph format. As the markup allows for incoming links links that are not shown on the wiki page itself, a global file database of page linkage was also implemented.

Graphingwiki uses the Python bindings of the Graphviz[4] suite of layout tools to visualise the semantic relations of a Wiki page as graphs. The inference module is a simple unifier-based design in the style of many Prolog implementations.

### 2.2 Wiki markup additions

The chosen markup additions resemble closely those utilised by the semantic Wikipedia -project [24] and semantic Mediawiki [14]. Similar semantic additions developed for MoinMoin [5] were investigated but deemed to include only a portion of the desired features.

The goal of the markup additions is not to implement the whole of the RDF notation, but to present the user a simple and intuitive way to make statements about a Wiki page.

[2] http://www.moinmoin.wikiwikiweb.de
[3] http://www.python.org
[4] http://www.graphviz.org
[5] http://theendmusic.org/programming/MetaDataPlugin

Statements can only describe the containing Wiki page in relation to page tag values, Wiki pages and URI resources. Semantic data is marked up within page content and rendered in a meaningful manner when the page is viewed.

There are two kinds of statements users can make about a Wiki page: MetaData statements and augmented link statements. MetaData statements are used to realise semantic page tags. They are implemented with a macro and therefore follow the MoinMoin macro syntax of the form `[[MacroName(arguments)]]`. The arguments of the MetaData macro consist of tag-value pairs with an optional third argument that omits the macro from page rendering. For example, the statement
`[[MetaData(SpecialPower, x-ray vision)]]`
on a superhero Wiki page denotes that he or she has the extraordinary ability to conduct airport security checks without external hardware, among other things.

Respectively, augmented link statements are used to implement semantic link tags. They extend the MoinMoin named link syntax forms
`[:OtherPage:Wiki page]` and
`[http://example.com URI resource]`
that create links with descriptive labels (see Figure 1). Augmented link syntax adds a link tag to this markup, resulting in links of the forms

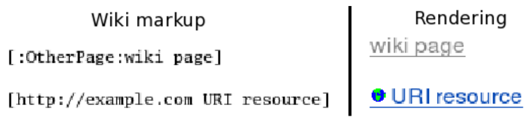


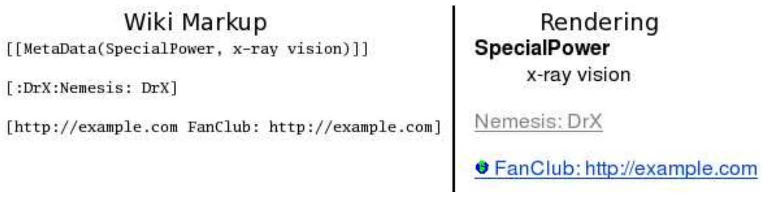
**Fig. 1.** Rendering of normal MoinMoin links.

`[:OtherPage:linktag: page]` and
`[http://example.com linktag: URI resource]`.

The special keyword "From" in the end of the type string denotes that the link is an incoming link, i.e. the referenced page links to the current page instead of the current page linking to it. For example, the statements
`[:OtherPage:linktagFrom: page]` and
`[http://example.com linktagFrom: URI resource]`
indicate that the current page is referenced by the Wiki page or the WWW page, respectively.

The statement `[:DrX:Nemesis: DrX]` on the superhero Wiki page tells that the nemesis of our hero is Dr. X, described in the same Wiki. Respectively,
`[http://example.com FanClub: http://example.com]`
states that the hero's fan club has its web page at the URI http://example.com. Repeating the link in the descriptive string is not required, the examples do so for reasons of clarity only. Figure 2 illustrates the rendering of these statements.

The notation defaults to the namespace designated by the Wiki. To avoid collisions with regular Wiki pages, the pages describing the page tags and the link tags are pre-


Wiki Markup

```
[[MetaData(SpecialPower, x-ray vision)]]

[:DrX:Nemesis: DrX]

[http://example.com FanClub: http://example.com]
```

Rendering

**SpecialPower**
x-ray vision

Nemesis: DrX

FanClub: http://example.com

**Fig. 2.** Rendering of semantic statements.

fixed with 'Property'. Thus, in the examples of the previous paragraphs, 'PropertySpecialPower' and 'PropertyFanClub' are pages in the same Wiki.

By editing the descriptions and semantic data on the Wiki pages describing the page tags and the link tags, the community creates a contract on the formal meaning of a domain - effectively an ontology. This lets the users freely edit the ontology in a very Wiki-like fashion, which reduces the entry barrier and encourages vocabulary growth and expressiveness. For example, users of the superhero Wiki can elaborate on the concept of special powers (i.e. the content of the 'PropertySpecialPower' page), adding further information, declaring exceptions, and so forth. The availability of discussions on the subject, along with relevant links and multimedia, will help in understanding the concept. [11]

Graphingwiki is not planned to support any deeper semantic meaning to ontology entries. RDF schema or datatypes are not supported, nor are pages checked for consistency with any formalism. However, template pages can be used to create implicit meta-ontologies similarly as in Wikitology [8]. For example, a 'SuperheroTemplate' could include statements common for all superheroes, so that when a page for a superhero is created using that template, the author is reminded about the kinds of semantic data that should probably be included.

The semantic markup supports namespaced statements. The list of valid namespaces is gathered from the Wiki's *InterWiki* list. For example, the statement

```
[[MetaData(Wardrobe:JumpSuit, Spandex)]]
```

tells us that the hero in question wears a flashy spandex jump suit, and that the specifics on the style of dress can be found in the Wardrobe Wiki. Respectively, the statement

```
[wiki:WikiTwo/PageTwo OtherWiki:SeeFrom: wiki:WikiTwo/PageTwo]
```

represents the situation where the page 'PageTwo' of the Wiki 'WikiTwo' has a relation with the referencing page defined by the page 'PropertySee' in the Wiki 'OtherWiki'. Naturally, by adding the line

```
dc http://purl.org/dc/elements/1.1/
```

to the *InterWiki* list of the Wiki in question enables the user to employ Dublin Core[6] definitions in the Wiki pages. Although *InterWiki* lists are currently not user-editable in MoinMoin, the *InterWiki* list provides a relatively clean and straightforward way to add new scope to Wiki editing. Graphingwiki uses the namespaces merely as URI prefixes to the resource names, the RDF data corresponding to the resource is not fetched. Still, the namespaced URIs offer some advantages, as users can use standardised seman-

[6] http://dublincore.org/

tic tags with well-defined meanings, some primitive inference rules involving different namespaces can be used, and external RDF tools can utilise the full scope of the external semantic data. The semantic data in the Wiki data can also be dumped from the Wiki in N3 [3] notation for further analysis with external RDF tools.

### 2.3 Visualisation

Visualisations are composed of the node of the current Wiki page, the links leading to the page and from the page, and the nodes depicting the linked pages. Alternatively, all pages belonging to a category of the current page can be used as the root nodes of the graph, instead of merely the current page node. Visualising a category shows a whole field at one glance, including the direct and indirect relations of all the members, along with their immediate surroundings.

Page tags can be used to colour the nodes of the graph, and pages can be filtered based on their tags. Respectively, augmented links are coloured with respect to their link tags, by which they can also be filtered. Filtering can greatly reduce the clutter in the visualisation, and helps in concentrating to desired aspects of the data. Graphs can also be ordered with respect to one of the page tags. The tag values are lexically sorted, determining the rank of the nodes corresponding to the pages. Colouring and ordering the nodes offers two dimensions by which to organise the semantic data.

As an example, Figure 3 depicts a visualisation made by Graphingwiki with data automatically extracted from the WiFiPedia[7] wireless standard resource.

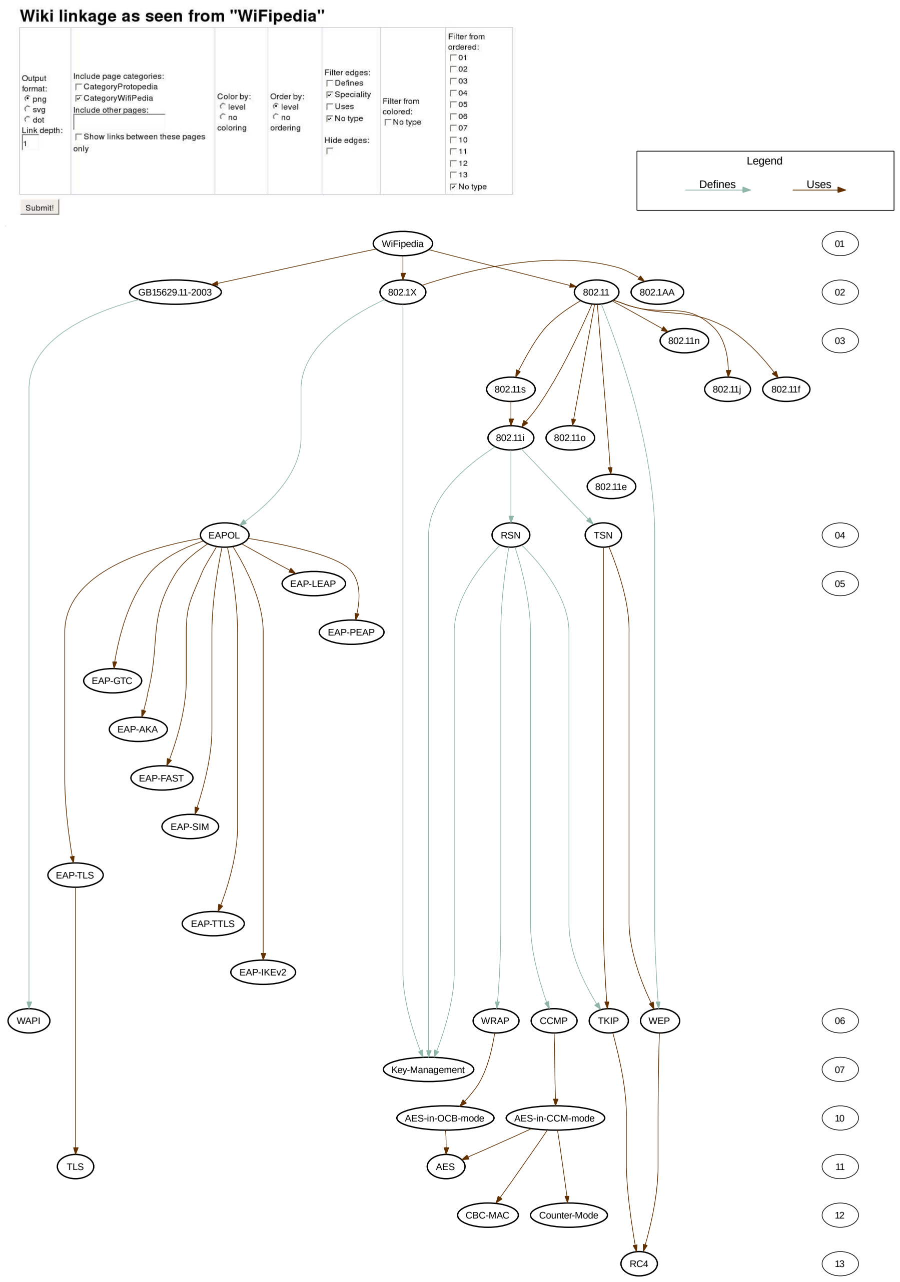
### 2.4 Inference

While visualisation makes semantics comprehensible, inference makes it operational. Generally speaking, inference is used to extend the set of known facts with the help of rules that concern them, and to find the facts, if any, that prove a stated goal. Inference engines that take the first approach are called forward chaining, as they start from valid data, while backwards chaining starts from the goal to be proved, and apply known facts and rules to produce a proof. [16]

A backwards-chaining inference engine is used to answer queries on semantic data. The engine uses Horn clause logic, i.e. clauses that do not have more than one positive literal, also used by many logic programming approaches such as Prolog. Horn clauses have desirable properties in that their satisfiability is solvable in polynomial time with algorithms linear to formula size. As the semantic data can be expressed in the terms of RDF triples, which are basically simple relations, it is straightforward to map them as clauses.

The inference rules and queries are stored as Wiki pages for easy editing and reference. The rules are expressed in the N3 notation, as Graphingwiki markup extensions do not include any way to express them. The result of the query is a set of RDF triples, also in N3 format, that maintain the conditions presented by the query. For example, according to the old adage “the enemy of my enemy is my friend” an evil mastermind, Dr. X, might want to query the superhero Wiki for enemies of his enemies to find new

[7] http://www.wifipedia.org/



Wiki linkage as seen from "WiFipedia"

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| Output format: [✓ png] svg dot<br>Link depth: 1 | Include page categories:<br>☐ CategoryProtopedia<br>☑ CategoryWifiPedia<br>Include other pages:<br>☐ Show links between these pages only | Color by: level / no coloring | Order by: [✓ level] / no ordering | Filter edges:<br>☐ Defines<br>☑ Speciality<br>☐ Uses<br>☑ No type<br>Hide edges: ☐ | Filter from colored:<br>☐ No type | Filter from ordered:<br>☐ 01<br>☐ 02<br>☐ 03<br>☐ 04<br>☐ 05<br>☐ 06<br>☐ 07<br>☐ 10<br>☐ 11<br>☐ 12<br>☐ 13<br>☑ No type |

Submit!

**Fig. 3.** Visualisations of wireless networking standards using data from WiFipedia.

allies to battle his nemesis, Goody Two Shoes. The rule, representing Dr. X's notion on enemies and allies, and the query would be as follows:

```
{?x Enemy ?y. ?z Enemy ?x} => {?z Ally ?y} .
{?who Ally DrX} => [] .
```

The query could result in the following reply:

```
CookieMonster Ally DrX .
DrEvil Ally DrX .
PowderedToastMan Ally DrX .
GoodyTwoShoes Ally DrX .
```

Having disproved the old adage, Dr. X curses his wretched query in frustration.

## 3 Practical usage scenario

Graphingwiki has been used for the purpose of discerning and visualising interdependencies of protocols. Data is gathered from technical specifications and from experts of different protocol environments. The accumulated data is then visualised, bringing up different aspects from the data related to protocol dependency and security. The resulting views can additionally be used as a communication method between researchers and other actors. Inference is used as an method of gaining deeper insight to dependency chains and networks.

### 3.1 Protocol dependency

Protocols can be thought of as languages shared by the information systems for communication. Most current information systems implement a large number of protocols, most of which it requires for normal functionality. In effect, the system can be communicated with by a number of means, and it parses diverse network data. This makes the system, as the other systems on the network, dependent on the implemented protocols in a multitude of ways. Assessing the dependency of protocols and the predominance of the protocols in the network is critical in the view of its robustness.

The issue is further complicated by the fact that protocols within a single protocol family or even between protocol families often have connections. Thus, the impact area of vulnerabilities in a shared component is greatly expanded due to protocol dependency. This may lead to faults that can have a significant effect on an infrastructure. [9]

### 3.2 Extraction and augmentation of data

Initially, protocol data is gathered from standardisation organisations and from indices collecting data on standards. Examples of semantic data in standards include status, types of relations with other standards, the protocols involved and so forth. The data is gathered with scripted methods and inserted into corresponding Wiki pages with similar means. Most of the structured data in the standard texts is imported, following the approach of aggressive population of semantic and ontological data from existing databases [4]. This results in the quick generation of a relatively rich body of data as a starting point for a comprehensive protocol Wiki. Also other semistructured data on standards can be inserted.

While the process of adding given semistructured data cannot be effectively automated for all cases, the extraction approach is a pragmatic one, making the best use of the data available. Although the different data sources may adhere to any number of conflicting explicit or implicit ontologies, a lightweight approach to ontology gives the leverage to process the resulting primordial soup. This represents a bootstrapping process for semantic Wikis, as the benefits of semantic data are illustrated only by the availability of such data. These benefits far outweigh the costs of generating the semantic data along with the data. Similar approaches to data extraction have been applied successfully [20] [19].

After the initial data gathering phase, the data is inserted into Graphingwiki. The details of this process are somewhat subject-dependent, but follow the same basic principles. Whenever new concepts are introduced in the data, new Wiki pages are created to describe them, and data concerning a protocol or other concept already in the Wiki is simply updated to that page.

As much of this data as possible is inserted to the pages in the forms of the attributes of the concept and its relations to other concepts, as these forms of data are machine-processable. Page templates can be used to help formalise the extended markup [24]. On the other hand, custom semantic tags for specific situations or scenario can be used. Explanations, quotes, and WWW resources can be written on the page as is.

In the collaboration phase, the experts are invited to join in to view and augment the results gathered in the Wiki from their interviews and additional sources. Experience has indicated that it may help in this phase if the data gathering phase has not been exceedingly careful in filtering contradictory or controversial arguments about the protocols. This is due to the fact that experts are often more keen to remove such flaws from existing data than to add complementary data to an empty page.

During these phases the data body is developed from a fairly generic and dry viewpoint towards exceedingly rich and specific use cases. Users immediately benefit from the practical domain experience included in the Wiki.

### 3.3 Visualisation and Reasoning

The ability to make logic deductions on the expert-supplied data can unearth results not easily discovered by traditional means. As an example from the Wiki context, Decker et al. uses reasoning to enable reuse of software engineering knowledge [8]. The approach taken in the development of Graphingwiki with respect to reasoning techniques is straightforward and pragmatic, so that the inclusion of logic is based on approaches that are known to work and are required. The focus lies heavily on immediate benefits of reasoning, the inclusion of higher-order structures is deferred until they are explicitly needed [4].

As an example case of inference on the domain of protocol dependency, the true cause of a network error related to two hosts containing a plenitude of services can be inferred from a data body on protocols and related implementations. Similarly, the gross effect of a single vulnerability for a network can be assessed, optionally involving even chains of vulnerabilities and exploits. Similar approaches have emerged in the context of security research, particularly in network vulnerability assessment (e.g. [15]), but also in inspecting the configurations of single workstations (e.g. [10]).

### 3.4 Limitations

The population of a Wiki with data from semistructured sources is a useful facility, but it may not be applicable to a portion of available material due to technical or licensing issues. In some cases, the data abstraction features may suffer from some constraints. Visualisation techniques are naturally limited to a certain volume of data that they can relay in an efficient manner.

Reasoning also has its limitations that have hindered its use in many cases. Main problem is the state space explosion resulting from massive knowledge bases. This can be countered by using monotonic logic and highly domain-specific data sets, although limits on query tree depth and traversal time can also be of help. All the statements made with Graphingwiki are essentially monotonic, as they only bring more data to the knowledge base without contradicting earlier statements. This is due to the inherent lack of meaning of the statements in the Wiki, as the different aspects and relations are only given meaning by humans interpreting them, or by the inference rules and queries.

While the statements are limited in their effect, there are no similar restrictions to the inference rules queries. Thus, great care must be taken when generating them, as they might bring contradiction or belief revision into the system. The heterogeneity of the data gathered from various sources can present limitations to reasoning. As there are no guarantees on given semantic data being present on all concerned pages, the inference rules may not match all relevant data [19].

## 4 Discussion

Semantic Wikis are a natural placeholder for various kinds of domain-specific data that are produced in normal course of work, enabling collaboration and groupwork. The gathering and visualisation of information was found straightforward with the methods explained in this paper. Visualising the relations of protocols has proved to be an effective method for realising the scope of a protocol in application and network contexts. The visualisations have been used in various stages of protocol-related vulnerability work.

It has been claimed that semantic tools also have applications in learning by evaluating, manipulating, and presenting data in new ways [20]. Visualising this data according to the requirements of a given domain presents an effective method for making its contents easier to grasp by humans. Consequently it is no surprise that in addition to fulfilling its intended purpose for creating protocol visualisations, Graphingwiki has proved to be useful for a variety of other tasks. New application areas emerged at a constant rate during its development, indicating that there is a great need for lightweight information visualisation facilities. Some of the these areas are illustrated by the examples in the following paragraphs.

Figure 4 is a organisational chart of a company that has been created with Graphingwiki. The nodes of the graph represent the roles of different employees while edges report the reporting and management chains between the roles. The roles are ordered by their required experience and colored according to the departments they belong to. Similarly, Wiki pages containing data on employee responsibilities and fields of know-how could enable efficient resource management and aid in problem resolution. Social network mapping techniques could be used on this data even further, for example to identify communities and communication bottlenecks.

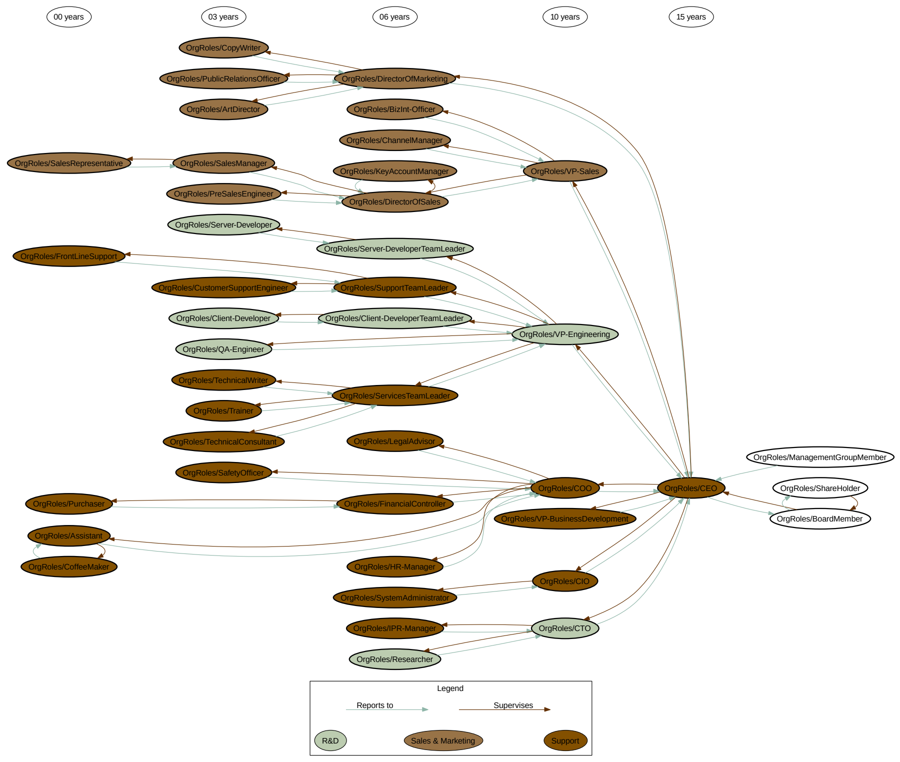


**Fig. 4.** An organisational chart created with Graphingwiki.

Figure 5 represents a survey on the research on laser technologies and on the manufacturers of laser products. Data on different actors of the field was inserted to a Wiki, along with their relations. This view on the Wiki data depicts the Finnish laser product vendors by location, with links to the application areas of their products.

Graphingwiki could be enhanced in a variety of ways to increase its efficiency and expressiveness, and to make it more approachable for users. A full support for different levels of ontology formalisation would be an obvious benefit, along with mechanisms

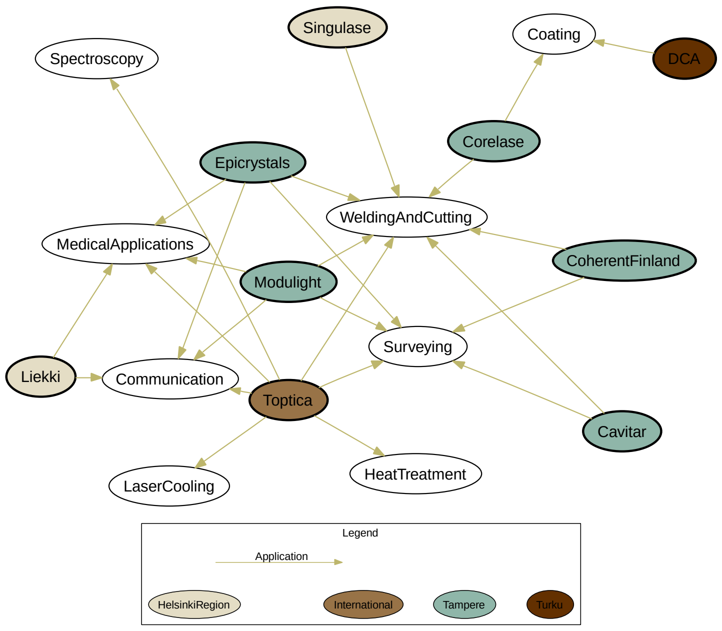


**Fig. 5.** Different laser applications, their producers, and locations.

that check the page's adherence to a specified ontology [18][8]. RDF schema to manipulate typed data could be added, as well as some OWL features. Many of the implicit Wiki relations, such as being part of a certain category or being made with a specific template, could be formed explicitly with these facilities. Importing RDF data related to instances of other namespaces would also increase the application scope of Graphingwiki.

The creation of ontologies might be easier and more scalable if users could first use the augmented link syntax to denote all statements, shifting to use the MetaData-macro only when it has been ascertained that the values of the link tags do not have further structure and can be considered to be mere tag value data.

Some of the semantic data in a Wiki could also be automatically generated from the knowledge of who created and modified the page, creation date, data on referring page given by the browser, and so on [8]. Similarly, page categories could be automatically suggested to the user by comparing the page with representatives from existing categories using Bayesian classification.

The inference engine in the extension will yet require some work to be fully operational in a practical manner. A major part of this work includes creating the basic queries representing the common use cases of the inference engine. It also includes queries with additional functionality such as "find all of the links from the Wiki that point to non-existing pages". Visualisating the results of the queries would increase their understandability in the case of complex queries.

Users could be greatly aided by the creation of semantic data macros specific to their domains of knowledge. Further, the user interface could include tag word suggestions to help converge the tagging scheme, similarly as in the del.icio.us service and the Makna semantic Wiki. Another aid for the tagging scheme would be the use of synonym-declaring relations. However, experiences from Wikipedia suggest that problems regarding the selection of tags are not critical, and that the situation is further ameliorated by the Wiki pages describing the tags [24].

The visualisation style and the GUI would benefit from user interaction studies and research on other visualisation styles. Different dimensional views such as Zzstructures and Polyarchies could be used to produce more data-compact views [13]. Wiki pages could include navigation section of related links created with the help of faceted classification [23] [1], providing another alternative to the traditional wiki category scheme.

Many common use cases of Wikis, such as systems documentation and contracts, can encompass a smorgasbord of pages while placing great demands for the trustworthiness of the included data. As Wiki pages are by nature under constant revision and refinement, these use cases require facilities for specifying the page versions that constitute the de facto state of the entity. Visualisations that are bound to specific page revisions could be used to facilitate the version control of such entities while making their structure easier to apprehend.

Encapsulating the revision state in visualisations is a similar concept as the transition of software version control from the per file Revision Control System (RCS) into the set oriented Concurrent Versions System (CVS). Whereas in software development the module hierarchy facilitates easy revision tagging, in non-hierarchical Wikis the bound visualisations can provide for one click capture of a snapshot of a larger concept. Following the evolution of these visualisations could give insight into the development of the entity, and the processes involved.

## 5 Conclusions

This paper has shown how the MoinMoin Wiki can be extended to include some semantic capabilities. Graphingwiki uses the MoinMoin plugin mechanism along with its existing capabilities of linking and category pages to create a simple and lightweight semantic tagging scheme. The tagging scheme was further used to provide for the visualisation of semantic data and making reasoning upon it.

Graphingwiki has been used for knowledge engineering in the domain of network protocols. The visualisations have proven to be an effective aid in discovering dependencies between protocols, while the reasoning capabilities showed promise for uncovering complex relationships in the semantic data. The visualisations have been used in various stages of protocol-related vulnerability work.

Future research on Graphingwiki include analyses on the visualisation style and user interaction methods in the tool. This research could result in more compact and easily manageable views. Another future direction is the inclusion of more sopisticated semantic features, the lack of which currently limits the use of Graphingwiki with other semantic tools and data sources.

A great demand was noted for the management and visualisation of data from diverse domains. Usage of the tool was then attempted in a number of application areas. Initial experiences on the applicability of Graphinwiki for purposes outside its intended domain of application were very encouraging.

Therefore, a similar approach to handling, visualising, and inferring on data would probably be of much use in many other domains, including enterprise resource management and social network mapping. Organisational human resources related skill and social network mapping and documenting information systems from deployment level to strategy view with dimensions on security policy and system interdependencies are examples of envisioned use cases.

## References

1. Aumueller, D. SHAWN: Structure Helps a Wiki Navigate. In Proceedings of the BTW-Workshop "WebDB Meets IR", Karlsruhe, Germany, March 1, 2005. URL: `http://dbs.uni-leipzig.de/~david/2005/aumueller05shawn.pdf`
2. Aumüller, D., Auer, S. Towards a Semantic Wiki Experience - Desktop Integration and Interactivity in WikSAR. In proceedings of the 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure, Galway, Ireland, November 6, 2005. URL: `http://www.semanticdesktop.org/SemanticDesktopWS2005/final/22_aumueller_semanticwikiexperience_final.pdf`
3. Berners-Lee, T. An RDF language for the Semantic Web - Notation 3 (1998). URL: `http://www.w3.org/DesignIssues/Notation3.html`
4. Berners-Lee, T. WWW2004 Keynote. Keynote speech in the 13th World Wide Web Conference, New York City, US, May 17-22, 2004. Slides available at: `http://www.w3.org/2004/Talks/0519-tbl-keynote/`
5. Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American 284, May 2001. URL: `http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2`
6. Cunningham, W. et. al. Do The Simplest Thing That Could Possibly Work. URL: `http://c2.com/cgi/wiki?DoTheSimplestThingThatCouldPossiblyWork`
7. Cunningham, W. et. al. Wiki Design Principles. URL: http://c2.com/cgi/wiki?WikiDesignPrinciples
8. Decker, B. et. al. Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis. Workshop on Semantic Web Enabled Software Engineering (SWESE), at the 4th International Semantic Web Conference (ISWC 2005), Galway, Ireland, November 6, 2005. URL: `http://www.mel.nist.gov/msid/conferences/SWESE/repository/11self-org_reuse_of_se.pdf`
9. Eronen, J., Laakso, M. A Case for Protocol Dependency. In proceedings of the First IEEE International Workshop on Critical Infrastructure Protection, Darmstadt, Germany, November 3-4, 2005. URL: `http://www.ee.oulu.fi/research/ouspg/protos/sota/matine/IWCIP2005-dependency/index.html`
10. Govindavajhala, S., Appel, A.W. Windows access control demystified (2006). URL: `http://www.cs.princeton.edu/~sudhakar/papers/winval.pdf`
11. Hepp, M., Bachlechner, D., Siorpaes, K. OntoWiki: Community-driven Ontology Engineering and Ontology Usage based on Wikis. In proceedings of the 2005 International Symposium on Wikis, San Diego, US, Oct 16-18, 2005. URL: `http://www.heppnetz.de/files/ontowikiDemo-short-camera-ready.pdf`

12. Manola, F., Miller, E. Resource Description Framework (RDF) primer. W3C Recommendation (2004). URL: http://www.w3.org/TR/rdf-primer/
13. McGuffin, M.J., Schraefel, m.c. A Comparison of Hyperstructures: Zzstructures, mSpaces, and Polyarchies. In Proceedings of ACM Conference on Hypertext and Hypermedia, Santa Cruz, USA, August 9-13, 2004. URL: http://eprints.ecs.soton.ac.uk/9230/
14. Muljadi, H. et. al. Semantic MediaWiki: a user-oriented system for integrated content and metadata management system. In Proceedings of the IADIS International Conference WWW/Internet 2005, Lisbon, Spain, Oct 19-22, 2005. URL: http://www-kasm.nii.ac.jp/papers/takeda/05/hendry05icwi.pdf
15. Noel, S. et. el. Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs. In proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03), Las Vegas, US, December 8-12, 2003. URL: http://www.acsa-admin.org/2003/papers/98.pdf
16. Norvig, P. Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp. Morgan Kaufmann Publishers (1992)
17. Rocha, L.M., Bollen, J. Biologically motivated distributed designs for adaptive knowledge management. In: Segel, L., Cohen, I. (eds) Design Principles for the Immune System and other Distributed Autonomous Systems. Oxford University Press, Oxford, UK (2001) 305-334
18. Schaffert, S., Gruber, A., Westenthaler, R. A Semantic WIKI for Collaborative Knowledge Formation. In proceedings of SEMANTICS 2005 Conference, Vienna, Austria, November 23-25, 2005. URL: http://www.salzburgresearch.at/research/gfx/SemWikiForCollKnowForm_20060120.pdf
19. Schraefel, M.C. et. al. The evolving mSpace platform: leveraging the Semantic Web on the Trail of the Memex. In Proceedings of the 16th ACM Conference on Hypertext and Hypermedia, Salzburg, Austria, September 6-9, 2005. URL: http://eprints.ecs.soton.ac.uk/10710/
20. Shadbolt, N. et. al. CS AKTive Space or how we learned to stop worrying and love the Semantic Web (2004). URL: http://eprints.ecs.soton.ac.uk/8817/
21. Shirky, C. The Semantic Web, Syllogism, and Worldview. Clay Shirky's Writings About the Internet (2003). URL: http://www.shirky.com/writings/semantic_syllogism.html
22. Shirky, C. Ontology is Overrated: Categories, Links, and Tags . Clay Shirky's Writings About the Internet (2005). URL: http://www.shirky.com/writings/ontology_overrated.html
23. Völkel, M. Personal Knowledge Management with Semantic Wikis (2005). URL: http://www.xam.de/2005/12_voelkel_oren_SPKM_submission_eswc2006.pdf
24. Völkel, M., Krötzsch, M., Vrandecicm, D., Haller, H. Semantic Wikipedia. In proceedings of the 15th International World Wide Web Conference, Edinburgh, UK, May 22-26, 2006. URL: http://www.aifb.uni-karlsruhe.de/WBS/hha/papers/SemanticWikipedia.pdf
25. Völkel, M. et. al. Semantic Wiki State of The Art Paper. Under development. URL: http://wiki.ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art

# Reusing Ontological Background Knowledge in Semantic Wikis

Denny Vrandečić and Markus Krötzsch

{vrandecic,kroetzsch}@aifb.uni-karlsruhe.de
AIFB, Universität Karlsruhe, Germany

**Abstract.** A number of approaches have been developed for combining wikis with semantic technologies. Many semantic wikis focus on enabling users to specify properties and relationships of individual elements. Complex schema information is typically not edited by the wiki user. Nevertheless, semantic wikis could benefit from taking existing schema information into account, and to allow users to specify additional information based on this schema.
In this paper, we introduce an extension of Semantic MediaWiki that incorporates schema information from existing OWL ontologies. Based on the imported ontology, the system offers automatic classification of articles and aims at supporting the user in editing the wiki knowledge base in a logically consistent manner. We present our prototype implementation which uses the *KAON2* ontology management system to integrate reasoning services into our wiki.

## 1 Introduction

Wikis allow for the simple and quick editing of web content. They considerably lower the barrier for contributing to a website, and especially allow to fix small glitches and errors quickly. This has lead to the deployment of wikis both for web communities – such as the programming languages pattern group where the idea of wikis originated [10] – and as an extension to existing intranet systems in corporate settings.

As of today, a number of approaches have been developed in order to combine wikis with semantic technologies, and actual implementations of semantic wikis are becoming more and more mature. While the actual goals and methods of existing systems vary greatly, many semantic wikis focus on enabling users to specify properties and relationships of individual elements or articles. Complex schema information, as considered in expressive semantic web languages such as OWL [23], is typically not considered for being edited in the wiki.

Instead of extending semantic wikis into general purpose ontology editors – which is a task where dedicated ontology editors [19, 25, 14] are usually more suitable for – we investigate how semantic wikis can benefit from existing, possibly expressive ontologies. Especially in corporate settings, a fixed schema for metadata is often already in use and part of the internal workflow of the company. Changes to the schema typically are a well managed task. Still, wikis can be a convenient tool for managing or annotating instance data for complex ontologies. Schemas often are quite stable, whereas instance data changes at a much higher pace. In practice, this means that semantic wikis should be able to take existing schema information into account, and allow users to specify additional information based on this schema.

In this paper, we present an extension of Semantic MediaWiki [15, 27] that incorporates schema information from existing OWL ontologies. Based on an imported ontology, the user interface facilitates the reuse of the ontological concepts to categorize articles and the properties to type links. It offers an automatic classification of articles and aims at assisting the user to edit the wiki knowledge base in a logically consistent manner. In our prototype implementation, we employ the KAON2 [12] system to integrate necessary reasoning services into our wiki.

The following section describes different use cases and scenarios, where the presented system would be of advantage. This is followed by the details of our mapping to OWL (Sect. 3) and concrete uses of ontological data (Sect. 4). We then describe how the implementation is working and how our system is used (Sect. 5). Before we conclude our work in the last section, we offer an overview over related work (Sect. 6).

## 2 Use Cases and Requirements

We start our investigations by discussing typical use cases for the reuse of ontological knowledge in semantic wikis. This motivates our approach and allows us to derive basic requirements for its technical realization.

Since one obvious requirement for any practical system is that it succeeds in performing its computations, we explicitly exclude global scale wikis such as Wikipedia[1] from our target environments. In the context of expressive ontology languages, one is quickly confronted with computational problems that are not tractable, so that semantic technologies typically do not scale indefinitely. However, wikis are also employed very successfully in contexts where the overall amount of data is much smaller.

Since we started to develop Semantic MediaWiki, we have been approached several times by companies that are interested in using semantic wikis in an enterprise setting. Wikis indeed are successfully applied to cooperatively manage knowledge in working groups or project teams. Concurrent access and ease of use are important advantages over other content management systems in this setting. The added value of semantic wikis in this scenario is the ability to leverage the wiki's contents in other enterprise applications.

For a concrete scenario, consider a wiki used for coordinating a particular project team within a company. Using semantic technologies, relevant parts of the wiki data shall automatically be gathered by the company's intranet search engine. In the wiki, project members coordinate their activities, and describe their progress on their deliverables. This data can then be collected from the wiki and reused in other applications, e.g. to create monthly report figures, or even up-to-date status reports that are generated on request. As the semantic wiki reuses the company's metadata schema for documents and respects the associated constraints (e.g. no document must have more than one title and topics must stem from a predefined set of topics), the automatic integration into the corporate information infrastructure works smoothly.

[1] `http://www.wikipedia.org`

Another frequent use case is to use existing ontologies to bootstrap the contents and vocabulary of a semantic wiki. For a concrete example, assume that an international conference wants to use a wiki for gathering information around the event. Participants can use the system to exchange information about accommodation and travel, to coordinate Birds-of-a-feather (BOF) sessions, or to actively provide links to their presentation material. At the same time, the organizers publish official schedules on protected wiki pages. Using a semantic wiki, this data can be queried and extended in complex ways, e.g. to provide a scheduling system that suggests sessions and BOF sessions based on a participants interests. Also, if the conference management system supports some form of RDF export, one can initialize the wiki pages with basic information about accepted papers and participants. The ESWC2006 wiki[2] is based on such a bootstrapped system.

In a third example, we consider the usage of semantic wikis in personal knowledge management [20]. There, the wiki is operated as a desktop application and cooperative editing is not required. Semantic technologies simplify data organization and search, and the machine-processable annotations provide suitable interfaces with other semantic desktop applications. For instance, the wiki can be used to take notes about persons, and one would like to combine this information with address book applications. Using vocabulary from existing ontologies, the wiki becomes compatible with various types of metadata, and thus its information could be used in RDF based desktop tools. Another example would be to import Friend-of-a-friend [8] files directly from the web.

All of the above use cases stem from our practical experience, and we have been asked by companies and organizations to support them. The following requirements are emerging from these use cases:

– referring to existing ontological vocabularies from the wiki,
– incorporating schema information and constraints from external ontologies,
– exporting data from the wiki in a standard ontology language,
– importing data from external ontologies, such that it is represented and edited through the wiki.

Note the difference between fully importing external data into the wiki and merely incorporating external information for editing tasks. The former implies that the imported knowledge is represented in the wiki afterwards and can be edited by users, whereas the latter means that the wiki is aware of additional external data or constraints that must be taken into account, but that cannot be modified within the wiki. This is further described in Sect. 4.1.

We argue that the full import and representation of all kinds of complex schema information into the wiki is not an immediate requirement, and is often not even desirable at all. Firstly, from a user perspective, it is quite complicated to edit such schema information within a wiki, and dedicated graphical user interfaces of full-fledged ontology editors might be much better suited for this task [19, 25, 14]. Secondly, in many of the above use cases the ontological schema should not be changed by users of the wiki at all. In the opposite, ontologies are often considered as vehicles to achieve interoperability with other applications, and this requires that all participants adhere to the original

[2] `http://wiki.eswc2006.org`

schema. The evolution of the schema is usually done by a central board, as described, for example, by the DILIGENT methodology [28]. Thirdly, distributed ontology editing in itself is not a trivial topic. In contrast to the situation in software engineering, it is neither easy nor usual (although desirable) to separate ontologies into independent modules of networked ontologies. Furthermore, small changes in some parts of an ontology can have strong effects on the overall semantics.

## 3 Semantic MediaWiki in Terms of OWL DL

Semantic MediaWiki is an extension to the MediaWiki system that allows users to add various types of ontological information to the wiki, and which forms the basis of the implementation that we describe in Sect. 5. In this section, we relate the formal information gathered within this wiki system to the Web Ontology Language OWL. In particular, we discuss export and import of OWL DL ontologies.

### 3.1 Extracting Ontological Knowledge from the Wiki

We first describe the expressive means that are available in the wiki, and specify their semantics in terms of the OWL DL part of the Web Ontology Language. This defines the formal semantics of the annotations that are used in the wiki such that a canonical OWL DL export becomes possible. We remark that the user interface of Semantic MediaWiki does not strictly require the formal interpretation in terms of OWL DL, or the restriction to the expressive means of this language. Since most complex features of OWL are not used in Semantic MediaWiki, one could even argue that the wiki's annotation mechanism might as well be used to author different ontology languages. However, we make use of some characteristic OWL features such as equality reasoning and transitive roles, and we feel that OWL DL's set-based semantics for classes and roles is more intuitive than the semantics of RDFS.

Also recall that OWL DL is conceptually related to description logics. In particular, one can divide ontological elements into *instances* that represent individual elements of the described domain, *classes* that represent sets of individuals, and *roles* which represent binary relations between individuals. The way in which Semantic MediaWiki represents knowledge was partially inspired by OWL DL and one can naturally relate the elements of the wiki, i.e. the individual content pages, to the basic vocabulary of OWL. Technically, the MediaWiki system employs *namespaces* to distinguish several types of content pages, and our semantic interpretation exploits this mechanism of "typing" pages as follows:

*OWL individuals* are represented by normal article pages. These pages typically constitute the majority of the wiki's contents, mostly contained in the MediaWiki's `Main` namespace. However, there are several additional namespaces, such as `Image` or `User`, which also are interpreted as individuals.

*OWL classes* in turn have natural counterparts in the wiki in form of MediaWiki *categories*. The category system, which was introduced only in 2004 [29], quickly became the most important feature for classifying articles in Wikipedia. Categories are represented as pages within the `Category` namespace. They can be organized in a hierarchical way, but it is not possible to make a category contain other categories. Thus Wikipedia's category system is more similar to the semantics of classes in OWL DL than to the semantics of classes in RDFS.[3]

*OWL properties,* i.e. roles in description logic, do not have a counterpart in MediaWiki, and were introduced by the Semantic MediaWiki extension. OWL further distinguishes object-properties (describing relationships between two individuals) from data-properties (associating individuals with values of a given datatype), and a similar distinction is found in Semantic MediaWiki. Object-properties are represented by pages in the namespace `Relation`, whereas data-properties are represented by pages in the namespace `Attribute`.

In addition to the above correspondences, the usage of some namespaces in MediaWiki suggests to ignore them completely for semantic interpretation. Most prominently, this includes all `Talk` pages since they do not represent separate concepts, but are merely used to collect notes and discussions on other articles. Semantic MediaWiki can be configured to ignore annotations given on such pages according to intended usage.

Based on the above mapping to OWL, Semantic MediaWiki allows users to describe various ontological statements within the wiki. An incomplete overview of OWL constructs that can be represented in the wiki is given in Table 1. OWL statements about some OWL individual/class/role are specified in Semantic MediaWiki by providing annotations on the wiki page that corresponds to this element. For example, in order to state that the object-property *is located in* holds between *SemWiki2006* and *Budva*, one writes `[[is located in::Budva]]` within the article about SemWiki2006. Further details on annotation in Semantic MediaWiki and the underlying principles are found in [27].

Semantic MediaWiki includes an export function that generates OWL/RDF documents according to mappings such as the ones in Table 1. The export function also associates URIs with all wiki pages. These URIs bijectively correspond to concrete pages. However, they are not *identical* to the article URLs in order to prevent confusion between ontological concepts (e.g. the city of Budva) and actual HTML documents (e.g. the article about Budva).

## 3.2 Adding External Ontologies

Table 1 indicates that, besides some rather simple schema information, the wiki is mainly used to provide concrete descriptions of individuals and their relationship. In description logics, this assertional part of a knowledge base is known as the *ABox*, and

[3] Note that the informal semantics of categories as used in Wikipedia varies. E.g. *Montenegro* does not describe the class of all "Montenegros," but the class of all article topics related to Montenegro.

**Table 1.** Representation of OWL constructs in Semantic MediaWiki.

| **OWL** | **Semantic MediaWiki** |
|---|---|
| OWL individual | normal article page |
| `owl:Class` | article in namespace `Category` |
| `owl:ObjectProperty` | article in namespace `Relation` |
| `owl:DatatypeProperty` | article in namespace `Attribute` |
| **Statement about element *page*** | **Syntax in wiki-source of *page*** |
| object-property | `[[property_name::object_article]]` |
| attribute-property | `[[property_name:=value_string]]` |
| `rdf:type` *class_name* | `[[Category:`*class_name*`]]` (on article pages) |
| `rdfs:subClassOf` *class_name* | `[[Category:`*class_name*`]]` (on category pages) |

in our case we even restrict to ABox statements without any complex concept terms. Moreover, the annotations in Semantic MediaWiki are restricted in such a way that the exported OWL/RDF cannot be logically inconsistent. In the following, we discuss how the ontologically simple knowledge base of the wiki can be combined with complex schema information from an external ontology.

We wish to combine the OWL representation of the wiki contents with an external OWL ontology. Since merging of OWL DL specifications is trivially achieved by taking the union of their statements, the only problem in doing so is the mapping between elements of the two ontologies. It was already mentioned that URIs for all elements of the wiki are generated automatically. These URIs are very similar to the page URLs of the wiki, and do generally not agree with the URIs used in the external ontology. However, OWL provides us with expressive means to describe that two different URIs represent the same entity (with respect to its extensional interpretation). Depending on the type of entity that one considers, this is achieved with `owl:sameAs`, `owl:equivalentClass`, and `owl:equivalentProperty`. We incorporate this specification into the wiki via a new attribute `equivalent URI` as shown in Table 2. To give an example, on the page `Category:Person` one could add the statement `[[equivalent URI:=http://xmlns.com/foaf/0.1/Person]]`, thus stating that every page tagged with this category describes a person in the sense of the FOAF vocabulary.

**Table 2.** Mapping of concepts to external URIs in Semantic MediaWiki.

| **OWL** | **Semantic MediaWiki** |
|---|---|
| `owl:sameAs` *URI* | `[[equivalent URI:=`*URI*`]]` (on article pages) |
| `owl:equivalentClass` *URI* | `[[equivalent URI:=`*URI*`]]` (on category pages) |
| `owl:equivalentProperty` *URI* | `[[equivalent URI:=`*URI*`]]` (on relation/attribute pages) |

While this provides a convenient way of mapping ontologies, this new expressive feature must be handled with care. The reason is that it yields numerous ways of creating undesired OWL/RDF specifications with the wiki. For example, one can create logical inconsistencies by declaring a non-empty wiki-class equivalent to `owl:Nothing`. While this can still be disallowed rather easily, it is also possible to make statements that are not possible in OWL DL but only in the undecidable OWL Full variant of the language. For example, one could assign the same URI to both a class and an individual, or one could even assign the URIs of language constructs such as `rdf:type` to wiki elements. Clearly, this often leads to a meaningless specification.

Both logical inconsistency and the usage of the undecidable language OWL Full prevent effective query answering over the wiki knowledge, and we therefore suggest to use `equivalent URI` only on pages that cannot be edited by arbitrary, possibly anonymous users. Since our use cases evolve around wikis of limited size that are used in a closed community, it is also realistic to assume that problems can be avoided by instructing users accordingly. Another option, discussed below, is to build appropriate checks into the wiki. One could also disallow the use of `equivalent URI` and require an external ontology where a mapping between the URIs of the wiki knowledge base and the target ontology is declared, but this would need to be administrated manually and outside of the wiki.

## 4 Usage of Ontological Data

In this section, we discuss concrete ways of using the knowledge of an external ontology and the associated challenges.

### 4.1 Inferencing Tasks

The principal purpose of specifying complex external schema information is to restrict the possible (formal) interpretations of the wiki's contents. Two major practical uses of this information are to impose constraints on the data within the wiki, and to infer additional information about this data.

Constraints are imposed by stating that certain situations are logically inconsistent. For a simple example, one could state that the categories (classes) *Workshop* and *Person* are disjoint. When a user tries to classify an article to belong to both classes, the system detects the inconsistency and can react appropriately. Typically, the reaction should involve a warning, and, if possible, an explanation. In general, inconsistency in OWL DL can arise for very complex reasons, and huge parts of the ontology can be involved in the deduction. The possible solutions to the resulting computational challenges are discussed in the next section. For now, let us note that the high complexity of the required reasoning also means that humans are typically not able to detect all consistencies easily for a complex schema. Since inconsistent ontologies do not represent any knowledge under the OWL semantics, it is clear that consistency checking is required for all applications discussed in Sect. 2.

Another application for ontological inferencing is automatic classification of instances, i.e. the assignment of OWL classes (wiki categories) to instances in the wiki. Automatic classification helps to structure the contents of the wiki, and therefore facilitates the creation and management of the wiki's content. In the case of MediaWiki, categories can be used to browse the wiki pages, and help to organise the wiki in a hierarchical way.

Besides this, classification yields a mechanism for users to understand the consequences of the external schema information. For example, the schema of a company's human resource ontology might define a property *supervised by* to describe a relationship between employees and interns. If a user now erroneously states that a certain project is supervised by somebody, then the automatic classification will classify the project as a *person*. This supports the user in immediately detecting the misconception even before further statements generate an actual inconsistency.

In both cases more expressive ontologies than those that are expressible with the current means of the Semantic MediaWiki are required. Therefor the system needs to refer to an external OWL ontology, that holds these more expressive ontologies. In Sect. 5.3 we describe an example where such an architecture is employed.

### 4.2 Practical Scalability of Inferencing

Useful as they might be, complex OWL DL inferences also impose huge scalability challenges. Reasoning in OWL DL is NExpTime complete, and thus appears to be unsuitable for a continuously growing knowledge base like a wiki. Yet, there are various possibilities to build practical systems that can still solve non-trivial reasoning tasks. We give a brief overview in the following.

First of all, it must be noted that current OWL DL reasoners are highly optimized systems that can already deal with complex ontologies to a certain complexity and size. So for small wikis, as they arise in many of the use cases from Sect. 2, it is quite feasible to employ standard software systems. Also, many of the system, while (N)ExpTime in the worst-case, have good "pay as you go" characteristics, so that simple ontologies require significantly less resources.

While restricting to simple ontologies can lead to improved computation time in OWL DL reasoners, it will not suffice for larger wikis since it is still not tractable. For such a system, the only feasible solution currently seems to be to restrict the expressive power of the employed ontology language. Various choices exist[4], and OWL fragments such as *Horn-$\mathcal{SHIQ}$* [13] and *EL++* [3] are quite expressive while still providing polynomial complexity for important reasoning tasks.

Another approach that is very interesting for reasoning in a wiki environment has been developed for the *KAON2* system [17]. Reasoning there is divided into two separate processing steps: the complex terminological part of an ontology is preprocessed in a first stage, whereas query answering over large ABoxes is performed in a second stage. The preprocessing is still ExpTime complete, but query answering can be performed in NP (wrt. the size of the ABox). If the ontology is restricted to Horn-$\mathcal{SHIQ}$, query answering is even possible in polynomial time. Evaluations show that KAON2 performs very well for ontologies with a fixes schema and large amounts of instance data [18] – which is exactly the scenario we address in this work.

[4] See `http://owl-workshop.man.ac.uk/Tractable.html` for an overview.

### 4.3 User Interface Enhancements

Finally, some uses of ontological background knowledge are possible without implementing complex reasoning. In particular, one can reuse the predefined vocabulary and known simple schema information to make suggestions during editing. In its simplest form, the wiki could warn users when using annotations that do not belong to the vocabulary that was imported into the wiki. This is not feasible in semantic wikis that are not based on an existing schema, since users there must be able to add new annotations to the system.

An example for more elaborate user interface enhancements are mechanisms for suggesting appropriate annotations to users. This can be realized syntactically by comparing user inputs to existing labels (e.g. in the form of autocompletion). On the other hand, semantic structure such as the specification of property domains can be exploited as well. For example, when editing a page on *SemWiki2006*, the classification of this article as a *workshop* might be evaluated by suggesting the properties *organized by* and *paper deadline* to the user.

## 5 Implementation

### 5.1 Ontology import

Our implementation of the ontology import is based on the Semantic Mediawiki extension to the Mediawiki[5] software. For importing the ontology we used the RDF API for PHP package RAP[6]. RAP is a software package for parsing, searching, manipulating, serializing, and serving RDF models.

The ontology import extension loads an ontology using RAP. Each statement in the ontology is checked for two criteria. First, it is checked whether the statement can be represented within the wiki. As we have seen in Sect. 3.1, we primarily regard the assertional information with the ontology, whereas complex class description will not be imported to the wiki. Second, it is checked if the statement already known to the wiki. If so, importing it is obviously unnecessary. The wiki then presents a list of all statements that passed both checks to the user, and allows her to choose which statements to import. For statements with subjects yet unknown to the wiki, the import will create articles, categories, relations, and attributes as appropriate. The text created to capture relations between articles is still very crude and recognizable as being machine created. In the future we imagine sophisticated text generation techniques [6] to augment this functionality. The name of a page is derived from the entity's label, or, if none exists, from the local part of the qualified name [7] of the entity. The comment is used as an initial text for the page, besides the generated text mentioned before.

Although the extension was first designed solely to allow the initial set-up of a wiki with an ontology, it also allows to upload ontologies at a later stage. This would allow to continuously add data from an ontology to a wiki, especially since the wiki is able to detect which parts of the ontology are missing.

[5] `http://mediawiki.org`
[6] `http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/`

In order to keep the ontology import simple, we refrained from using inferencing and mapping techniques on the ontology to be imported. We only take into account the basic relations described in table 1, and further only those explicitly stated in the ontology. As the wiki can be linked to any external ontology, inferencing can be part of the later stage as described in the next section and the example below. It is also possible to materialize derived statements within the ontology, or to create materialized mappings, prior to the import, if the wiki should represent this knowledge explicitly.

The implementation of the ontology import is finished and is part of Semantic Mediawiki in the current version.[7] As an ontology import potentially leads to a big number of changes, access to this feature is controlled via Mediawiki's user rights management system.

### 5.2 KAON2 integration

In order to reason with the ontology, we choose the KAON2 ontology management infrastructure [12], which is written in Java. In order to employ it in the PHP-based MediaWiki, we hooked KAON2 to a simple, Java-based server – Jetty[8] – and defined a simple ad hoc protocol to expose the required features and test the functionality. We will consider DIG [4] as a possible protocol, but DIG does not yet allow for SWRL-based extensions (which are possible with the current architecture), and it is also unclear how well DIG would scale in our example. Semantic MediaWiki uses the *curl* PHP extension in order to communicate via HTTP to the KAON2 server. The results are then integrated into the output of the page.

The separation of the reasoner and the wiki engine on possibly different machines, and the loose coupling via HTTP offer several advantages. First, we do not need to reimplement an OWL reasoner in PHP, which would be a highly non-trivial task, but instead we can rely on a mature inferencing engine that is based on sound and complete algorithms. Second, with a well-defined protocol it would be possible to plug in different reasoners if required. Third, a reasoner may require significant resources (both in terms of processing and memory) which could slow down the wiki considerably. By distributing the tasks to different machines we allow the wiki to remain responsive regardless of the computations required for reasoning. We consider asynchronous calls to a reasoning service as a possibility to combine possibly expensive tasks with a responsive user interface.

Right now, there is a first proof-of-concept implementation of the KAON2 integration, but a more mature prototype is planned for the near future.

### 5.3 Example

In order to exemplify the workflow in our implementation, we consider a possible semantic wiki about the *SemWiki2006* workshop. Instead of creating a new ontology for the workshop, the wiki administrator decides to reuse the ontology for the semantic web research community, SWRC [26]. The SWRC contains numerous terms that may serve

[7] `http://sf.net/projects/SeMediaWiki`

[8] `http://jetty.mortbay.org`

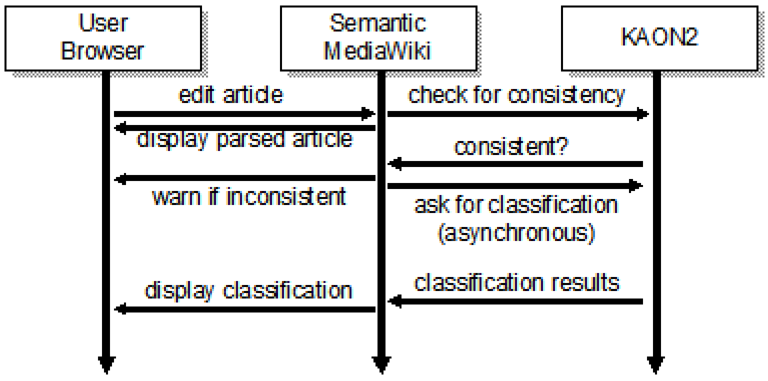


**Fig. 1.** Sequence diagram of how the SMW extension cooperates with the reasoner

as an initial base for the use case, including concepts such as *Workshop*, *Article*, and *Person*, and properties such as *organiser or chair of*, *member of PC*, and *at event*.[9]

The wiki administrator imports the SWRC ontology to have the initial relations and categories set up appropriately. Now she can start to populate the wiki manually, e.g. to add pages for the members of the programme committee. Assume that the accepted papers are already available in a machine readable format through the conference management system. In this case, it would be possible to export this data to an RDF document that reuses the SWRC ontology. Depending on the available format of the data this might require some simple conversion or mapping. However, if a suitable conversion has been implemented once, it can easily be reused for future events. The RDF version of the accepted papers can now be imported easily into the wiki to kick-start the wiki with existing ontological data. We consider it to be far easier to augment existing pages than to start from an empty wiki.

Now the wiki is set up and the wiki administrator sends its URL to interested parties. Workshop participants are encouraged to enhance the text in the wiki (as the automatically generated one does sound quite awkward as of now), or to add further details and references. One of the presenters may decide to add a link to his supervisor, typing it appropriately with a relation *supervised by* which, for the sake of the example, we assume to be part of SWRC. After this, he creates a new article about his supervisor with a few sentences and a link to the homepage – but without stating that the supervisor is indeed a *Person*.

But when the edit is saved, the automatic classification mechanism automatically adds the supervisor to the category *Person*, since this can be inferred from the ontological definition of the range of the property *supervised by*. After saving, the article on the author's supervisor would indeed state that it is a person, so that other users can find the article when browsing the according category page.

[9] For readability, we omit namespaces and camel case capitalization.

More complicated description could be added easily to the ontology. Imagine the wiki administrator adding the class *Good Paper* as one that is *endorsed by* many, or one that is *cited by* many other papers. The system then could automatically annotate "good papers" based on the semantic description of the article about the respective paper.

Also, inconsistencies in the wiki knowledge base could be detected by the reasoner: if, for example, we know that only a *Professor* may be a supervisor, categorizing a supervising person as a *PhD-Student* can be recognized as being inconsistent with the given knowledge base, and the user will be warned about this. Typically, this supports users in recognizing misconceptions about the intended usage of some relation. In the given example, another relation *tutored by* might be appropriate. Based on a built-in suggestion mechanism, the system can assist the user to find such relations without studying the background ontology.

## 6 Related work

Since the introduction of *Platypus Wiki* [9] in 2004, a number of semantic wikis have been developed. The focus of early systems such as Platypus or the *Rhizome Wiki* [24] was to provide capabilities for editing RDF content within a wiki environment. Due to the typical splitting of wiki source and RDF, importing data into these ontologies would be possible, whereas advanced features such as consistency checking and classification are mostly out of scope. The main reason is that RDF is treated as a freely edited data format without a semantic relationship to the wiki's content.

More recently, a number of new semantic wikis have been introduced. Since many of these systems are under active development, their exact capabilities and features are still evolving while this paper is written. Some of these systems, such as *IkeWiki* [22], adhere to the strict separation of semantic content and wiki text. In contrast, some wikis integrate semantic annotations into the wiki source code, as it is done in *WikSAR* [1, 2], and *Semantic MediaWiki* [27]. Finally, some wiki systems feature a WYSIWYG interface, that allows users to edit content without editing wiki markup. The only semantic wiki of this type that we are aware of is *SweetWiki*[10] which is still in prototype stage.

Only very few semantic wikis provide any support for inferencing or ontology import. The most advanced system in this respect currently seems to be IkeWiki, which allows users to import data from external ontologies and exploits schema data to provide editing support. The employed ontology language by default is (a subset of) OWL, and the system uses a reasoning engine in the back. To the best of the authors' knowledge, IkeWiki does not employ a complete OWL reasoner, but it provides partial reasoning support to structure wiki content and to browse data.

IkeWiki differs from our system in various ways. First of all, IkeWiki emphasizes the use of external ontologies much more than Semantic MediaWiki. The wiki can be initialized with multiple ontologies, and users choose annotation elements from the according namespaces. In contrast, Semantic MediaWiki uses external ontologies only for RDF export, and users work with internal identifiers. These identifiers might be equal to the abbreviated URIs in an external ontology, but it is also possible to choose

[10] `http://wiki.ontoworld.org/wiki/SweetWiki`

more human-readable names, e.g. on a wiki that is run in German instead of English.[11] On the other hand, IkeWiki provides user-friendly special purpose interfaces for editing annotations, the implementation of which is facilitated by the wiki's separation of RDF and text.

IkeWiki's stronger reference to existing ontologies implies further conceptual differences. For instance, ontological concepts in IkeWiki must be declared or imported before usage. In Semantic MediaWiki, many annotations can be used without prior declaration – necessary URIs are generated from the URL of the wiki system. Declaring references to external ontologies is an added feature that is not enabled by default, since we consider it as problematic in public wikis. In particular, free access to elements from the RDFS and OWL vocabulary enables users to generate OWL Full ontologies and logical inconsistencies, which is an interesting combination since one cannot generally detect inconsistencies in OWL Full automatically. On the other hand, an even tighter integration of *selected* ontologies can be an interesting feature that is planned for Semantic MediaWiki as well.

Other than IkeWiki, we are only aware of *KaukoluWiki*[12] as another wiki system that features ontology import and inferencing. There, the primary ontology language is RDFS. Various related features are planned or currently implemented, but we are not certain about the current status and integration of reasoning support.

Finally, Semantic MediaWiki appears to be the only wiki with extended support for XML Schema (XSD) datatype annotations. The current implementation allows users to provide data values in various syntactic forms, transforms values into XSD conformant representations, and incorporates units of measurement into the RDF export.

## 7 Conclusions

We have developed and implemented a semantic wiki that meets several requirements:

- it refers to existing ontological vocabularies,
- it incorporates schema information and constraints from external ontologies,
- it exports data in a standard ontology language,
- it imports data from external ontologies, so that it is represented in and editable through the wiki.

In order to meet these requirements, we enabled the wiki to test its knowledge base for inconsistent facts and to classify articles automatically. We also showed how the ontological knowledge could be used for enhancing the user interface. In order to design the system, we have taken results from research in the field of scalable reasoning over web ontologies into account and geared the system towards a fast and interactive user experience.

Various tasks still are left open for future work. Better natural language generation techniques [6] would considerably improve the ontology import function. The inconsistency check right now only tells us that the ontology is inconsistent, but not in what

[11] Semantic MediaWiki supports easy internationalization and is available in various languages.
[12] `http://kaukoluwiki.opendfki.de`

way and how to resolve the inconsistency (or even offer automatic correction capabilities) [11, 21, 16]. It would also be interesting to synchronize a wiki knowledge base with a dynamic ontology outside the wiki, i.e. a continuous import from an ontology. Finally, although we have presented some ideas on how the ontological background knowledge can be reused to enhance the user experience, we could not yet test or properly evaluate how users interact with such a system. Although we are positive that it will be of great help to the user, there are still some pitfalls: frequent inconsistencies or inexplicable automatic classification could lead to frustration about the "stubborn" system.

The paper has also shown that we can use ontologies for both exporting and importing knowledge from and to a wiki. Although the exchange format is far from being as complete or even useful as other wiki exchange syntaxes, it has the advantage of being based solely on the W3C standards RDF for data exchange and OWL for the vocabulary, and thus may not only interact with other wikis, but also with the ever growing set of ontology-based tools. The inconsistency check and automatic classification presented in this paper are a mere example of this.

Whereas previous work on Semantic Mediawiki [27] has presented a way to turn wikis, and especially the Wikipedia, into a major foundation of the Semantic Web [5], in this paper we propose the inverse approach: reusing Semantic Web resources for a wiki. These two technologies together allow to integrate wikis fully into the Semantic Web idea.

## Acknowledgments

Research reported in this paper has been partially financed by the European Union in the IST project SEKT (IST-2003-506826), `http://www.sekt-project.com`, and by the German BMBF project "SmartWeb." The views expressed in this paper are those of the authors and not necessarily those of the projects or their institute as a whole. We would like to thank our colleagues for fruitful discussions.

## References

1. D. Aumüller. Semantic authoring and retrieval in a wiki (WikSAR). In *Demo Session at the ESWC 2005*, May 2005.
2. D. Aumüller and S. Auer. Towards a semantic wiki experience – desktop integration and interactivity in WikSAR. In *Proc. of 1st WS on Semantic Desktop, Galway, Ireland*, 2005.
3. F. Baader, S. Brandt, and C. Lutz. Pushing the EL envelope. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 364–369, 2005.
4. S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris. A proposal for a description logic interface. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 33–36, 1999. CEUR Workshop Proceedings `http://ceur-ws.org/Vol-22/`.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, (5), 2001.
6. K. Bontcheva. Generating tailored textual summaries from ontologies. In A. Gómez-Pérez and J. Euzenat, editors, *2nd European Semantic Web Conference, ESWC 2005*, volume 3532 of *LNCS*, pages 241–256, Heraklion, Crete, Greece, May 2005. Springer.

7. T. Bray, D. Hollander, and A. Layman. Namespaces in XML. W3c recommendation, World Wide Web Consortium, January 1999.
8. D. Brickley and L. Miller. FOAF vocabulary specification. Namespace Document 27 July 2005 ('Pages about Things' Edition).
9. S. E. Campanini, P. Castagna, and R. Tazzoli. Platypus wiki: a semantic wiki wiki web. In *Semantic Web Applications and Perspectives, Proceedings of 1st Italian Semantic Web Workshop*, Dec 2004.
10. W. Cunningham and B. Leuf. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, 2001.
11. P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 353–367. Springer, November 2005.
12. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proceedings of the KR2004*, pages 152–162. AAAI Press, 2004.
13. U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. 19th IJCAI*, pages 466–471, 2005.
14. A. Kalyanpur, B. Parsia, and J. Hendler. A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems*, 1(1), 2004.
15. M. Krötzsch, D. Vrandečić, and M. Völkel. Wikipedia and the Semantic Web – the missing links. In *Proc. of the 1st Int. Wikimedia Conf., Wikimania*, Aug 2005.
16. S. J. Lam, D. Sleeman, and W. Vasconcelos. Retax++: a tool for browsing and revising ontologies. In *Proc. of ISWC 2005 Demo Session*, Galway, Ireland, November 2005.
17. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Germany, 2005.
18. B. Motik and U. Sattler. Practical DL reasoning over large ABoxes with KAON2, 2006. available at `http://kaon2.semanticweb.org/`.
19. N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
20. E. Oren. Semperwiki: a semantic personal wiki. In *Proc. of 1st WS on The Semantic Desktop, Galway, Ireland*, 2005.
21. B. Parsia, E. Sirin, and A. Kalyanpur. Debuging OWL ontologies. In *Proc. of the 14th World Wide Web Conference (WWW2005)*, Chiba, Japan, May 2005.
22. S. Schaffert, A. Gruber, and R. Westenthaler. A semantic wiki for collaborative knowledge formation. In *Semantics 2005*. Trauner Verlag, 2005.
23. M. K. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide, 2004. W3C Recommendation.
24. A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20:87–91, 2005.
25. Y. Sure, J. Angele, and S. Staab. Ontoedit: Multifaceted inferencing for ontology engineering. *Journal on Data Semantics*, 1(1):128–152, November 2003. LNCS 2800.
26. Y. Sure, S. Bloehdorn, P. Haase, J. Hartmann, and D. Oberle. The SWRC ontology - Semantic Web for Research Communities. In C. Bento, A. Cardoso, and G. Dias, editors, *Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005)*, volume 3803 of *LNCS*, pages 218 – 231, Covilha, Portugal, Dec 2005. Springer.
27. M. Völkel, M. Krötzsch, D. Vrandečić, H. Haller, and R. Studer. Semantic Wikipedia. In *Proc. of the 15th int. conf. WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, May 2006.
28. D. Vrandečić, H. S. Pinto, Y. Sure, and C. Tempich. The DILIGENT knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, Oct 2005.
29. Wikipedia. History of Wikipedia – Wikipedia, The Free Encyclopedia, 2006. Online version of 28 March 2006.

# Kaukolu: Hub of the Semantic Corporate Intranet

Malte Kiesel

DFKI GmbH, Kaiserslautern, Germany,
malte.kiesel@dfki.de

**Abstract.** Due to their low entry barrier, easy deployment, and simple yet powerful features, wikis have gained popularity for agile knowledge management in communities of almost all sizes. Semantic wikis strive to give entered information more structure in order to allow automatic processing of the wiki's contents. This facilitates enhanced navigation and search in the wiki itself as well as simple reuse of information in external applications or for generating different views on the same information. This makes semantic wikis especially interesting for corporate intranet deployment, implementing the *Semantic Intranet*. In this paper, we will have a look at Kaukolu, an open source semantic wiki prototype, being deployed in a corporate intranet. External applications use information authored in Kaukolu, effectively forming a cluster of applications interacting and sharing data.

## 1 Introduction

Wikis become more and more important for managing content of corporate intranets, serving as a platform for information exchange and as knowledge repositories. Typically, part of the information found in corporate intranets are simply plain text (e.g., texts describing projects, templates for mails, brainstorming, tips and best practices, ...), but a major part of the content consists of structured data such as lists relating people to projects, product feature lists, publication lists, or simply collections of annotated web links. While simply being able to manage all of this different content by dropping it—as text—into the wiki is handy, a direct consequence of this is that everything in the wiki *is* essentially text and therefore cannot be imported into other applications such as spreadsheet applications, databases, or a content management system used for the external web site. So, a lot of data duplication needs to be done, ultimativaly resulting in unnecessary workload, outdated content, and inconsistencies in the data presented at different places for differing audiences.

Semantic wikis try to implement a way to establish and maintain structure of the wiki's content using semantic web technologies in order to facilitate information reuse or, in general, to facilitate accessibility of the information to automated means. Also, knowledge of the inner structure of the information contained in wiki pages can be used to enhance browsing and search in the wiki.

In section 2, we give a short overview over the basic wiki ideas and explain the shortcomings of wikis concerning structured data. In section 3, an overview over (semantic) wiki implementations is given, some semantic wiki features are explained, and several problems with existing semantic wikis are mentioned. *Kaukolu*, our implementation of a semantic wiki, is introduced in section 4, along with a walkthrough in section 5. In section 6, a number of possible enhancements of Kaukolu are presented. We end with a conclusion in section 7.

## 2 What is a Wiki?

Wikis allow a group of people to collaboratively author information using a tool that is easy to use. The main features a wiki provides are kept simple, but flexible, in order to allow for using the basic features for a number of different purposes. For example, the very basic wiki idea of editing a text page allows both editing a document and doing a discussion. Backlinks, another standard wiki feature, can be used for navigation, tagging, or grouping sets of pages. Basic content format of wikis is text so import/export functionality is limited to text formats.

Current applications of wikis range from open encyclopedias such as Wikipedia to collaborative information spaces for both open communities such as open source software projects (e.g., http://wiki.mozilla.org/—even software project management software such as Trac[1] feature wikis for documentation and information exchange) and closed communities such as company intranets.

### Structured Data Falls Through the Cracks

A major drawback of wikis is that they are intended only to edit and display plain text[2]—content that represents structured data (e.g., tables or sets of wiki pages using the same structure) can only be exported as text or HTML. These formats preserve content and looks to some degree, but the information structure (i.e., explicit knowledge of what values populate what properties of what entities) gets lost[3]. This is unfortunate: People who maintain, for example, their publication list in the wiki, have to manually re-enter the same information in other places such as the company extranet. However, duplication of information is tedious work, often leading to inconsistencies and large amounts of outdated information. Also, lack of data structure prevents us from running queries or compiling statistics against the data.

*Importing* information into a standard wiki suffers from the inability to process structured data, too. For example, while it is possible to import spreadsheet data by attaching the spreadsheet file to a wiki page, this data cannot be accessed

[1] http://www.edgewall.com/trac/

[2] Text can get formatted, but this is for the looks only.

[3] Some wikis support structured data to some degree using templates or similar features. However, typically these are proprietary approaches that provide no interoperability with other applications.

or edited in the wiki. The spreadsheet's data structure cannot be exploited and reused. Another way to import spreadsheet data would be to export the data as HTML in the spreadsheet application and import the HTML into the wiki. However, importing large amounts of HTML prevents the users from contributing to the content. Even if users dare to edit the HTML code, keeping the original spreadsheet document in sync with the changes can only be done manually.

This means that existing structures effectively cannot be maintained inside the wiki. Since structured data gets "flattened" on import, users may end up with an unstructured data repository that is difficult to manage and difficult to keep in sync with the corresponding information outside of the wiki.

How to solve this issue?– A naive solution for importing spreadsheet data, for example, would be to import *comma–separated values*[4] (CSV) into the wiki. This would at least bring one benefit of wikis, namely collaborative editing, together with structured data. However, readability and flexibility of CSV is very low. Anybody trying to introduce a new property of an item (anybody who would try to add a new column) would have to reformat all data that has been added so far. Also, anybody depending on the old CSV structure would have to get notified.

### The Semantic Wiki Idea

Semantic wikis try to overcome the problem stated above by combining semantic web standards such as RDF/S or OWL with the wiki paradigm. One idea is to *annotate* structure in the wiki by providing *metadata* for existing features such as links and pages. On the other hand, one can strive to completely *represent* the wiki content using instances of the respective ontology language.

## 3 An Overview over Several Wikis

In [9], an overview of semantic wikis and personal wikis is given, resulting in the description of *SemperWiki*, a semantic desktop wiki.

In most traditional wikis, the idea of metadata typically only appears in a very technical way. For example, in *JSPWiki*[5], metadata is added directly into the wiki text using special tags, and mostly serves the purpose of implementing access control. In *SnipSnap*[6], labels may get attached to wiki pages, serving mainly as a categorization scheme.

The semantic wiki *Platypus*[7] 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.

[4] http://en.wikipedia.org/wiki/Comma-separated_values
[5] http://www.jspwiki.org/
[6] http://snipsnap.org/
[7] http://platypuswiki.sourceforge.net/

The *Semantic MediaWiki*[8] [7] is an extension of *MediaWiki*[9], 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 datatypes such as coordinates and temperatures, along with conversion between different unit scales.

*Rhizome*[10] [14] 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 queries' 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.

*IkeWiki*[11] [13] 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.

*OpenRecord*[12] 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.

### Problems Found in Existing Semantic Wikis

Existing (semantic) wikis lack in some areas:

**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.

**Annotation complexity:** In existing semantic wikis, RDF is mainly used for *annotations*: RDF supplies semantic information that describes existing human–readable features. Since the basic blocks of wikis are pages and links between them, mapping a wiki to RDF can be done by using pages as representatives of RDF resources, with links between wiki pages denoting relations between RDF resources. This approach is typically implemented by enabling the user to attach RDF triples to wiki pages[13], but setting the subject of each triple of the page

[8] http://semediawiki.sourceforge.net/
[9] http://mediawiki.sourceforge.net/
[10] http://rx4rdf.liminalzone.org/Rhizome
[11] http://ikewiki.salzburgresearch.at/
[12] http://openrecord.org/
[13] Often in terms of selecting types for links to other pages

to the page's URI[14]. 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[15], we think that it reaches its limits as soon as more elaborate ontologies and structures are used. For example, in Figure 1 the RDF structure of a *foaf:person* is depicted. In semantic wikis that identify an RDF resource with a wiki page, one wiki page of the type *foaf:person* can be used to model this data. However, if we try to model the data shown in Figure 2 (a bibtex entry represented in a format similar to the format used by the *bibtex2rdf*[16] converter), we would need four wiki pages (one wiki page per RDF resource) for just one bibtex entry.

There are other cases that make the problem even more obvious. 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.

In general, we believe that imposing a structure on wiki contents due to technical reasons is against the wiki way. Users should be free to use whatever page structure they want. Structured data, as is RDF, is only *another view* on the wiki content.

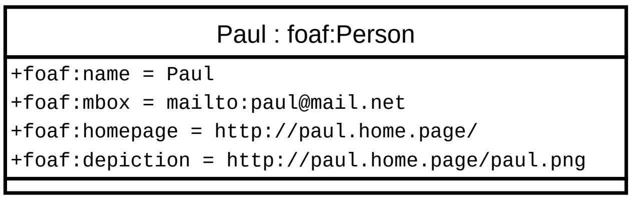

Paul : foaf:Person
+foaf:name = Paul
+foaf:mbox = mailto:paul@mail.net
+foaf:homepage = http://paul.home.page/
+foaf:depiction = http://paul.home.page/paul.png

**Fig. 1.** A foaf:person.

**Smooth migration:** 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.

[14] The point is that in this approach triples are bound to pages because of their subject URI. It does not really matter whether this URI is the URL the wiki page can be browsed at, a separate "wiki page concept URI", or an arbitrary URI.

[15] For some applications such as Gnowsis [12], this approach is followed by our wiki implementation, too [6].

[16] http://www.l3s.de/∼siberski/bibtex2rdf/

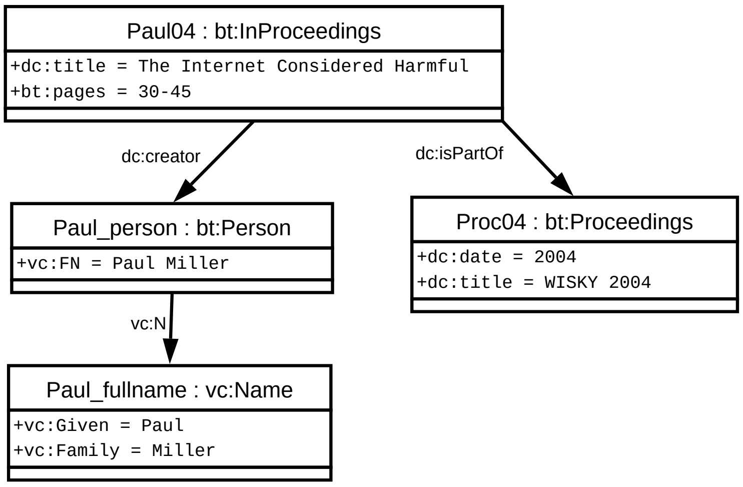


**Fig. 2.** A bibtex RDF entry.

**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 [10].

## 4 Why Kaukolu is Different

*Kaukolu*[17], our implementation of a semantic wiki, builds on JSPWiki[18] and Sesame 2[19]. It differs in several aspects from the existing semantic wikis.

- no restrictions are imposed on RDF triples attached to a page (triple's subjects are *not* fixed)
- arbitrary RDF(S) files can be imported
- aliases can be defined for resources and predicates
- autocompletion supports the user when formalizing content

We currently use Kaukolu internally in our department. An evaluation in another company will take place this year.

### 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 the triple is located at. This solves the issues explained in section 3 and allows for more complex RDF data.

[17] Available at http://kaukoluwiki.opendfki.de/ including sources
[18] http://www.jspwiki.org/
[19] http://www.openrdf.org/

### 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, we will explain this later. 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.

### 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 something like *(this) 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. In Figure 3, 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 [3].

### Autocompletion for Both Semantic and Non–Semantic Content

Of course, even with 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 *Paul knows*, with *Paul* 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 "InfoOnPaul" and "InfoOnSarah" pages in the wiki, those both page names are suggested.

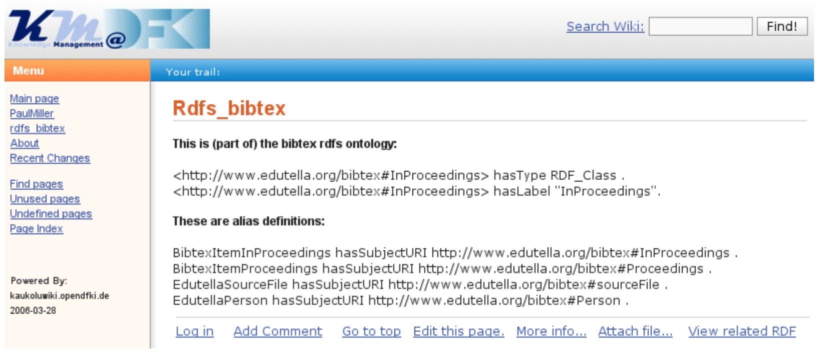
**All Standard Wiki Features are Implemented**

Most other semantic wikis have been rewritten from scratch and therefore miss several standard wiki features such as file attachments, access control, plugin support, or support for multiple backends. Kaukolu is based on JSPWiki[20], an established wiki that is quite feature–complete.

## 5 Kaukolu in Practice

In the following, we will demonstrate some of Kaukolu's features. Paul Miller will import an ontology describing bibtex entries into Kaukolu, and add a new publication item to a wiki page (ontology–driven autocompletion will help here). Then, Paul will export the bibtex RDF generated into an external application *RDFHomepage* [5] which generates an HTML page containing a publication list.

**Ontology Import:** In Figure 3, we see a wiki page holding the bibtex RDFS ontology used for our publication list. Any RDFS ontologies can get imported. On import, they will be converted to RDF wiki syntax which is similar to N3 [1]. Ontologies can be created using ontology editors such as Protégé-2000 [8]. Note that one can now collaboratively edit the ontology within Kaukolu. Export to RDFS is also possible using the "View related RDF" button to the lower right. Updating the ontology can be done either directly in the wiki or by re–importing the ontology.



Rdfs_bibtex

**This is (part of) the bibtex rdfs ontology:**

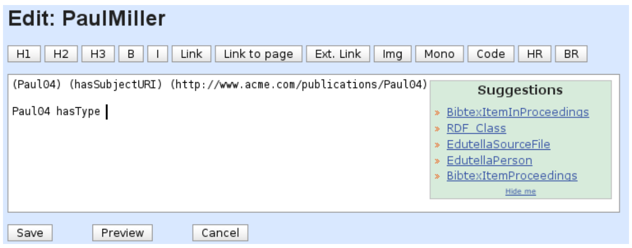
```
<http://www.edutella.org/bibtex#InProceedings> hasType RDF_Class .
<http://www.edutella.org/bibtex#InProceedings> hasLabel "InProceedings".
```

**These are alias definitions:**

```
BibtexItemInProceedings hasSubjectURI http://www.edutella.org/bibtex#InProceedings .
BibtexItemProceedings hasSubjectURI http://www.edutella.org/bibtex#Proceedings .
EdutellaSourceFile hasSubjectURI http://www.edutella.org/bibtex#sourceFile .
EdutellaPerson hasSubjectURI http://www.edutella.org/bibtex#Person .
```

**Fig. 3.** The wiki page holding the bibtex RDFS ontology.

**Add a Publication Entry:** In Figure 4, we see a user adding a publication entry to his wiki page. Since *hasType* (corresponding to *rdf:type*) implies that



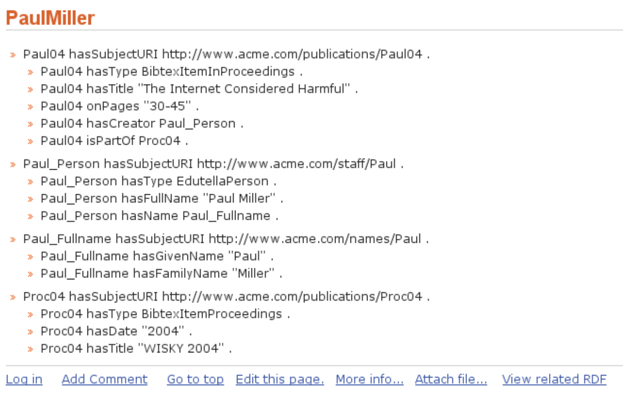
```
(Paul04) (hasSubjectURI) (http://www.acme.com/publications/Paul04)

Paul04 hasType
```

Suggestions
- BibtexItemInProceedings
- RDF_Class
- EdutellaSourceFile
- EdutellaPerson
- BibtexItemProceedings

**Fig. 4.** Ontology–based autocompletion in action.

the triple's object will be of type *rdfs:Class*, only instances of *rdfs:Class* are displayed in the autocompletion suggestion box.

The complete page describing Paul's publication item is shown in Figure 5. Note that one can use standard wiki markup along with the RDF extensions (a bullet list is used).



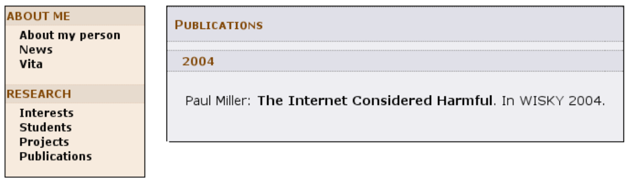
PaulMiller

- Paul04 hasSubjectURI http://www.acme.com/publications/Paul04 .
  - Paul04 hasType BibtexItemInProceedings .
  - Paul04 hasTitle "The Internet Considered Harmful" .
  - Paul04 onPages "30-45" .
  - Paul04 hasCreator Paul_Person .
  - Paul04 isPartOf Proc04 .
- Paul_Person hasSubjectURI http://www.acme.com/staff/Paul .
  - Paul_Person hasType EdutellaPerson .
  - Paul_Person hasFullName "Paul Miller" .
  - Paul_Person hasName Paul_Fullname .
- Paul_Fullname hasSubjectURI http://www.acme.com/names/Paul .
  - Paul_Fullname hasGivenName "Paul" .
  - Paul_Fullname hasFamilyName "Miller" .
- Proc04 hasSubjectURI http://www.acme.com/publications/Proc04 .
  - Proc04 hasType BibtexItemProceedings .
  - Proc04 hasDate "2004" .
  - Proc04 hasTitle "WISKY 2004" .

**Fig. 5.** The complete bibtex item.

**Export Publication List to External Application:** In Figure 6, we see an HTML page generated by *RDFHomepage* using the RDF formulated on the user's wiki page. Note that the page generated is intended for external audiences and cannot be edited. The RDF created in Kaukolu, a Java–based application, is processed by RDFHomepage, a PHP–based application.

[20] http://www.jspwiki.org/



Paul Miller: **The Internet Considered Harmful**. In WISKY 2004.

**Fig. 6.** The publications page as generated by RDFHomepage.

### Further Applications

Kaukolu can be used for different ontologies, too. For example, RDFHomepage not only generates a publication list, but also uses information contained in an *organizational repository* (formulated in RDF/S) for generating a list of projects the respective person using RDFHomepage participates in. Kaukolu allows to collaboratively maintain the organizational repository which up to now has been maintained centrally in our department.

## 6 Future Work

### Limitations of Kaukolu

Kaukolu currently does not provide means to generate a number of RDF constructs. Most notably, it cannot deal with RDF containers such as bags, sequences, and alternative values yet. While this is a drawback when authoring or importing RDF, expressivity is not touched by leaving these features away since these constructs can be substituted by RDF lists and multiple–valued properties in most cases. Alternatively, Kaukolu's wiki syntax for creating RDF triples could be extended to incorporate handling of containers.

Another limitation of Kaukolu is its inability of generating blank nodes. On import, any blank nodes may get assigned a random and unique URI. However, Kaukolu currently never exports blank nodes.

### Planned Features

We have identified several ways in which Kaukolu may be improved.

**Use of RDF metadata within Kaukolu:** While currently the main reason for generating RDF is its usage in external applications, Kaukolu can use RDF data for navigation: If RDF data is attached to a page, it can be shown in a navigation sidebar. We are aware that this is only a very basic feature. Further possibilities for using annotations within the wiki would be to use them for search or feature a reverse translation of RDF to wiki markup or HTML directly for display which would allow to create customized views on formalized wiki content ("Show a list of all persons working in project X here" or "Show a list of all properties of software X here"). Previous work on this topic includes Fresnel [2] and Haystack [11].

**Proposal of RDF metadata from natural texts:** We plan to use the SProUT natural language processing module [4] in Kaukolu. This will allow to generate RDF from natural text and partly eliminate the dependence on formulating triples directly. Also, this will be a great feature for switching from a standard wiki to Kaukolu since existing texts can get mined for RDF data then automatically. While we do not expect the extracted data to be perfect, we believe that it will serve as a start. SProUT has been used in the SmartWeb project[21] for extracting RDF instances from natural texts in the sports domain.

**Creation of RDFS instances:** Creating instances of RDFS classes by entering the corresponding RDF triples is quite time–consuming. There should be a more comfortable way to create new instances. A lightweight way would be to create default triples according to the ontology and to let the user fill in object values.

**Better embedding of RDF triples:** Currently, any RDF expressed in Kaukolu must be part of a wiki page's text and therefore gets displayed when the page gets rendered. Since Kaukolu allows aliases for subjects, predicates, and properties to get embedded in natural text (as in "*PaulMiller* came to *know SarahMiller* in 2005"), there is no absolute need of separating the parts of the text that represent RDF from the remaining text. However, in practice embedding triples often leads to awkward sentences. A more flexible way of embedding RDF–generating content into wiki pages seems desirable. In the future, we will implement a feature that allows to separate RDF–generating statements from normal text. Features to generate these statements as well as features to keep them in sync with normal text will be added.

## 7 Conclusion

In this paper, we gave an overview over the ideas of (semantic) wikis and implementations that are available. Our semantic wiki prototype Kaukolu addresses some of the shortcomings of existing semantic wikis and is intended for intranet usage. Its main features are its ability to import and export RDF and its ontology–supported autocompletion feature which relieves users from having to know the ontologies used letter by letter. We believe that export *and* import of structured data are essential features for a semantic web application. A demonstration of theses features was given, using a publication list as an example for structured content which gets formulated in Kaukolu and used in a separate application, illustrating Kaukolu's export functionality. Finally, we discuss some issues in Kaukolu, along with ideas how to address them.

[21] http://www.smartweb-project.de/

## 8 Acknowledgments

This work has been supported in part by the NEPOMUK project, which is funded by the IST Programme of the European Union under grant FP6-027705.

## References

1. Berners-Lee, T. Getting into RDF & Semantic Web using N3, 2000.
2. Bizer, C., Lee, R., and Pietriga, E. Fresnel - Display Vocabulary for RDF, 2005.
3. Buitelaar, P., Sintek, M., and Kiesel, M. Integrated Representation of Domain Knowledge and Multilingual, Multimedia Content Features for Cross-Lingual, Cross-Media Semantic Web Applications. In *Proceedings of the ISWC 2005 Workshop on Knowledge Markup and Semantic Annotation* (2005).
4. Drozdzynski, W., Krieger, H.-U., Piskorski, J., Schäfer, U., and Xu, F. Shallow processing with unification and typed feature structures — foundations and applications. *Künstliche Intelligenz 1* (2004), 17–23.
5. Grimnes, G., Schwarz, S., and Sauermann, L. RDFHomepage or Finally, a Use For Your FOAF File. In *Proceedings of Semantic Web Scripting Workshop at ESWC06* (2006). http://rdfhomepage.opendfki.de/.
6. Kiesel, M., and Sauermann, L. Towards Semantic Desktop Wikis. *UPGRADE special issue on "The Semantic Web"* (2005).
7. Krötzsch, M., Vrandecic, D., and Völkel, M. Wikipedia and the Semantic Web — The Missing Links. In *Proceedings of Wikimania 2005* (JUL 2005), Wikimedia Foundation. http://www.aifb.uni-karlsruhe.de/WBS/mak/pub/wikimania.pdf.
8. Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., and Musen, M. A. Creating Semantic Web contents with protege-2000. *IEEE Intelligent Systems 16*, 2 (2001), 60–71.
9. Oren, E. SemperWiki: A Semantic Personal Wiki. In *Proceedings of the 1st Semantic Desktop Workshop at the ISWC2005* (2005).
10. Prud'Hommeaux, E., and Seaborne, A. SPARQL query language for RDF. World Wide Web Consortium, Working Draft WD-rdf-sparql-query-20060220, Feb. 2006.
11. Quan, D., Huynh, D., and Karger, D. R. Haystack: A platform for authoring end user semantic web applications. In *International Semantic Web Conference* (2003), pp. 738–753.
12. Sauermann, L. Gnowsis semantic desktop iswc2004 demo. In *Proceedings of the International Semantic Web Conference 2004* (2004).
13. Schaffert, S., Gruber, A., and Westenthaler, R. A Semantic Wiki for Collaborative Knowledge Formation. In *Semantics* (2005).
14. Souzis, A. Rhizome Position Paper, 2004. http://rx4rdf.liminalzone.org/FOAFPaper.

# Creating and using Semantic Web information with Makna

Karsten Dello, Elena Paslaru Bontas Simperl, and Robert Tolksdorf

Freie Universität Berlin, Institut für Informatik,
AG Netzbasierte Informationssysteme, Takustr. 9, D-14195 Berlin, Germany
{dello,paslaru,tolk}@inf.fu-berlin.de
http://www.ag-nbi.de/

**Abstract.** Combining Wiki and Semantic Web technologies is considered by many members of the two communities as a promising alternative to current approaches for collaboratively creating and using information on the Web. The user-friendliness of the former as regarding multi-site content generation and the power of semantic technologies as w.r.t. organizing and retrieving knowledge are likely to complement one another towards a new generation of Web-based content management systems. Our system Makna (stands for *"knowledge"* in Indonesian) elaborates on this ideas by extending the Wiki engine JSPWiki with generic, easy-to-use ontology-based components for authoring, querying and browsing Semantic Web information.

## 1 Introduction

Combining Wiki and Semantic Web technologies is considered by many members of the two communities as a promising alternative to current approaches for collaboratively creating and retrieving information on the Web. The success of the former is primarily due to their simplicity and user-friendliness; a Wiki is a hypermedia system consisting of a collection of interconnected Web documents, which can be accessed, revised and extended by arbitrary parties with the help of a simplified hypertext syntax. However, while Wiki systems are targeted at collaborative *authoring*, they still need means to *organize and retrieve* the created content. Though its importance is widely acknowledged among Wiki solution providers, this issue is marginally addressed in current implementations, which restrict to organizing Wiki articles/pages according to a manually defined (and maintained) set of categories.

The Semantic Web provides the technological infrastructure to alleviate this situation. RDF statements can be applied to enhance the semantics of Wiki pages and of the links between them, while ontologies and associated reasoning services are a valuable extension to currently employed plain topic classifications and information retrieval capabilities. Figure 1 illustrates this idea with a simple example: the relationship between the Wiki article introducing the American actor Humphrey Bogart and the one describing his home town New York is related by the typed link `livedIn`, a property defined in a particular domain ontology.

The interconnected articles themselves are annotated with typing information, according to which they are classified as an instance of the concept `Actor` and `City`, respectively.

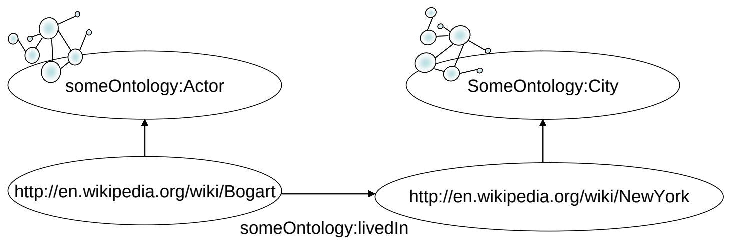

someOntology:Actor
SomeOntology:City
http://en.wikipedia.org/wiki/Bogart
http://en.wikipedia.org/wiki/NewYork
someOntology:livedIn

**Fig. 1.** Typed pages and links

Complementarily the Semantic Web can utilize Wikis as support tools in various application scenarios, the most important being probably distributed knowledge engineering and semantic content generation. The usage of Semantic Web technologies is currently inconceivable without a high level of IT expertise, with the consequence that the amount of Web information available in languages like RDF(S) and OWL is minimal compared to the dimensions of the traditional Web. In order for the Semantic Web to overcome this technical barrier of entry there is a need for tools which allow humans to contribute to the creation of Semantic Web content transparently from the underlying technologies.

In this paper we elaborate on these ideas by implementing Makna, a Wiki-based tool for distributed knowledge engineering.[1] Makna extends an existing Wiki engine (in our case the Java-based JSPWiki system)[2] with generic, easy-to-use ontology-driven components for collaboratively authoring, querying and browsing Semantic Web information. In contrast to similar attempts for combining the two fields of research, our system explicitly focuses on *the immediate and comfortable exploitation of the semantic content*, while implementing many key features of hypermedia systems targeted at supporting distributed knowledge engineering processes [4, 14, 18, 21].

The remainder of this paper is organized as follows. Section 2 specifies a set of core requirements to be fulfilled by the Semantic Wiki implementation. Building upon these we discuss the design principles and the architecture of the Makna system in Sections 3 and 4, respectively. Details on the implementation are presented in Section 5. We compare our solution with related approaches in Section 6 and conclude with a summary of future research and development directions in Section 7.

[1] `http://makna.ag-nbi.de`
[2] `http://www.jspWiki.org/`

## 2 Requirements analysis

A Wiki system can be understood as a collection of simplified hypertext documents which are read and edited by a community of Wiki users[5].

In relation to the rapid growth of the amount of information available within a Wiki system, its users need means to locate the relevant content, prior to accessing, creating or modifying it in any form. This task is commonly approached with the help of keyword-based information retrieval mechanisms. Mainstream Wiki implementations maintain a full-text index of their contents and provide users with the possibility of formulating unstructured queries in a simple query interface.[3]

The limitations of this simplistic approach become visible when Wiki users necessitate very specific information on a particular topic or information which is distributed over multiple pages. These use cases can not be successfully realized using keyword-based search algorithms executed on a plain set of textual documents; in turn they require heuristics which take into account the semantics of the underlying domain and of the links between individual Wiki contents. We illustrate these issues by means of two simple examples in the movies domain. A query like `Who was the actor Humphrey Bogart married to?` by Wikipedia delivers—if formulated properly—a list of movies starring the actor, as well as the Wiki article containing his biography. Though the encyclopedia contains a page dedicated to the prominent wife of Humphrey Bogart, the actress Lauren Bacall, this information can not be exploited without human intervention, in the absence of machine-processable domain knowledge and of an adequate link mechanisms between the two pages. Further on, a search for `American actors` in the same information repository returns a set of Wiki entries containing actor biographical information, but also dozens of movies or other types of articles. In order to improve the precision of such general queries, Wiki systems need to implement mechanisms for typing Wiki pages according to pre-defined schemes (such as ontologies).

To summarize, typing Wiki articles and the links interrelating them by means of ontologies enables the implementation of advanced content- and structure-oriented retrieval facilities.

Another aspect to consider is context-based navigation. Wiki systems rapidly tend to contain an impressive number of internal links.[4] If the relation between Wiki pages would be represented in a precise and formal way—and that is exactly what Semantic Web technologies provide—the Wiki engine could provide facilities to semantically navigate between meaningfully related resources, such as the articles dedicated to the aforementioned actors.

Besides domain-focused search heuristics and semantic navigation, ontology-driven technologies provide additional advantages to organizing and retrieving

[3] Often users are re-directed to a page by search engines like Google who have spidered the Wiki contents, integrating it into their own index.

[4] As stated at `http://stats.Wikimedia.org/DE/TablesWikipediaEN.htm` the English Wikipedia contained 19.3 million internal links to 922.000 articles as of November 2005. This is a rate of approximately 21 out- and in-links per article.

knowledge in a Wiki system. If Wiki contents were classified according to an ontology, this could be utilized as a commonly agreed query vocabulary, enabling users to formulate more precise and structured queries. Further on, formally represented ontologies in correlation with reasoning services are an ideal means to operationalize various quality assurance procedures, which become indispensable in any loosely coupled collaborative content authoring endeavor [1]. A third use case is the automatic link computation; given a semantic classification of Wiki articles, the system could consult the ontological content in order to automatically detect links to semantically related resources. According to a recent study by [1] the lack of adequate support for link creation and management is one of the key usability problems encountered within Wiki systems.

The aforementioned issues correspond to well-established requirements for so-called *"forth-generation hypermedia systems"*. As stated in [4, 21] advanced Web-based hypermedia systems—including Wikis—can take benefit from implementing features such as[5]

- typed annotated nodes
- typed attributed links
- computed links
- personalized links
- content- and structure-based search
- content- and structure-based navigation
- multiple view presentation

On the other hand, while extending Wiki technologies with semantics definitely contributes to the realization of advanced search and browse facilities, it also imposes several *additional* system requirements at both functionality and usability level. First, the system should provide a concept for the consistent authoring and manipulation of *semantic information.* This issue relates to the *usage* of the system ontologies, but also to the *development* of the ontologies themselves. The former primarily requires means to reference ontological primitives within content generation tasks. The latter, however, induces the need for methodological and technological support for collaborative knowledge engineering tasks. This non-trivial research question, recently tackled in approaches such as [9, 14, 18] *solely at methodological level*, is still in its infancy in the Semantic Web community. Usability requirements mainly refer to transparency and performance issues, as semantic technologies are not necessarily popular for seriously addressing any of them yet.

## 3 System design

Accounting for the results of the requirements analysis the realization of Makna system was influenced by two categories of design decisions, which are introduced in the remainder of this section.

[5] Hypermedia systems differentiate between *nodes* and *links*. *Nodes* denote information objects (e.g. documents, or document fragments) which are connected to each other by means of *links*.

### 3.1 Minimally invasive Wiki extensions

The first category of design decisions has the objective to preserve the advantages provided by conventional Wiki technology while enriching its capabilities.

The new engine should support the same usage patterns as traditional approaches. In particular this applies for the hypertext syntax employed, which need to extended with dedicated keywords. Further on, the system should provide an easy-to-use concept for the creation and management of semantic data [1]. This should allow Wiki users to annotate plain Wiki content in terms of adding, deleting or modifying RDF statements. However, as our system is not targeted at collaborative ontology engineering yet, it is not reasonable to permit *arbitrary users* to perform changes at ontological level. Without an adequate methodological support an uncontrolled ontology evolution might have considerable implications on the way Wiki data is classified and subsequently retrieved [8, 14, 17, 18, 20]. The manipulation of the employed ontologies as well as the import of semantic instance data are therefore currently limited to a particular group of users e.g. administrators (cf. Figure 3).

Technologically the minimal invasive character of Makna is reflected in the decision to build upon existing Wiki systems instead of an implementation from scratch (as for example in [2, 15, 19]). The benefits of this decision are twofold. First re-using established implementations has clear costs and quality advantages [7, 11]. Another motivation is related to the philosophy that *"the Semantic Web is not a separate Web but an extension of the current one."*[3]. Following this idea it should also be possible to turn an existing Web application like a Wiki engine into an Semantic application by plugging in the necessary extensions.

### 3.2 Versatile use of semantic technologies

A Semantic Wiki should provide facilities for flexibly integrate and use arbitrary Semantic Web ontologies available on the Web. This implies the possibility to refer to multiple ontologies at Wiki syntax level and their seamless usage in classification, retrieval and navigation tasks.

Wiki users should be able to comfortably access available ontologies in order to suitably annotate Wiki articles. This can be achieved by providing dedicated components to facilitate the interaction with the ontological content (cf. Section 5).

Inference is another important point to consider. Reasoning services can be applied on Wiki contents in order to enhance the retrieval capabilities of the system or to exercise consistency checking in relation to specific quality assurance procedures. However, it might be necessary to restrict this feature to a carefully defined set of ontologies, as inferencing on arbitrary ontologies on the Web could under circumstances lead to serious performance problems.

Complementarily to the integration of multiple Web ontologies the engine should provide means to import and export data formalized using Semantic Web representation languages.

Finally we consider the relations between pages which are not naturally represented as hyperlinks. This can be solved in the same way in which links between non-existent pages can be created. If it is not required that statements are exclusively formulated in the Wiki syntax the problem resolves, thus making available RDF data representable as Wiki instance.

## 4 System architecture

The architecture of the Makna system is depicted in Figure 2. It consists of the Wiki engine JSPWiki, extended with several components for the manipulation of semantic data, and the underlying persistent storage mechanisms. We chose JSPWiki, because it is written entirely in Java and has a clear design structure (i.e based on the Model-View-Controller-Pattern) facilitating system extensions. The Jena API was used as de facto standard Semantic Web framework for creating, managing and querying RDF data.

Additionally we used a relational database for the persistent storage of the semantic model. The persistent storage of the Wiki pages and attachments can be provided through any of the versioning storage provider modules for JSPWiki.[6]

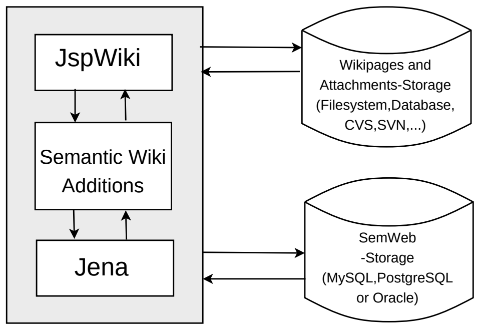

JspWiki
Semantic Wiki Additions
Jena
Wikipages and Attachments-Storage (Filesystem,Database, CVS,SVN,...)
SemWeb -Storage (MySQL,PostgreSQL or Oracle)

**Fig. 2.** Makna architecture

Due to the fact that our system currently does not include any mechanisms for collaboratively constructing ontologies, we differentiate between two types of semantic data: the ontology data and the instance data. This distinction is similar to the common terminology in Description Logics (or OWL) and corresponds to different ways of creating and manipulating the data within the system. Ontologies are expected to be imported to the Wiki instance by administrators. These can revise or extend them using external tools (such as conventional ontology editors). The instance data is the sum of

– the RDF statements formulated in Wiki syntax by the users

[6] `http://www.jspWiki.org/Wiki/PageProvider`

– the statements manually added by the user (using assistants, see below)
– and the (external) instance data imported by the administrators

This separation is reflected in the restriction that the ontology data can be modified solely in the administrator interface, while the instance data can be manipulated in diverse ways in the Web interface by arbitrary users (cf. Figure 3).

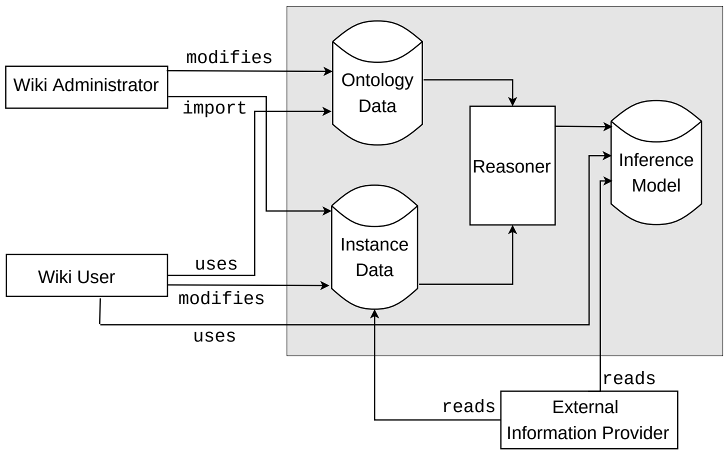


**Fig. 3.** Separation between ontology and instance data

The administration interface is responsible for the ontology- and configuration-related functionality:

– specify the ontology/ontologies used within the system
– import external RDF data
– define shortcuts for a more comfortable usage of ontological primitives
– configure inference engines and persistent storage systems.

The user interface embeds facilities for creating and using Semantic Web information on the basis of the imported ontologies:

– refer to ontological primitives for annotating Wiki content or defining link types
– formulate and execute content- and structure-based queries
– browse the Wiki contents on a content- or structure basis
– export semantically represented data as RDF or N3.

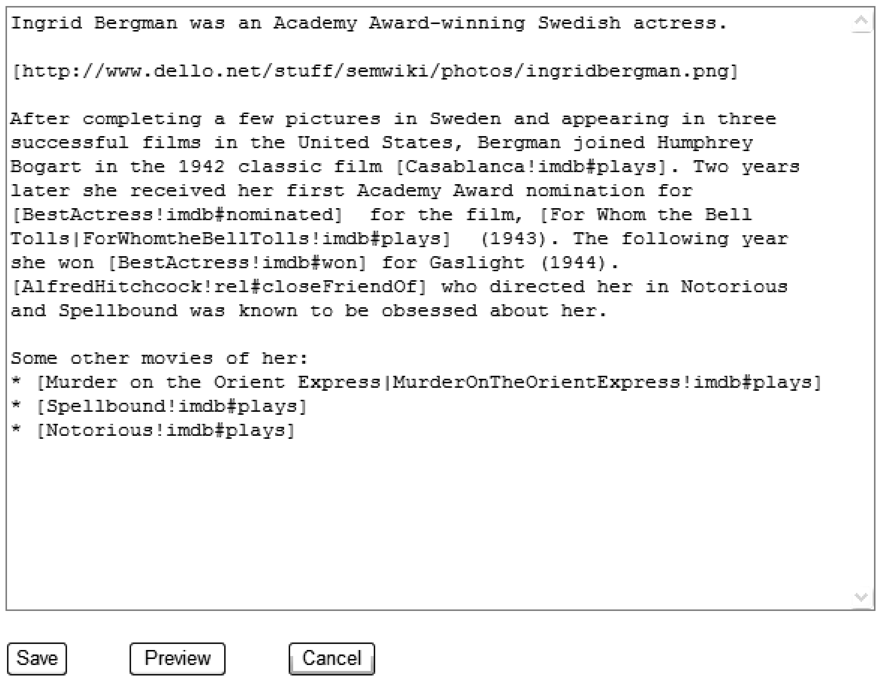
Details about the implementation of these features are discussed in the next section.

## 5 Implementation

### 5.1 Authoring semantic content

Wiki users are able to create semantic content (in form of RDF statements referencing pre-configured ontologies) in the classical Wiki manner. They are provided with an extended Wiki syntax and with assistant tools simplifying the interface to the ontologies employed. Further on, users can create, modify and delete RDF statements associated with Wiki pages.

**Extended Wiki syntax** In JSPWiki's syntax a link is represented by `[<Target>]` where `<Target>` is either an absolute URL or another page in the Wiki.



```
Ingrid Bergman was an Academy Award-winning Swedish actress.

[http://www.dello.net/stuff/semwiki/photos/ingridbergman.png]

After completing a few pictures in Sweden and appearing in three
successful films in the United States, Bergman joined Humphrey
Bogart in the 1942 classic film [Casablanca!imdb#plays]. Two years
later she received her first Academy Award nomination for
[BestActress!imdb#nominated]  for the film, [For Whom the Bell
Tolls|ForWhomtheBellTolls!imdb#plays]  (1943). The following year
she won [BestActress!imdb#won] for Gaslight (1944).
[AlfredHitchcock!rel#closeFriendOf] who directed her in Notorious
and Spellbound was known to be obsessed about her.

Some other movies of her:
* [Murder on the Orient Express|MurderOnTheOrientExpress!imdb#plays]
* [Spellbound!imdb#plays]
* [Notorious!imdb#plays]
```

**Fig. 4.** Makna syntax

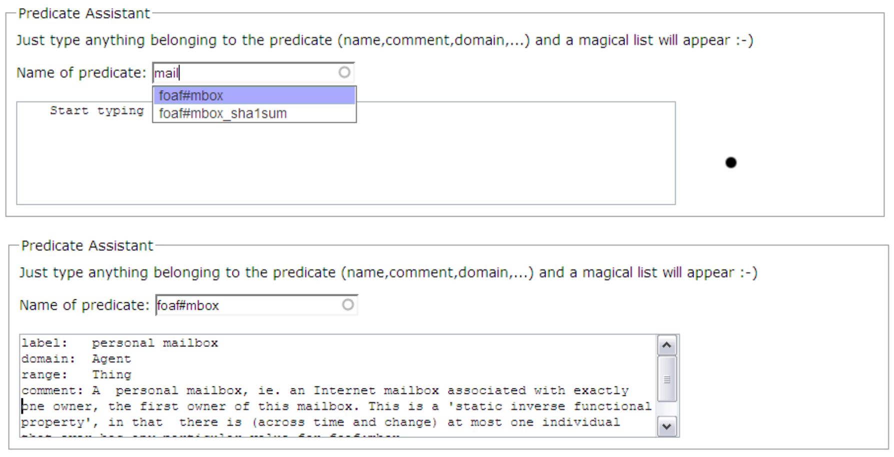
We extended this syntax to `[<Target>!<Term>]` to support semantic linking.[7] This extended link element creates a semantic statement, in which the URI of the edited page is the subject, `<Term>` is the predicate and `<Target>` is the object. Figure 4 presents the edit-page corresponding to the Wiki article on Ingrid

[7] The exclamation mark was selected because it has no other function in links in JSPWiki syntax.

Bergman, which is depicted in Figure 7 below. `<Term>` can be an URI, but more likely either a shortcut or a *namespace:predicate* combination from the resources configured in the Wiki instance. In order to support statements with literals, the range of `<Target>` was extended too. If `<Target>` is enclosed in hyphens, the system recognizes it as an literal and creates the statement accordingly.

**Interactive Assistants** In order to support the user formulating accurate semantic links in the proposed Wiki-syntax we integrated several interactive assistants based AJAX[6]. We will limit our description here to those two assistants which have been integrated into the edit-page of the Wiki: the predicate assistant and the page assistant. The former guides the user in finding a predicate in the configured ontologies. The latter assists the user in seeking for the names of other pages in the Wiki.

Figure 5 illustrates the functionality of the predicate assistant. The user enters the term *mail* and a drop down list with the matching predicates is automatically created. After selecting a predicate from the list, the text area below is updated with detailed information about the selected predicate.


Predicate Assistant

Just type anything belonging to the predicate (name,comment,domain,...) and a magical list will appear :-)

Name of predicate: mail

foaf#mbox
foaf#mbox_sha1sum

Start typing

Predicate Assistant

Just type anything belonging to the predicate (name,comment,domain,...) and a magical list will appear :-)

Name of predicate: foaf#mbox

```
label:   personal mailbox
domain:  Agent
range:   Thing
comment: A  personal mailbox, ie. an Internet mailbox associated with exactly
one owner, the first owner of this mailbox. This is a 'static inverse functional
property', in that  there is (across time and change) at most one individual
```

**Fig. 5.** Predicate assistant

As soon as the user starts typing into the input field, an asynchronous request is sent from the client-browser to the server in the background, which returns an XML-document with the matching terms. These terms are compared on a lexical basis to the natural language labels of the corresponding items (predicates and Wiki pages, respectively).

Our system also supports statements which do not appear in the Wiki pages themselves. This means that meta-statements can be expressed, and links between two non-existent pages and/or from external resources can be formulated.

The associated assistant provides a simple interface which allows users to access the content of available ontologies in order to specify subjects, predicates and objects of new statements. Figure 6 depicts this functionality: after selecting the FOAF-ontology, the assistant updates the predicate form with the corresponding information (top part of Figure 6); depending on these two parameters the assistant further computes appropriate object resources (bottom part of the same figure)

**Consistency of the semantic model** The consistency of the semantic model can be guaranteed by our implementation. To achieve this we implemented two functions. If a statement is submitted by the user with the syntax-external assistant (cf. Figure 6) we check its validity immediately; in case the statement is found to be invalid according to the semantic model it is rejected and the user gets notified. The same happens if the statement is found to cause inconsistencies in the model. The other function is the verification of statements which are formulated in the Wiki syntax. After the edited page is submitted the system extracts all semantic statements and checks if they are consistent with the semantic model. If statements are found which cause inconsistencies, the user is returned to the edit page, gets informed about the details of the problem detected and is asked to correct his input. By doing so we assure that the semantic model is always consistent, as there is no possibility to add statements that cause inconsistencies.

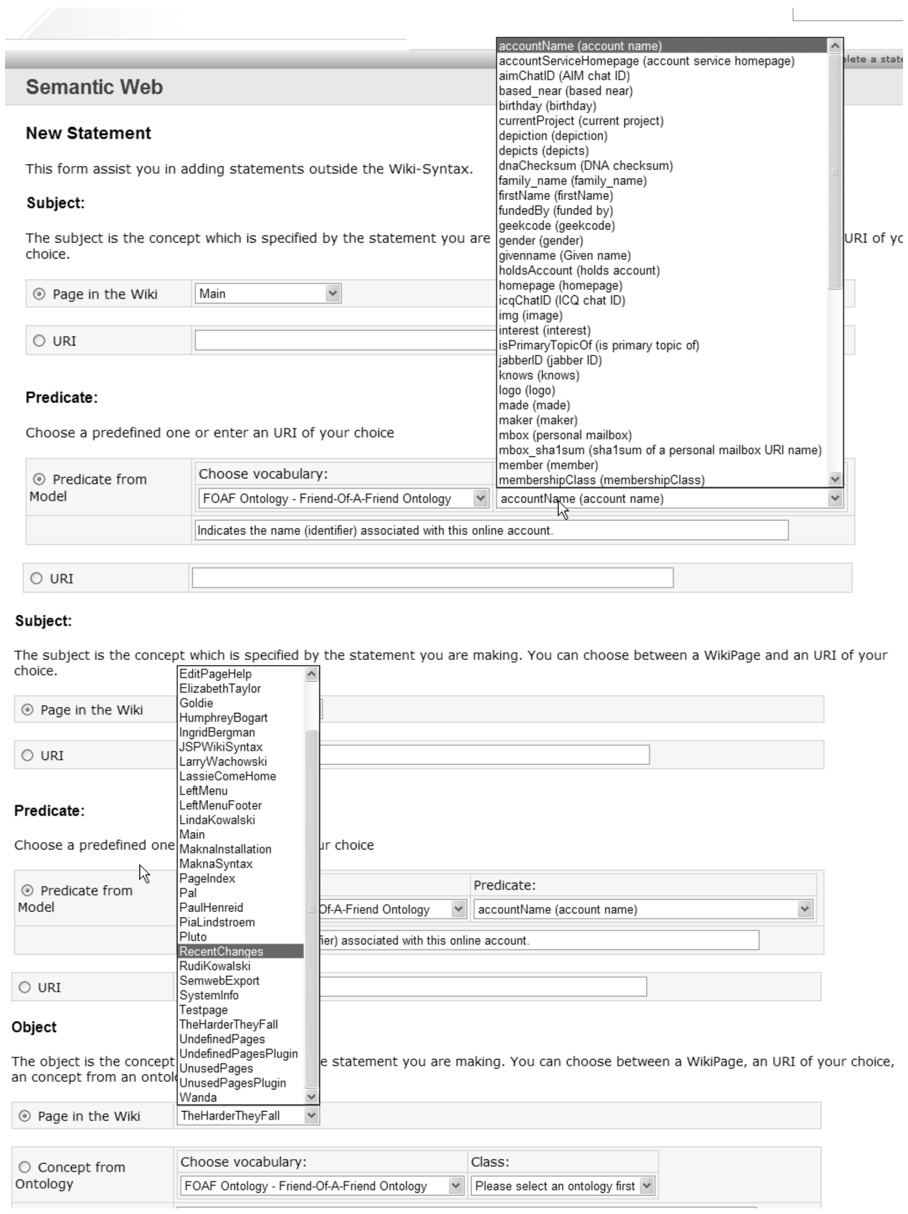
This behavior is currently being refined in order to support application scenarios with loose consistency requirements. In conjunction to the extension of our system towards advanced collaborative ontology engineering support (cf. Section 7) we are examining ways to ensure local consistency on personal ontologies, while the global shared ontology does not have to satisfy this (sometimes unfeasible) constraint.

### 5.2 Context-based presentation and navigation

When a call to a Wiki page is issued the Wiki engine extracts a subgraph of the semantic model which contains all statements which have the current page either as their subject or their object—no matter if they were formulated in the Wiki syntax or elsewhere.

In Figure 7 we illustrate the Wiki article about the actress Ingrid Bergman in a fictive Makna movie instance. The navigation block on the right side of the screen consists of two parts: the summary of the semantic relations of the current page (on the top) and the list of the prepared search requests for related resources (on the bottom).

The summary of the semantic relations can help the user to quickly navigate to a related topic in one step (i.e. one click). The customized links to searches for related pages are created through the following scheme: for each property of a page a search link is provided to find other resources with the same property. This is true for incoming links as well, which means that is is possible to navigate


Semantic Web

New Statement

This form assist you in adding statements outside the Wiki-Syntax.

Subject:

The subject is the concept which is specified by the statement you are making. You can choose between a WikiPage and an URI of your choice.

(•) Page in the Wiki: Main

( ) URI

Predicate:

Choose a predefined one or enter an URI of your choice

(•) Predicate from Model — Choose vocabulary: FOAF Ontology - Friend-Of-A-Friend Ontology — accountName (account name)

Indicates the name (identifier) associated with this online account.

( ) URI

accountName (account name)
accountServiceHomepage (account service homepage)
aimChatID (AIM chat ID)
based_near (based near)
birthday (birthday)
currentProject (current project)
depiction (depiction)
depicts (depicts)
dnaChecksum (DNA checksum)
family_name (family_name)
firstName (firstName)
fundedBy (funded by)
geekcode (geekcode)
gender (gender)
givenname (Given name)
holdsAccount (holds account)
homepage (homepage)
icqChatID (ICQ chat ID)
img (image)
interest (interest)
isPrimaryTopicOf (is primary topic of)
jabberID (jabber ID)
knows (knows)
logo (logo)
made (made)
maker (maker)
mbox (personal mailbox)
mbox_sha1sum (sha1sum of a personal mailbox URI name)
member (member)
membershipClass (membershipClass)

Subject:

The subject is the concept which is specified by the statement you are making. You can choose between a WikiPage and an URI of your choice.

(•) Page in the Wiki

( ) URI

Predicate:

Choose a predefined one or enter an URI of your choice

(•) Predicate from Model — Of-A-Friend Ontology — Predicate: accountName (account name)

…fier) associated with this online account.

( ) URI

Object

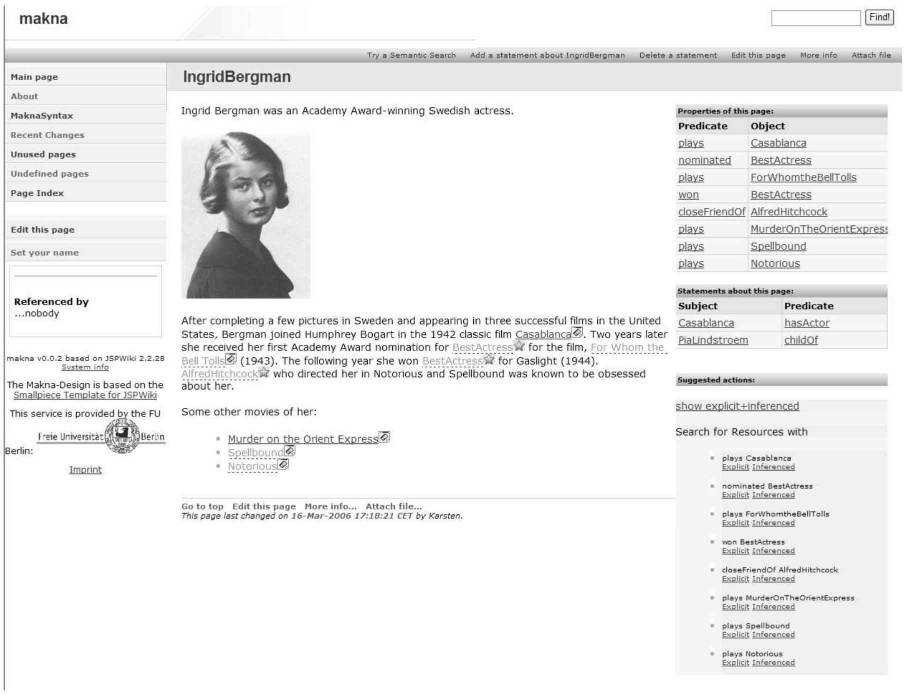
The object is the concept … the statement you are making. You can choose between a WikiPage, an URI of your choice, an concept from an ontol…

EditPageHelp
ElizabethTaylor
Goldie
HumphreyBogart
IngridBergman
JSPWikiSyntax
LarryWachowski
LassieComeHome
LeftMenu
LeftMenuFooter
LindaKowalski
Main
MaknaInstallation
MaknaSyntax
PageIndex
Pal
PaulHenreid
PiaLindstroem
Pluto
RecentChanges
RudiKowalski
SemwebExport
SystemInfo
Testpage
TheHarderTheyFall
UndefinedPages
UndefinedPagesPlugin
UnusedPages
UnusedPagesPlugin
Wanda

(•) Page in the Wiki: TheHarderTheyFall

( ) Concept from Ontology — Choose vocabulary: FOAF Ontology - Friend-Of-A-Friend Ontology — Class: Please select an ontology first

**Fig. 6.** Statement assistant



makna

Find!

Try a Semantic Search | Add a statement about IngridBergman | Delete a statement | Edit this page | More info | Attach file

Referenced by
...nobody

makna v0.0.2 based on JSPWiki 2.2.28
System info

The Makna-Design is based on the Smallpiece Template for JSPWiki

This service is provided by the FU Berlin: Freie Universität Berlin

Imprint

**IngridBergman**

Ingrid Bergman was an Academy Award-winning Swedish actress.

After completing a few pictures in Sweden and appearing in three successful films in the United States, Bergman joined Humphrey Bogart in the 1942 classic film Casablanca. Two years later she received her first Academy Award nomination for BestActress for the film, For Whom the Bell Tolls (1943). The following year she won BestActress for Gaslight (1944). AlfredHitchcock who directed her in Notorious and Spellbound was known to be obsessed about her.

Some other movies of her:

- Murder on the Orient Express
- Spellbound
- Notorious

Go to top Edit this page More info... Attach file...
*This page last changed on 16-Mar-2006 17:18:21 CET by Karsten.*

Properties of this page:

| Predicate | Object |
|---|---|
| plays | Casablanca |
| nominated | BestActress |
| plays | ForWhomtheBellTolls |
| won | BestActress |
| closeFriendOf | AlfredHitchcock |
| plays | MurderOnTheOrientExpress |
| plays | Spellbound |
| plays | Notorious |

Statements about this page:

| Subject | Predicate |
|---|---|
| Casablanca | hasActor |
| PiaLindstroem | childOf |

Suggested actions:

show explicit+inferenced

Search for Resources with

- plays Casablanca
  Explicit Inferenced
- nominated BestActress
  Explicit Inferenced
- plays ForWhomtheBellTolls
  Explicit Inferenced
- won BestActress
  Explicit Inferenced
- closeFriendOf AlfredHitchcock
  Explicit Inferenced
- plays MurderOnTheOrientExpress
  Explicit Inferenced
- plays Spellbound
  Explicit Inferenced
- plays Notorious
  Explicit Inferenced

**Fig. 7.** Makna article on Ingrid Bergman

quickly to other resources which have the same relation to the current page. Both functions work with inference (which can for convenience be switched on and off by the user), thus enabling the user to navigate the Wiki contents conducted by semantic relations.

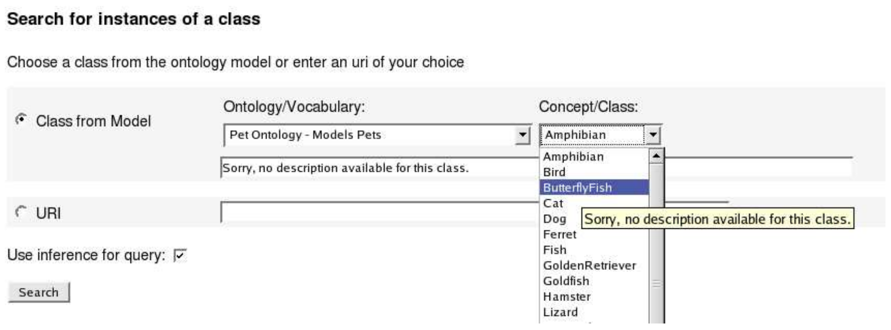
### 5.3 Content- and structure-based retrieval

Makna implements a search interface resorting to form-based search patterns, which allow users to use the ontology in formulating structure-based queries and the underlying inference engines for enabling semantic search.

We have developed several templates for comfortably formulating typical content- and structure-based query patterns. Figure 8 shows the implementation of a template returning instances of a user-defined class in a knowledge base. After choosing a vocabulary from the left drop-down list, the right list is filled up with all concepts from the chosen ontology.

## 6 Related Work

A multitude of promising approaches for combining Semantic Web and Wiki technologies are currently under development (cf., for example, [2, 15, 16, 19, 22]).


Search for instances of a class

Choose a class from the ontology model or enter an uri of your choice

| | | |
|---|---|---|
| Class from Model | Ontology/Vocabulary: Pet Ontology - Models Pets | Concept/Class: Amphibian |
| | Sorry, no description available for this class. | |
| URI | | |

Use inference for query: ✓

Search

**Fig. 8.** Form for a simple semantic search

However, while these proposals share the declared common goal of realizing a Semantic Wiki system, a closer investigation of their—planed or implemented—features evidences that they are oriented at slightly divergent application scenarios.

Approaches such as [16, 19] develop Wiki engines which support the generation of RDF data. However, they clearly distinguish between semantic and plain Wiki contents and their usage in Semantic Web context requires technical expertise on RDF. Acknowledging for this limitations more recent proposals focus on the minimal invasive usage of semantic technologies within Wiki systems[2, 15, 22]. Makna differs from these approaches from a multitude of viewpoints. In contrast to WikSAR [2] our system is oriented at non-technical users exercising collaborative knowledge engineering. This focus motivated a different design of the user interfaces. Further on, the present implementation of Makna restricts the access to *ontological knowledge* (as opposed to *instance knowledge*) to a predefined set of Wiki users. While this might be considered as a limitation of our system as compared to other approaches (such as IkeWiki [15]), we argue that implementing such functionality without adequate process-level support might have uncontrolled consequences on the operation of the overall Wiki system. This was confirmed by recent advances in the area of distributed ontology engineering[14, 18]. The Semantic MediaWiki project has come up with a different approach [10, 22]. Their development aims at turning the hugest existing traditional Wiki-based information repository—the Wikipedia encyclopedia—into a semantic Wiki. Similar to our own requirements analysis, the authors identified the need for typed annotated links and articles in Wikipedia (cf. Section 2). The current release of the Semantic MediaWiki system shares many commonalities with the functionality of Makna. In contrast to our implementation it does not support consistency checking mechanisms and the usage of multiple ontologies. In turn, it allows for an unrestricted access to both ontological and instance data, an option which we consider disputable for arbitrary settings without an adequate process support.[8] Because Makna focuses on small to mid-size Wikis we can utilize some performance critical mechanism like real-time inference, enabling features like guaranteed consistence of the model and context-based navigation based on real-time state of the semantic model. Furthermore Makna's separation of instance and ontology data—accompanied by the user and administrator interfaces (cf. Section 4)—enables predictable response times and memory usage which are the foundation for an efficient Semantic Wiki system. Finally we mention the SemperWiki approach [12], which introduces Wikis as enabling technology for personal information management. Unlike conventional Wiki and Semantic Wiki solutions—including Makna—it is targeted at single user environments.

## 7 Outlook

A feasible combination of Wiki and Semantic Web technologies should preserve the key advantages of both technologies: the simplicity of Wiki systems as regarding shared content authoring, as well as the power of Semantic Web technologies w.r.t. structuring and retrieving knowledge. Building upon an analysis of the requirements induced by a collaborative knowledge engineering scenario to Wiki implementations we have introduced our concept of a Semantic Wiki engine addressing this problem.

We are currently extending the functionality of the current release of Makna w.r.t. methodological and technological support for knowledge engineering. In particular we are investigating means to automatically extract ontological structures from existing domain-focused Wiki instances (following the approach in [13]) and are implementing a component for collaborative ontology engineering based on [14].

### Acknowledgements

This work has been partially supported by the EU Network of Excellence "KnowledgeWeb" (FP6-507482).

## References

1. Désilets A., S. Paquet, and N. G. Vinson. Are Wikis Usable? In *Proceedings of the 1st ACM Wiki Symposium WikiSym2005*, 2005.
2. D. Aumüller. Semantic Authoring and Retrieval in a Wiki. In *Demo Session at the European Semantic Web Conference ESWC2005*, 2005.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
4. M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, and H. Oinas-Kukkonen. Forth generation hypermedia: some missing links for the World Wide Web. *International Journal of Human-Computer Studies*, 47, 1997.

[8] However, this design decision might prove to be appropriate for the Wikipedia use case.

5. W. Cunningham and B. Leuf. *The Wiki Way. Quick Collaboration on the Web.* Addison-Wesley, 2001.
6. J. Garrett. Ajax: A New Approach to Web Applications (white paper). http://www.adaptivepath.com/publications/essays/archives/000385.php, February 2005.
7. T. Hemmann. How Knowledge Engineering Can Benefit from Software Engineering with Respect to Reuse: Towards Reusable Knowledge Models. In *Proceedings of the ERICM Workshop on Methods and Tools for Software Reuse*, pages 212–227, 1993.
8. M. Klein, A. Kiryakov, D. Ognyanov, and D. Fensel. Ontology Versioning and Change Detection on the Web. In *Proceedings of the 13th International Conference on Knowledge Engineering and Management EKAW02*, 2002.
9. K. Kotis, G. A. Vouros, and J. Padilla Alonso. HCOME: tool-supported methodology for collaboratively devising living ontologies. In *Proceedings of the 2nd International Workshop on Semantic Web and Databases SWDB2004*, 2004.
10. M. Kroetzsch, D. Vrandecic, and M. Voelkel. Wikipedia and the Semantic Web: The Missing Links. In *Proceedings of 1st International Wikipedia Conference Wikimania2005*, 2005.
11. C.W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(2):131–183, June 1992.
12. E. Oren. Semperwiki: a Semantic Personal Wiki. In *Proceedings of the 1st Semantic Desktop Workshop*, 2005.
13. E. Paslaru Bontas, D. Schlangen, and T. Schrader. Creating Ontologies for Content Representation – the OntoSeed Suite. In *Proceedings of the 4th International Conference on Ontologies, Databases, and Applications of Semantics ODBASE2005*, 2005.
14. H. S. Pinto, S. Staab, and C. Tempich. DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of oNTologies. In *Proceedings of the European Conference of Artificial Intelligence ECAI2004*, pages 393–397, 2004.
15. S. Schaffert. IkeWiki - A Semantic Wiki for Collaborative Knowledge Management. Technical report, Salzburg Research, 2006.
16. A. Souzis. Building a Semantic Wiki. *IEEE Intelligent Systems*, 20:87–91, 2005.
17. L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-Driven Ontology Evolution Management. In *Proceedings of the 13th European Conference on Knowledge Engineering and Management EKAW02*, 2002.
18. Y. Sure, C. Tempich, and D. Vrandecic. Ontology Engineering Methodologies. In Davies, J. and Studer, R. and Warren, P., editor, *Semantic Web Technologies: Trends and Research in Ontology-based Systems.* Wiley, 2006.
19. R. Tazzoli and P. Castagna et al. Towards a Semantic Wiki Wiki Web. In *Poster Session at the International Semantic Web Conference ISWC2004*, 2004.
20. P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *Proceedings of the AAAI Spring Symposium on Ontological Engineering AAAI97*, 1997.
21. F. Vitali and M. Bieber. Hypermedia on the Web: What Will It Take? *ACM Computing Surveys*, 31(4), 1999.
22. M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic Wikipedia. In *Proceedings of the World Wide Web Conference WWW2006 (to appear)*, 2006.

# Annotation and Navigation in Semantic Wikis[⋆]

Eyal Oren[1], Renaud Delbru[1], Knud Möller[1], Max Völkel[2], and Siegfried Handschuh[1]

[1] DERI Galway, Ireland
`firstname.lastname@deri.org`
[2] Forschungzentrum Informatik, Karlsruhe, Germany
`voelkel@fzi.de`

**Abstract.** Semantic Wikis allow users to semantically annotate their Wiki content. The particular annotations can differ in expressive power, simplicity, and meaning. We present an elaborate conceptual model for semantic annotations, introduce a unique and rich Wiki syntax for these annotations, and discuss how to best formally represent the augmented Wiki content. We improve existing navigation techniques to automatically construct faceted browsing for semistructured data. By utilising the Wiki annotations we provide greatly enhanced information retrieval. Further we report on our ongoing development of these techniques in our prototype SemperWiki.

## 1 Introduction

Wikis are collaborative hypertext authoring environments. Wikis allow people to collaboratively collect, describe, and author information. Since most information in ordinary Wikis consists of natural-language texts, structured access and information reuse are practically not possible [13].

Semantic Wikis allow users to make formal descriptions of resources by annotating the pages that represent those resources. Where a regular Wiki enables users to describe resources in natural language, a Semantic Wiki enables users to additionally describe resources in a formal language. By adding metadata to ordinary Wiki content, users get added benefits such as improved retrieval, information exchange, and knowledge reuse.

An ordinary Wiki should offer functionality[3] such as access control, binary data management, version management, notification, and data export. In our opinion, a Semantic Wiki should specifically address three additional questions:

1. how to annotate content?
2. how to formally represent content?
3. how to navigate content?

[⋆] This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694 and by the European Commission under the Nepomuk project FP6-027705.

[3] `http://en.wikipedia.org/wiki/Wiki`

Recently several Semantic Wikis have been developed, such as Platypus [22], WikSAR [2], Semantic MediaWiki [23] and IkeWiki [20]. These Wikis answer these questions in a rather limited way: (a) they allow only simple annotations of the current Wiki page; (b) they do not formally separate the page and the concept that it describes; and (c) they do not fully exploit the semantic annotations for improved navigation.

In this paper we specifically address these three questions in a broader way: in Sect. 2 we analyse Wiki annotations from a conceptual level, discuss representation mechanisms, and current annotation support in Semantic Wikis. In Sect. 3 we offer an improved navigational model based on semantic annotation; the navigation model is similar to e.g. Longwell[4] for faceted browsing of semistructured data, but works, in contrast to existing approaches, for arbitrary datasets with arbitrary structure. We report on our prototype implementation SemperWiki [12] in Sect. 4; the implementation has been updated to include these new ideas.

## 2 Annotations

In the following section we discuss our first question: how to annotate Wiki content?

Let us first analyse what an annotation is. We annotate data all the time: when we read a paragraph, and mark "great!" in the margin, that is an annotation; when our text editor underlines a misspelled word, that is also an annotation. Annotations add some information to some other information; to annotate means "to make notes or comments" [16].

Another way to view annotations is metaphorically: URIs[5] are the "atoms" of the Semantic Web and semantic annotations are the "molecules". The Semantic Web is about shared terminology, achieved through consistent use of URIs. Annotations create a relationship between URIs and build up a network of data.

### 2.1 Conceptual model

We now explore the conceptual model behind annotation in more depth. The term "annotation" can denote both the process of annotating and the result of that process [9]. Where we say "annotation" we mean the result. An annotation attaches some data to some other data. An annotation establishes, within some context, a (typed) relation between the annotated data and the annotating data.

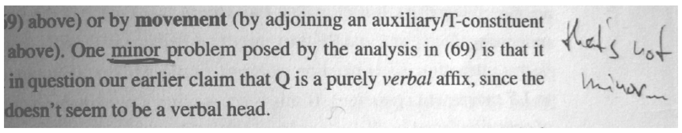
Investigating the nature of annotation further, we can model it as a quadruple:

**Definition 1 (Annotation).** *An annotation A is a tuple* $(a_s, a_p, a_o, a_c)$*, where* $a_s$ *is the subject of the annotation, the annotated data,* $a_o$ *is the object of the annotation, the annotating data,* $a_p$ *is the predicate, the annotation relation, that*

[4] `http://simile.mit.edu/longwell/`
[5] `http://www.w3.org/Addressing/`

*defines the type of relationship between $a_s$ and $a_o$, and $a_c$ is the context in which the annotation is made.*

*Example 1 (Informal annotation).*



9) above) or by **movement** (by adjoining an auxiliary/T-constituent above). One minor problem posed by the analysis in (69) is that it in question our earlier claim that Q is a purely *verbal* affix, since the doesn't seem to be a verbal head.

that's not minor

The *annotation subject* can be formal or informal. For example, when we put a note in the margin of a paragraph, the informal convention is that the note applies to the paragraph, but that pointer is not formally defined. If we however use a formal pointer such as a URI[6] to point to the paragraph then the subject is formally specified.

The *annotation predicate* can be formal or informal. For example, when we put a note in the margin, the relation is not formally defined, but we may informally derive from the context that that the note is a comment, a change-request, an approval or disapproval, etc. If we use a formal pointer to an ontological term that indicates the relation (e.g. `dc:comment`) then the predicate is formally defined.

The *annotation object* can be formal or informal. If an object is formal we can distinguish different levels of formality: textual, structural, or ontological. For example, then string "This is great!" is a textual object. A budget calculation table in the margin of a project proposal is a structural object. And an annotation object that is not only explicitly structured but also uses ontological terms[7] is an ontological object.

The *annotation context* can be formal or informal. Context can could indicate when the annotation was made and by whom (provenance), or within what scope the annotation is deemed valid, for example in a temporal scope (it is only valid in 2006) or in a spatial scope (it is only valid in Western Europe). Usually context is given informally and implicitly. If we use a formal pointer such as a URI then the context is formally defined.

Combining the levels of annotation subject, predicate, and object, we can distinguish three layers in annotations: i) informal annotations, ii) formal annotations (that have formally defined constituents and are thus machine-readable),

[6] One can use XPointer to point to a paragraph in a document and XPointer can be used as a URI, as discussed in `http://www.w3.org/TR/xptr-framework/#escaping`.

[7] Ontological means that the terminology has a commonly understood meaning that corresponds to an shared conceptualisation called ontology [8]. Whether a term is *ontological* is a social matter and not a technical or formal matter. It is sometimes mistakenly understood that using a formal ontology language makes terms ontological. An ontology however denotes a shared (social) understanding; the ontology language can be used to formally capture that understanding, but does not preclude reaching an understanding in the first place.

and iii) semantic annotations (that have formally defined constituents and use only ontological terms). We have given some simple examples for each kind of annotation in Examples 1 (a handwritten margin annotation in a book), 2 (formally expressed in N3[8]) and 3 (formally expressed and using ontological terms), respectively. All three examples are here given without any explicit context.

**Definition 2 (Formal annotation).** *A formal annotation $A_f$ is an annotation $A$, where the subject $a_s$ is a URI, the predicate $a_p$ is a URI, the object $a_o$ is a URI or a formal literal, and the context $a_c$ is a URI.*

*Example 2 (Formal annotation).*

```
<http://papers.org/minimalism#minor>
 <http://localhost/schema#disagree>
 "that's not minor!".
```

**Definition 3 (Semantic annotation).** *A semantic annotation $A_s$ is a formal annotation $A_f$, where the predicate $a_p$ and the context $a_c$ is an ontological term, and the object $a_o$ conforms[9] to an ontological definition of $a_p$.*

*Example 3 (Semantic annotation).*

```
<http://papers.org/minimalism#minor>
  ibis:con
   [ rdf:type ibis:Argument;
     rdf:label "that's not minor!" ].
```

## 2.2 Annotations in Wikis

We can, similarly to [18], distinguish three levels of annotations in a Semantic Wiki:

**Layout** Annotations that describe textual formatting without additional structural information, such as **bold** or *italic* words[10].

**Structure** Annotations that describe the structure of a page or of a set of pages, such as hyperlinks (inter-page structure), headings, subheadings, and paragraphs (internal page structure), and itemised and numbered lists.

**Semantics** Annotations that relate pages or page elements to arbitrary resources through typed ontological relations, such as categorising a page in a taxonomy, specifying the friends of a described person, or the books of a described author.

[8] `http://www.w3.org/DesignIssues/Notation3.html`

[9] The notion of "conformance" is rather weak in some ontology languages (such as RDFS or OWL) since these are not constraint-based languages (as opposed to e.g. database schemas). However, we use the notion of conformance to differ between "good" usage of textual objects, for example to indicate the name of a person, and "bad" usage of textual objects, for example to indicate the friends of a person.

[10] These annotations could formally be considered semantical, because they have an explicit and shared meaning, which is used by the rendering engine.

Annotations in a regular Wiki are limited to layout and structural annotations. Semantic annotations are unique to Semantic Wikis, and are the further focus of this section.

We now present one possible annotation syntax for semantic annotations, namely the one used in SemperWiki [12]. To simplify the annotations, we only consider annotations that have the page on which they appear as subject. The annotation subject is thus implicitly defined. We also limit ourselves, for simplicity, to annotations with an implicit context. The annotations are then restricted to defining the *predicate* and *object*, which is done by simply stating the two on a separate line.

The example page shown in Fig. 1 describes the World Wide Web Consortium. The page includes some English text, and some annotations which state (using the Wordnet and Semantic Web Research Community ontologies) that the W3C is an organisation lead by Tim Berners-Lee. The syntax includes referencing using namespace abbreviations, internal Wiki pages, and full URIs; see [12] for more information.

**W3C**

```
The World Wide Web Consortium (W3C) develops interoperable
technologies (specifications, guidelines, software, and
tools) to lead the Web to its full potential.

rdf:type wordnet:Organization
swrc:head http://www.w3.org/People/Berners-Lee/card#i

dc:date "2006/01/01"
```

Fig. 1: Simple Wiki page about the W3 Consortium

### 2.3 Representation

Having defined annotations in Wikis, we now answer the second question: how to formally represent Wiki content?

RDF[11] is a straightforward way to represent these annotations formally, since it has exactly the same model as our annotations. We can either use standard RDF to represent annotations without context, or RDF quads (which is a common RDF extension) for annotations with context.

RDF does pose some constraints on the constituents of triples: the subject must be a URI or a blank node (not a literal), and the predicate must be a URI (not a literal or blank node). If we follow these restrictions in our annotations, RDF offers a good representation model.

[11] `http://www.w3.org/RDF/`

We represent pages and their annotation in RDF as follows: each page is an RDF resource, and each annotation a property of that resource. We can represent not only the semantic annotations in RDF but the whole Wiki content. The (natural language) Wiki content is captured through the predicate `semper:content`, the outgoing links to other pages through the predicate `semper:links`. Figure 2 shows the RDF graph that represents the page in Fig. 1.

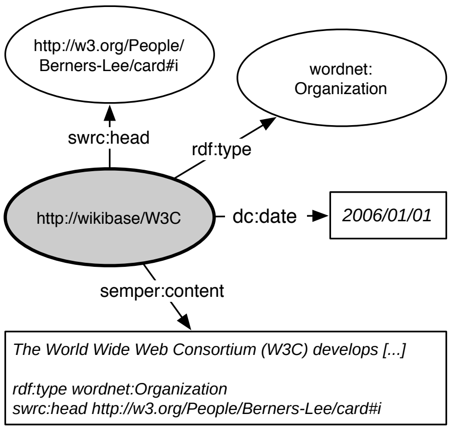

http://w3.org/People/Berners-Lee/card#i
wordnet: Organization
swrc:head
rdf:type
http://wikibase/W3C
dc:date
2006/01/01
semper:content
The World Wide Web Consortium (W3C) develops [...]
rdf:type wordnet:Organization
swrc:head http://w3.org/People/Berners-Lee/card#i

Fig. 2: RDF graph for the W3C page in Fig. 1

**Problem: documents vs. concepts** Because annotations can describe concepts (the W3 consortium) and web documents (the page about the W3 Consortium), the question arises which URI to use as the annotation subject.

For example, the Wiki page in Fig. 1 also contains the statement that it was created on January 1, 2006. But does this statement say that the *document* was created in 2006 or that the subject *concept* of the document, i.e. the W3C, was created in 2006? We may derive with some background information that we mean the first, but we actually need a way to say both: we sometimes want to make statements about a concept and sometimes about the document describing that concept.

This issue (often referred to as the "URI crisis") is well-known from early discussions on Web architecture, and has gained renewed interest in the Semantic Web community. The problem is that it is unclear what a URI denotes (at least, it is unclear for URIs that are URLs, but the discussion focuses primarily on http URIs which are indeed URLs). A URL can denote a name, an abstract concept, a web location, or a document [5]. The root of the problem is that the same URI can be used to identify a subject directly (web document) or indirectly (concept that is subject of document) [15].

Hawke [10] suggests[12] to disambiguate the concept and the document syntactically by using the `#` symbol: `http://google.com/` would denote the web document and `http://google.com/#` would denote the concept. The solution is not ideal [15] since the hash symbol is a legal URI character and can be used to denote a document fragment, while referring to document fragments with URI fragment identifiers is crucial for fine-grained document annotation[13].

**Solution: locators vs. names** As Pepper remarks, "using a locator for something that does not have a location is asking for trouble" [15]. The obvious solution is to not use a locator (URL) but a non-addressable identifier[14] (URN) for non-locatable things such as concepts.

Unfortunately, using a URN to identify concepts violates the fundamental Web principle that a URI should point to a location with useful information about the thing it identifies [4]. However, that could be remedied by using a syntactical convention (*mirror-URIs*) to relate the document URL to the concept URN, such as prefixing the URL with the `urn:` protocol handler.

To complete this solution, we need to extend our Wiki syntax in two ways to include a way:

1. to distinguish annotations about a document (Wiki page) from annotations about the concept, which we do by prefixing the annotation with the `!` symbol.
2. to relate a page to the concept it describes (in case the page describes a concept in a different naming authority, e.g. a page on `http://wikibase/W3C` that describes `urn://w3.org`), which we do with `semper:about`.

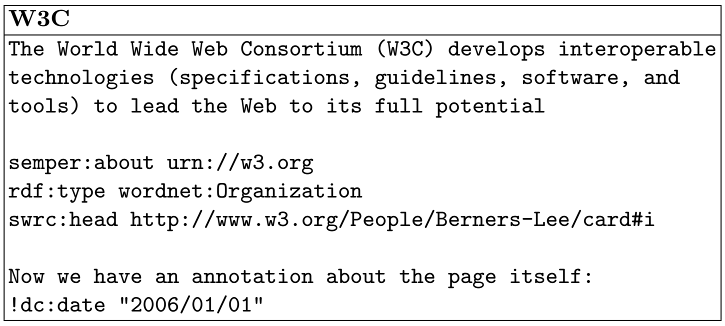
Figure 3a shows how these extensions are used to now correctly state that the W3C (identified by `urn://w3.org`) is an organisation headed by Tim Berners-Lee, and that this page (identified by `http://wikibase/W3C`) was created on January 1st, 2006, and Fig. 3b shows the corresponding RDF graph.

### 2.4 Annotation in current Semantic Wikis

Having answered the first two questions (how to annotate and how to represent Wiki content), we now characterise the annotation and representation in several existing Semantic Wikis.

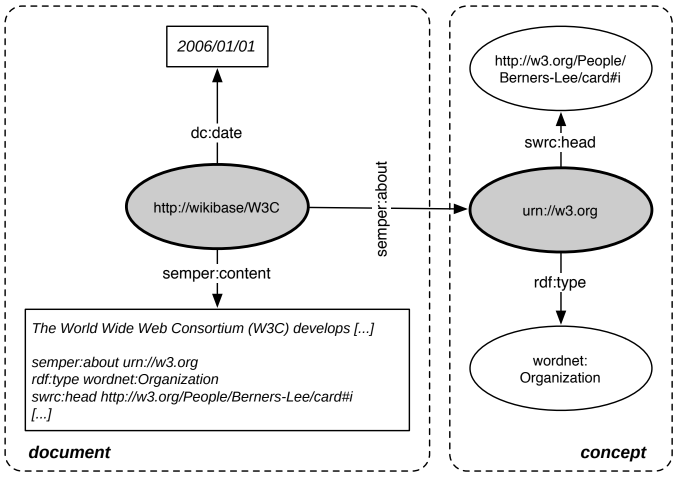
Annotations in Semantic Wikis are formal and possibly semantic, i.e. they are formally defined, and possibly use ontological terms. We have selected several dimensions to classify annotations in Semantic Wikis from the literature (we again focus on the annotation result, not the annotation process). We have added one new dimension to capture the important notion of annotation *context*:

[12] The proposal is a bit more intricate, but for our purposes this explanation suffices.
[13] see e.g. `http://w3.org/TR/annotor` [9].
[14] Clarification on the relation between URIs, URLs and URNs can be found at `http://www.w3.org/TR/uri-clarification/`.


**W3C**

```
The World Wide Web Consortium (W3C) develops interoperable
technologies (specifications, guidelines, software, and
tools) to lead the Web to its full potential

semper:about urn://w3.org
rdf:type wordnet:Organization
swrc:head http://www.w3.org/People/Berners-Lee/card#i

Now we have an annotation about the page itself:
!dc:date "2006/01/01"
```

(a) example page


2006/01/01
dc:date
http://wikibase/W3C
semper:about
semper:content
The World Wide Web Consortium (W3C) develops [...]
semper:about urn://w3.org
rdf:type wordnet:Organization
swrc:head http://w3.org/People/Berners-Lee/card#i
[...]
document
http://w3.org/People/Berners-Lee/card#i
swrc:head
urn://w3.org
rdf:type
wordnet: Organization
concept

(b) RDF representation

Fig. 3: RDF representation of an example page

**Subject attribution** (also called "scope" [19]) Indicates the subject of the annotation: is the subject of the annotation the same as the page on which it appears or an arbitrary page? In a Wiki, the possible attributions are: the page on which an annotation appears, an arbitrary page, or an anonymous resource.

**Subject granularity** (also called "lexical span" [18]) Indicates the granularity of the annotation subject: e.g. is the annotation about a document, a section inside a document, a sentence, or a word?

**Representation distinction** (also called "instance identification vs. reference" [3]) Indicates whether the Wiki distinguishes annotations about the Wiki page itself from annotations of the concept described on the page?

**Terminology reuse** (also called "interoperability" [19]) Indicates whether an annotation is self-confined with its own terminology, or whether an annotation uses terms from existing ontologies, and are thus interoperable and understandable for others.

**Object type** (also called "annotation form" [7]) Indicates the type of annotation object: is it a literal or textual object, a structural object (including a hyperlink to another page), or an ontological object?

**Context** Indicates the context of the annotation: when was it made, by whom (provenance), and within what scope: the annotation could for example be temporally scoped (it is only valid in 2006) or spatially scoped (it is only valid in Western Europe).

These dimensions can indicate the level of annotation in current Semantic Wiki approaches. We do not provide an exhaustive evaluation, but evaluate WikSAR [2], Semantic MediaWiki [23], IkeWiki [20] and SemperWiki [12] as the most prominent systems under ongoing development.

| **dimension** | **WikSAR** | **Sem. MediaWiki** | **IkeWiki** | **SemperWiki** |
|---|---|---|---|---|
| attribution | current | current | current | current, any URI |
| granularity | page | page | page | page, any fragment |
| repr. distinction | no | no | yes | yes |
| terminology reuse | no | no | yes | yes |
| object type | literal, page | literal, page | literal, page | literal, page, URI |
| context | no | no | no | no |

Table 1: Annotations in current Semantic Wikis

**Subject attribution** Most existing Wikis only allow statements about the current page. The subject of an annotation is never explicitly stated, but always implicitly assumed to be the page on which the statement appears. In SemperWiki the user can explicitly state the subject of the annotations, because we separate the page and the thing it describes (as explained in Sect. 2.3), and annotations can thus be attributed to arbitrary URIs.

**Subject granularity** Most existing Wikis only allow annotation of complete pages, not of subsections or arbitrary parts of text, for the same reason (implicitly) as mentioned above.
Since SemperWiki allows users to attribute annotations to arbitrary URIs one could annotate a document fragment as follows: create a Wiki page, point it to the document fragment using an XPointer URI, and annotate the page.

**Representation distinction** Of the discussed Wikis only SemperWiki clearly separates the page from the concept that it describes, and offers a syntax that distinguishes annotations of the page from annotations of the concept. IkeWiki also separates pages from the concepts that they describe (a concept can be represented on multiple pages), but does not, as far as we know, offer a syntax to manually express this distinction.

**Terminology reuse** IkeWiki and SemperWiki allow existing terminology to be reused in annotations (through namespace definitions or full URIs), the rest can only create annotations using internal Wiki pages and can thus not make use of existing terminology.

**Object type** All discussed Wikis allow an object to be a literal or an internal Wiki page. Of the discussed Wikis, only SemperWiki allows the object of an annotation to be an arbitrary URI. No Semantic Wiki allows unnamed resources (blank nodes) as objects.

**Context** Is ignored in all existing Wikis.

Summarising, we have developed a conceptual model for annotations in general, and for semantic annotations in the context of Semantic Wikis specifically. Given this model we have seen that current Semantic Wikis offer only limited annotation possibilities (which is not necessarily wrong, but has now been recognised explicitly), and do not clearly separate the page from the concept that it describes. We have shown how SemperWiki addresses these limitations.

## 3 Navigation

Having answered the first two questions, we now investigate the third question: how to navigate Wiki content?

When navigating an ordinary Wiki, all content is considered either a hyperlink or some natural language text. The hyperlinks between pages can be followed, and the full-text can be searched by keyword. But if users can not exactly formulate their information need, an exploration technique is necessary that helps users to discover data [11].

In our opinion, navigating a Wiki has two phases: looking *for* a page, and looking *at* a page. In an ordinary Wiki, exploration in both phases is limited to predefined hyperlinks. In Semantic Wikis, the semantic annotations structure the Wiki content, and we can use that structure to offer improved exploration through a technique called faceted browsing [24].

Existing approaches for faceted browsing rely on manually constructing the facets for a fixed data structure. But since Wiki content can form arbitrary and fluent structures (because users can add arbitrary annotations to pages), we need to adjust faceted browsing to arbitrary data structures.

In this section, we present our approach to automatically construct facets for an arbitrary semi-structured dataset, independent of its structure.

### 3.1 Background

Faceted browsing is a superior exploration technique for large structured datasets [24,21,6] based on the theory of facet analysis [17].

In faceted browsing, the information space is partitioned using orthogonal conceptual dimensions of the data (these dimensions are called facets). Each facet has multiple restriction values; users select a restriction value to constrain relevant items in the information space.

In the Semantic Wiki, a facet corresponds to an annotation predicate $a_p$ and a restriction value corresponds to an annotation object $a_o$. The annotation subject is the result (or purpose) of the faceted browsing: faceted browsing is a search process that takes the predicate and object values as input and returns possible matching the subject.

For example, a collection of art works can consist of facets (predicates) such as type of work, time periods, artist names and geographical locations. Users can select a certain restriction value (object) such as the 20th century to constrain the visible collection to only some art works. Multiple constraints are applied conjunctively.

Existing approaches [24,11] cannot navigate arbitrary datasets: they are limited to manually defined facets over predefined data structures. A technique for automatic classification of new data under existing facets has been developed [6], but requires a predefined training set of data and facets, and only works for textual data. A technique for automatic facet construction based on lexical dispersion has been developed [1], but is also limited to textual data.

### 3.2 Automatic facet extraction

We combine several existing techniques to offer faceted browsing for arbitrarily structured data. Setting up faceted browsing for a specific dataset involves two steps: i) selecting proper facets and ii) partitioning each facet into a number of restriction values.

In most existing faceted browsers, both steps are done manually: an administrator examines the dataset (e.g. a museum collection), selects useful facets (e.g. time period, artist name, location), and partitions each facet into useful restriction values: e.g. the time facet would be divided in 20 centuries, the artist facet into 26 starting letters, and the location (hierarchically) into continent and then countries.

We focus on automation of the first step: selecting proper facets.

### 3.3 Facet selection

A facet should only represent one important characteristic of the classified entity [17]. This entity corresponds to our notion of RDF resource. In RDF, each resource is defined by one or more predicates; these predicates could be considered as entity characteristics. Our goal is to find, among all available predicates, those that best represent the dataset.

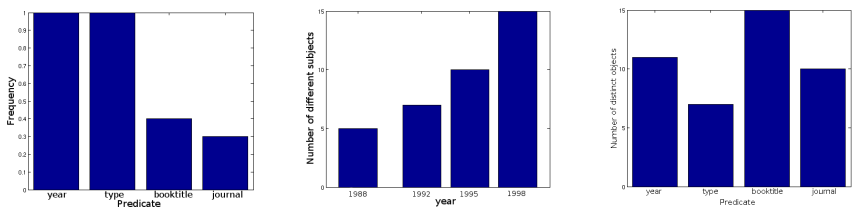
**Frequency** A good predicate has a high occurrence frequency inside the collection. The more distinct resources a predicate covers, the more useful it is in dividing the information space [6]

**Distinguishing power** A good predicate has a uniform value distribution (its distinguishing power is high). A division in which the information is distributed uniformly across all partitions enables the fastest navigation to an item of interest.

**Object values** A good predicate has a limited number of different object values (between 2 and 20). If there are too many different objects to choose from, then the options are difficult to display and may disturb the user.

**Intuition** A good predicate reflects the scope of the information space and is intuitive for the user. For example, a user who only knows the author of some book will try to find it by using the facet "author". Conversely, a user who only knows the title of a book will try to find it using the "title".

We define three metrics (for the first three properties) that rank the appropriateness of each predicate; we exclude the mathematical treatment for brevity. Fig. 4 shows these metrics for a sample (CiteSeer) dataset. We cannot define a metric for intuition, since we cannot properly define intuition.



(a) Predicate frequency (b) Distinguishing power (c) Object values

Fig. 4: Metrics in sample data

**Frequency** To measure the frequency of a predicate, we use a simple function based on the number of distinct resources that have the predicate. For example, in Fig. 4a we see that *year* and *type* occur frequently in the sample data.

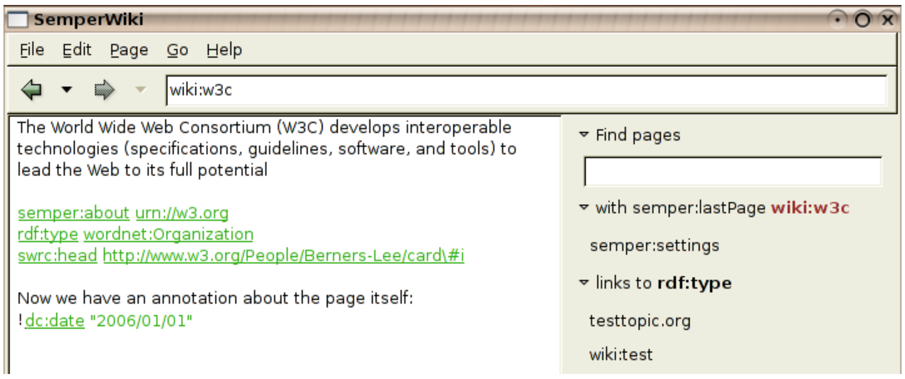
**Distinguishing power** To measure the distinguishing power of a predicate we use a simple function based on the number of distinct subjects having the same object. If each object has the same number of distinct subjects, the score of the predicate is highest. For example, in Fig. 4b we see that the predicate *year* is not very balanced: there are more publications in later years.

**Object values** For displaying and usability purposes (the user should be able to have an overview of options and decide on a restriction value), the number of different object values should be approximately between [2, 20]. For example, in Fig. 4c we see that the predicate *booktitle* has many different object values, and the predicate *type* only a few (so the latter one would be more usable).

## 4 Implementation

This section presents our prototype implementations of the previous ideas.

Our open-source prototype SemperWiki[15] [12] was initially developed as personal Wiki for knowledge management, and therefore designed as a desktop application. The original version of SemperWiki, shown in Fig. 5, is implemented in Ruby[16], using the GTK[17] graphical toolkit.



The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential

semper:about urn://w3.org
rdf:type wordnet:Organization
swrc:head http://www.w3.org/People/Berners-Lee/card\#i

Now we have an annotation about the page itself:
!dc:date "2006/01/01"

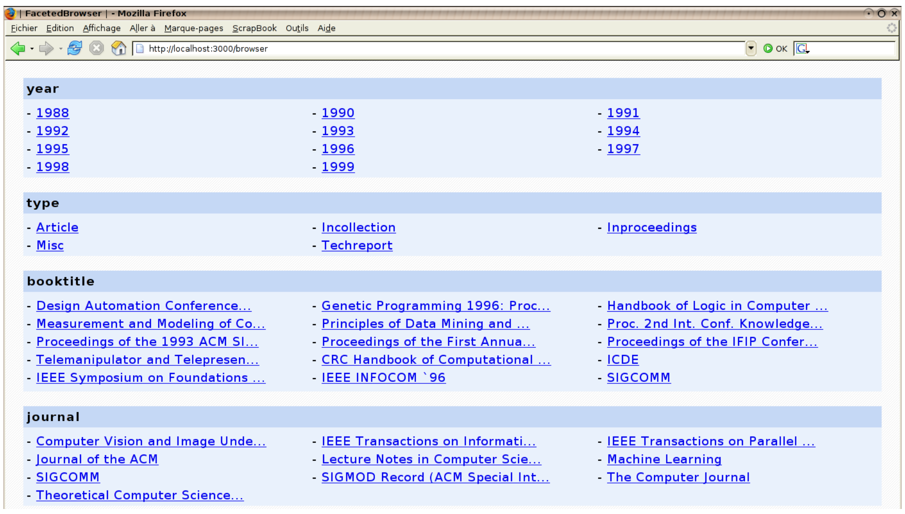
Fig. 5: SemperWiki prototype

We are currently porting SemperWiki to a Web architecture to make it cross-platform accessible, using ActiveRDF [14] and Ruby on Rails[18]. The new version of SemperWiki contains all the annotation functionality described in Sect. 2, and clearly distinguishes between documents and concepts, as discussed in Sect. 2.3.

[15] `http://semperwiki.org`
[16] `http://ruby-lang.org/`.
[17] `http://gtk.org`.
[18] `http://rubyonrails.org`

Secondly, we have built a prototype that implements the automatic selection of facets. The resulting faceted browsing interface is shown in Fig. 6; please note that this interface is automatically generated for arbitrary data. In this dataset, year, type, booktitle and journal are the facets (selected from the predicates), and 1988, 1992, etc. are the facet values (annotation objects without clustering). The prototype is implemented in Ruby and ActiveRDF, and works on arbitrary RDF data sources through the generic RDF API of ActiveRDF.

We have not yet done a comprehensive assessment, but an initial evaluation[19] looks promising: the metrics automatically select the most important predicates (such as year, type and author) as the most important facets.


| FacetedBrowser | - Mozilla Firefox
Fichier Edition Affichage Aller à Marque-pages ScrapBook Outils Aide
http://localhost:3000/browser
OK

**year**

| | | |
|---|---|---|
| - 1988 | - 1990 | - 1991 |
| - 1992 | - 1993 | - 1994 |
| - 1995 | - 1996 | - 1997 |
| - 1998 | - 1999 | |

**type**

| | | |
|---|---|---|
| - Article | - Incollection | - Inproceedings |
| - Misc | - Techreport | |

**booktitle**

| | | |
|---|---|---|
| - Design Automation Conference... | - Genetic Programming 1996: Proc... | - Handbook of Logic in Computer ... |
| - Measurement and Modeling of Co... | - Principles of Data Mining and ... | - Proc. 2nd Int. Conf. Knowledge... |
| - Proceedings of the 1993 ACM SI... | - Proceedings of the First Annua... | - Proceedings of the IFIP Confer... |
| - Telemanipulator and Telepresen... | - CRC Handbook of Computational ... | - ICDE |
| - IEEE Symposium on Foundations ... | - IEEE INFOCOM `96 | - SIGCOMM |

**journal**

| | | |
|---|---|---|
| - Computer Vision and Image Unde... | - IEEE Transactions on Informati... | - IEEE Transactions on Parallel ... |
| - Journal of the ACM | - Lecture Notes in Computer Scie... | - Machine Learning |
| - SIGCOMM | - SIGMOD Record (ACM Special Int... | - The Computer Journal |
| - Theoretical Computer Science... | | |

Fig. 6: Faceted browsing prototype

## 5 Discussion

The results of our work allows us to give good answers to the three initial research questions of this paper. We are satisfied with this overall results but we will also have in the following a short discussion about possible unsettled points.

Our approach for annotation in the Semantic Wiki ignores the context of annotations. Actually, to our knowledge, all annotation approaches ignore the notion of context. More research is needed on identifying and on modelling context of annotations.

[19] On a sample CiteSeer dataset from `http://www.csd.abdn.ac.uk/~ggrimnes/swdataset.php`.

Secondly, when annotating Wiki concepts we might encounter a naming ambiguity if two people use different URNs for the same real-world concept. But a large-scale social system as Wikipedia shows us that naming ambiguity tends to resolve over time (people reuse socially accepted names), especially if enhanced with a popularity-based recommendation system.

The solution for the representation problem of documents vs. pages, as presented in Sect. 2.3, has one drawback concerning existing RDF data. Unfortunately the world is already full of RDF statements that do not clearly distinguish documents and pages, but use URLs to refer to both. Employing our solution, encountering a URN as subject we would know that the concept is meant, but encountering a URL we would not be sure that the document is meant; the URL could be a "legacy" URL that does not conform to our distinction and is (wrongly) used to identify a concept. Our solution has therefore only limited applicability, but that is unfortunately the nature of the problem.

## 6 Conclusion

As explained in the introduction, a Semantic Wiki needs to address three questions:

1. how to annotate content?
2. how to formally represent content?
3. how to navigate content?

We have developed an elaborate model of annotations and shown how SemperWiki –as opposed to other Semantic Wikis– supports very rich annotations. We have shown how to formally represent content, and shown how SemperWiki –as opposed to other Semantic Wikis– correctly distinguishes between documents and concepts, without limiting the possible annotations. Further, we have presented how the existing technique of faceted browsing can be adjusted to flexible semistructured data, by automatically constructing facets from the data. Finally, we have developed metrics for facet (predicate) selection and techniques for object clustering inside each facet.

Faceted browsing is a superior data exploration technique [24]. We have shown how this technique can be employed for semistructured Wiki content. The technique works for any formal annotation, without conforming to a fixed data-schema; and it additionally rewards semantical annotations (because consistent use of shared terminology reduces the search space).

We are currently extending our work in several directions. First, we are integrating the faceted browser into the Web version of SemperWiki. Secondly, we are developing the clustering step of the faceted browser, and evaluating the quality of the facet construction algorithm. Thirdly, we are working on a page recommendation system, that works in the second phase of Wiki navigation and recommends (similar or related) pages to the current page, based on the structure of the Wiki content.

## References

1. P. Anick and S. Tipirneni. Interactive document retrieval using faceted terminological feedback. In *HICSS*. 1999.
2. D. Aumueller. Semantic authoring and retrieval within a wiki. In *ESWC*. 2005.
3. S. Bechhofer, *et al.* The semantics of semantic annotation. In *ODBASE*. 2002.
4. T. Berners-Lee. Putting the Web back in Semantic Web, 2005. Keynote presentation at ISWC 2005, `http://www.w3.org/2005/Talks/1110-iswc-tbl/`.
5. D. Booth. Four uses of a URL: Name, concept, web location, and document instance. `http://www.w3.org/2002/11/dbooth-names/dbooth-names_clean.htm`.
6. W. Dakka, P. Ipeirotis, and K. Wood. Automatic construction of multifaceted browsing interfaces. In *CIKM*. 2005.
7. J. Euzenat. Eight Questions about Semantic Web Annotations. *IEEE Intelligent Systems*, 17(2):55–62, Mar/Apr 2002.
8. T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, (eds.) *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.
9. S. Handschuh. *Creating Ontology-based Metadata by Annotation for the Semantic Web*. Ph.D. thesis, University of Karlsruhe, 2005.
10. S. Hawke. Disambiguating RDF identifiers, 2002. `http://www.w3.org/2002/12/rdf-identifiers/`.
11. E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining view- and ontology-based search with semantic browsing. In *Proceedings of XML Finland*. 2003.
12. E. Oren. SemperWiki: a semantic personal Wiki. In *SemDesk*. 2005.
13. E. Oren, J. G. Breslin, and S. Decker. How semantics make better wikis. In *WWW*. 2006. Poster.
14. E. Oren and R. Delbru. ActiveRDF: Object-oriented RDF in Ruby. In *Scripting for Semantic Web (ESWC)*. 2006.
15. S. Pepper and S. Schwab. Curing the web's identity crisis. `http://www.ontopia.net/topicmaps/materials/identitycrisis.html`.
16. N. Porter, (ed.) *Webster's Revised Unabridged Dictionary*. 1913 edn.
17. S. R. Ranganathan. *Elements of library classification*. Bombay: Asia Publishing House, 1962.
18. F. Rinaldi *et al.* Multilayer annotations in Parmenides. In *Proc. of the K-CAP2003 workshop on Knowledge Markup and Semantic Annotation*. 2003.
19. P. Sazedj and H. S. Pinto. Time to evaluate: Targeting annotation tools. In *Proc. of Knowledge Markup and Semantic Annotation at ISWC 2005*. 2005.
20. S. Schaffert, A. Gruber, and R. Westenthaler. A semantic wiki for collaborative knowledge formation. In *Semantics 2005*. 2005.
21. V. Sinha and D. Karger. Magnet: Supporting navigation in semistructured data environments. In *SIGMOD*. 2005.
22. R. Tazzoli, P. Castagna, and S. E. Campanini. Towards a semantic wiki wiki web. In *ISWC*. 2004.
23. M. Völkel, *et al.* Semantic wikipedia. In *WWW*. 2006.
24. K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*. 2003.

# SweetWiki : Semantic WEb Enabled Technologies in Wiki

Michel Buffa[1,2], Gaël Crova[3], Fabien Gandon[2], Claire Lecompte[3], Jeremy Passeron[3]

[1] Mainline Group, I3S Laboratory, University of Nice
`buffa@Unice.fr`
[2] Acacia Group, INRIA Sophia-Antipolis, France
`Fabien.Gandon@sophia.inria.fr`
[3] University of Nice, France

**Abstract.** Wikis are social web sites enabling a potentially large number of participants to modify any page or create a new page using their web browser. As they grow, wikis suffer from a number of problems (anarchical structure, large number of pages, aging navigation paths, etc.). We believe that semantic wikis can improve navigation and search. In SweetWiki we investigate the use of semantic web technologies to support and ease the lifecycle of the wiki. The very model of wikis was declaratively described: an OWL schema captures concepts such as WikiWord, wiki page, forward and backward link, author, etc. This ontology is then exploited by an embedded semantic search engine (Corese). In addition, SweetWiki integrates a standard WYSIWYG editor (Kupu) that we extended to support semantic annotation following the "social tagging" approach made popular by web sites such as flickr.com. When editing a page, the user can freely enter some keywords in an AJAX-powered textfield and an auto-completion mechanism proposes existing keywords by issuing SPARQL queries to identify existing concepts with compatible labels. Thus tagging is both easy (keyword-like) and motivating (real time display of the number of related pages) and concepts are collected as in folksonomies. To maintain and re-engineer the folksonomy, we reused a web-based editor available in the underlying semantic web server to edit semantic web ontologies and annotations. Unlike in other wikis, pages are stored directly in XHTML ready to be served and semantic annotations are embedded in the pages themselves using RDF/A. If someone sends or copy a page, the annotations follow it, and if an application crawls the wiki site it can extract the metadata and reuse them.

## 1 Introduction

Why did wikis become such a phenomenon? At WikiSym 2005, Ward Cunningham and Jimmy Wales [18, 19] provided some elements of an answer: "*a wiki is like a garden; users (...) must take care of it. Start with some seeds and watch it grow, and the wiki will become moderated by its users' community, (...) respect and trust the users, (...) good things happen when you trust people more than you have reason to, let everybody express his opinion, no censorship, consensus must be reached, (...) the* *wiki is adapted to a dynamic social structure because of its refactoring features. Do not impose a rigid structure, users will refactor and structure the wiki as it grows (...)*". This sounds revolutionary and indeed, social aspects are very important and cannot be neglected when talking about wikis. "Wiki introduced groundbreaking innovations at the level of technology for supporting collaborative web authoring, but also at the level of the process, philosophy and even sociology of such collaborative authoring" [16, 17]. However, even when wikis have been adopted by a large community, they are not a zero-defect solution. The main problem reported is the open structure that makes navigation, orientation and search difficult [2, 3, 23] and often fails to scale with the number of pages.

Wikipedia defines a Semantic Wiki as a "Wiki that has an underlying model of the knowledge described in its pages. (…). Semantic Wikis allow capturing or identifying further information about the pages (metadata) and their relations. Usually this knowledge model is available in a formal language, so that machines can (at least partially) process it". We believe that semantic wikis can be searched, navigated and shared with other applications in better ways than regular wikis. SweetWiki is such a semantic wiki. To address the lack of structure and structuring tools SweetWiki integrates semantic web technologies at the very core of its wiki engine. It does so without changing the ease of use that makes wikis so popular.

In section 2 we focus on the problems encountered by large wikis, in particular navigation and search, and we will explain the concepts of social tagging and folksonomies as means to improve navigation and search. In section 3 we present SweetWiki in details and insist on its innovative features. In section 4 we present related works and compare them to SweetWiki. Finally we discuss the future of semantic wikis and we mention the extensions we are working on.

## 2 Return on Experience on Wikis

Very few academic papers have addressed the intranet-wiki topic [1]. In [3] we detailed two experiences we conducted over several years with intranet wikis: (1) six years ago we installed a wiki which is today at the heart of the intranet of the Computer Science department of the University of Nice, with about 400 regular users [4]; and (2) since 2001, we have a close relationship with the ILOG Company which has developed a very impressive wiki-powered intranet [2].

Companies like Google, Motorola and the New-York Times have made public the way they use a wiki in their organization [5, 6, 7, chapter 12 of 8]. In [3] we defined the goals of a business organization intranet and showed how the web technology and tools helped or failed to reach these goals. We focused on the wiki concept and concluded that its success relies on several social conditions that cannot always be found in the business organization's culture (e.g. people must understand why they are working together; there must not be too much social friction, etc.)

Finally, wikis are not a zero-defect solution. The main problem reported is the difficulty experienced by users in finding their way in navigating and searching the wiki, especially when it becomes large. Traditional wikis do not scale very well unless their structure and structuring tools are improved.

ILOG uses the aspSeek search engine to index and search the resources of their wiki-based intranet. Looking at their logs over time it become apparent that the use of the search engine suddenly dropped, and after a short time people just stopped using it. Interviews and investigations proved that the assumption that everybody knows how to use a search engine was wrong [2, 3]. In addition, on the Internet, people can accept not finding what they are searching for -maybe it is just not out there; on an intranet, when people know that what they are looking for is there, they don't understand why a search engine does not find it and finally distrust it altogether. After many usability tests, the user interface for the search engine was improved, but people still complain about the difficulty to find things on the wiki. The interviews and questionnaires at the University of Nice confirmed the same problems with their wiki: search is considered less and less useful as the wiki grows [3].

The New York Times Digital used a wiki that became huge, with thousands of pages and several problems occurred [8]. They first added navigation bars, but this did not solve the navigation problem. WikiNames collision was another problem: when one creates a wiki page one has to choose a name for this page; after two years, users sometimes had to try dozens of different names before finding a name that had not already been used. The original idea with WikiNames collision was that if you find out that there is a page that already exists with the same name, you would "join it" because it is supposed to be the best place for saying what you have to say. But it just did not work at the NY Digital: people wanted to create their own page. They invented funny WikiNames that were no longer meaningful according to their content. Navigation and searching were so difficult that it was nearly impossible to find a document without having bookmarked it. Everybody realized that the wiki was becoming a mass of increasingly inaccessible pages but the user community was not ready to do the necessary work for refactoring and organizing it all. The writing and publishing process in a national newspaper is very structured, and NY Times Digital's employees could not get any trace of such a workflow in the wiki. What appeared as a promising tool that had been widely adopted turned out to be a faulty solution for helping in the publishing process. Structure and organization became such a big problem that they had to stop relying on a wiki. It was not completely abandoned but relegated to a shared notepad, with the structured work being done in other tools.

One can argue that the choice of another wiki engine could have changed the outcome of this experience, in particular a wiki engine supporting the concept of workspaces like TWiki, MoinMoin, JotSpot, SocialText, etc. But we think the problem runs deeper. Wikis are designed to be structured by the users themselves. People differ from each other, every individual has his own way of classifying and organizing data, and this may change over time. A hierarchical structure like the one proposed by the workspaces is certainly a good thing from a technical point of view but it provides a superficial modularization of a wiki [14]. Horizontal navigation (following links in the page itself) is the means most people use. Usability tests showed that most people at ILOG don't even know the names of the different workspaces.

Interestingly, a common behavior we noticed is that users started to add category keywords on the wiki pages. These keywords are WikiNames that lead to pages that propose hyperlinks to all pages belonging to the same category. This naïve classification helps but does not scale. We drew a parallel between this emergent behavior and the phenomenon of social tagging used in the public Web by popular sites such as del.icio.us and flickr.com and also widely used in blogs. You can annotate your blog entries or the pictures you posted to flickr by associating keywords to them forming a quasi-classification on-the-fly. These tags are used by technorati.com's web bots and a link to your tagged resource is added to the other entries that share the same tag. The main interest in this way of tagging is its social approach to classification. People can use whatever tags they feel represent the content of their writing, but they may find out that this tag has never been used before. So there is a higher probability they will add other tags that link them to other resources. If one creates a new tag, it is just added and will be proposed as a choice when another person enters a tag that starts with the same letters, and maybe this person will in turn choose it. This way, users as individuals, can categorize their writing any way they want and at the same time begin a grass roots taxonomy or *folksonomy*.

Social tagging and folksonomies are the subjects of debate in different communities, including the semantic web community [12]. These concepts are often described as an alternative to ontologies and to the semantic web approach in general [11, 15]. Grubert in [15] published an interesting survey of these different points of view. Some describe tags and folksonomies as "cheap metadata for the masses" (taxonomies and ontologies being the land of experts) [33] and others think they are the one true way [11] and that a flat-hierarchy is more human-brain-friendly, imitating the word-as-a-label-for-things. But this is also the main drawback of the tags: human-language-structured thought can jump between concepts; the same word can have totally different meanings. Last but not least: each human has their own world-experience, their own tagging-system that may not be generalized. Where categories are managed by specialists to achieve the best classification, tags are users' rough approximation of classification for a practical use (*ethnoclassification).*

## 3 SweetWiki

Wikis were designed in the mid-nineties to exploit the web technologies of the time *i.e.* mainly HTML, HTTP and URIs. To make up for the lack of simple remote edition and storage facilities Wikis developed WikiML variants, WikiWords for specifying hypertext links, simple versioning mechanisms, etc. The idea of SweetWiki is to re-visit the design rationale of Wikis, taking into account the wealth of new standards available for the web eleven years later to address some of the shortcomings identified through experience.

After evaluating several wiki engines (regular or semantic), we decided to write a new engine because our vision of the wiki of the future was not compatible with what we found in existing wikis. We wanted our wiki to:

- *rely on web standards*: standards for the wiki page format (XHTML), for the macros one can put in a page (JSPX/XML tags), etc.;
- *be articulated around a semantic engine* that supports semantic web languages like RDF/RDFS/OWL/SPARQL/etc.;

- *get rid of the WikiML* dialects used and modified by most wiki systems. We took this decision based on the painful experiences we had with the ILOG intranet where we integrated WYSIWYG editors in TWiki. We encountered many problems during the translation between the WikiML and the XHTML languages. Many WikiML variants do not support all the XHTML produced by the existing editors. Mixing wikiML with XHTML code was not a clean approach and users were asking for more intuitive interfaces. Furthermore, we wanted an alternative to translating WikiML to XHTML each time a page is viewed and doing the reverse translation each time a page is saved.
- propose faceted navigation and enhanced search tools;
- propose metadata editing in the same user interface used for content editing.

### 3.1 Principles

Wikis are Web sites where pages are organized around WikiWords and sometime other constructs such as WikiWebs. To go beyond this informal hyperlink structure, semantic tagging and restructuring functionalities are needed. To make explicit, manipulate and exploit such a structure we introduced two ontologies:

- *an ontology of the wiki structure*: the wiki concepts are usually buried in their *ad hoc* implementations; this structure is a special kind of meta-data (forward links, authors, keywords, etc.) relying on an ontology of wikis (WikiPage, WikiWord, WikiWeb, etc.); By making this structure and its ontology explicit, we can reason on it, e.g. to generate navigation pages, we can modify it, e.g. re-engineer the wiki structure, and we can build on it, e.g. interoperability between several wikis.
- *an ontology of the topics*: each wiki page addresses one or more topics. In order to ease navigation while maintaining the usual simplicity, we implemented the usual keyword mechanism with a domain ontology shared by the whole wiki. By making this topic ontology explicit we can once again, reason on it, e.g. find semantically close topics, make complex queries, we can modify it, e.g. tidy the ontology, merge equivalent concepts, etc.

The ontology of the wiki structure is maintained by developers of the wiki. The domain ontology is enriched directly by the users and may be restructured by administrators of the site to improve the navigation and querying capabilities. Other ontologies may be added at runtime and be immediately accessible to users. To implement these principles we relied on a semantic web server architecture described in the following section.

### 3.2 Architecture

Starting from the users' side, SweetWiki is based on Kupu[34] an XHTML editor in Javascript which allows us to replace traditional WikiML editing by a WYSIWYG interface in the user's browser. Since editing directly produces XHTML, we decided to use it as a persistence format. Thus, once saved, a page stands ready to be served by the Web server.

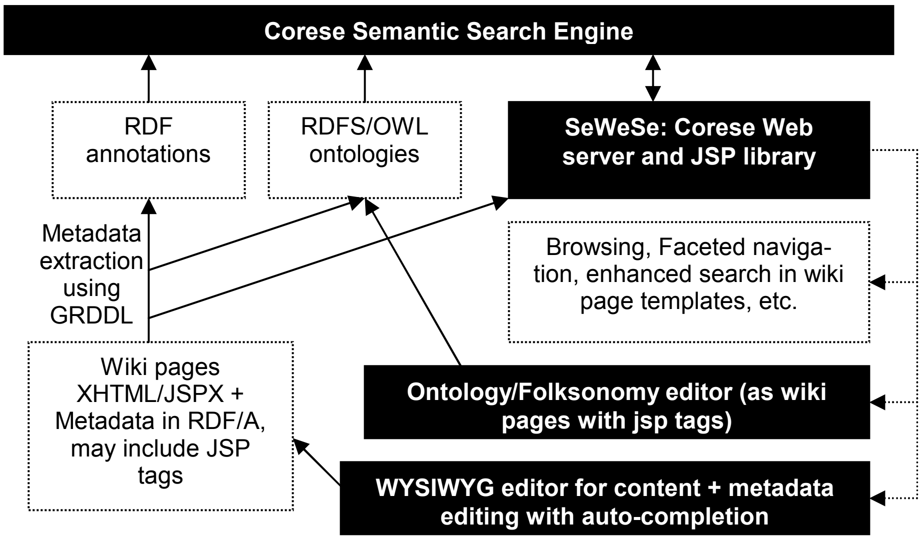

Corese Semantic Search Engine
RDF annotations
RDFS/OWL ontologies
SeWeSe: Corese Web server and JSP library
Metadata extraction using GRDDL
Browsing, Faceted navigation, enhanced search in wiki page templates, etc.
Wiki pages XHTML/JSPX + Metadata in RDF/A, may include JSP tags
Ontology/Folksonomy editor (as wiki pages with jsp tags)
WYSIWYG editor for content + metadata editing with auto-completion

**Fig 1.** SweetWiki architecture

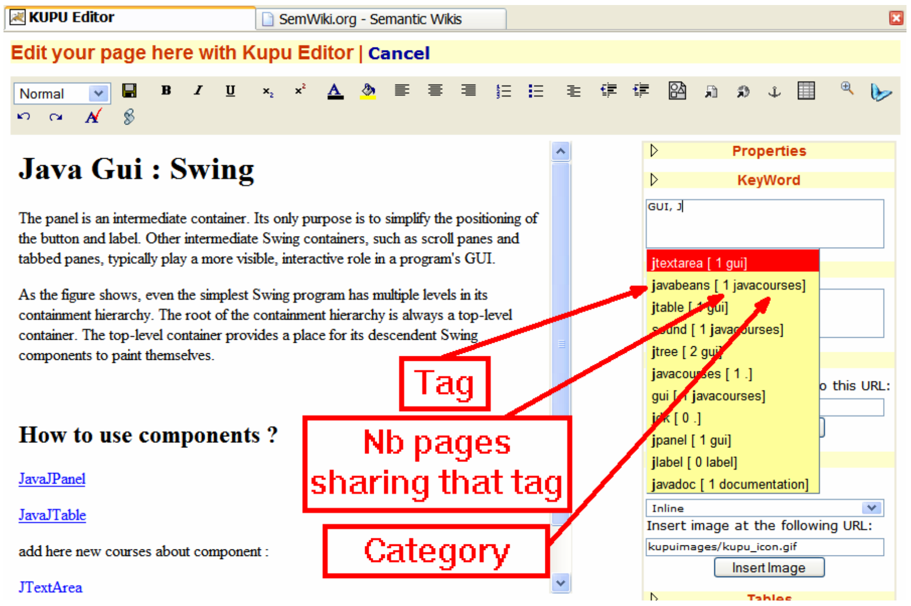
Pages are standalone XHTML files including their metadata and thus they can be crawled by other applications. To address structuring and navigation problems in wikis we wanted to include tagging at the core of the wiki concept, thus we integrated four new web technologies: RDF/S and OWL are W3C recommendations to model metadata on the web [35]; SPARQL is a recommendation for a query language for RDF [39]; RDF/A is a draft syntax for Embedding RDF in XHTML [36]; GRDDL is a mechanism for getting RDF data out of XML and XHTML documents using explicitly associated transformation algorithms, typically represented in XSLT [37].

With RDF/A we have both page data and metadata in the same standalone file, we use pure XHTML, pages can be crawled by external applications or saved by users using their browser without any loss of information.

Figure 1 summarizes the architecture of SweetWiki. The implementation relies on the CORESE semantic search engine for querying and reasoning [38] and on SEWESE, its associated web server extension that provides API and JSP tags to implement all the web-based interfaces, as well as a set of generic functionalities (security management, ontology editors, web application life cycle, etc.)

### 3.3 Focus on tagging: using semantic web technology to implement a folksonomy

We are not extremists and like [13 and 14], we propose a mixed approach in order to "organize the tags": we link the tags together within a folksonomy described using the semantic web languages, where tags are organized in a hierarchy and related one to another using relationships like `subClassOf`, `seeAlso`, etc.. Grubert goes further and proposed in [15] to define "an Internet ecology" for folksonomies *i.e.* an ontology for describing folksonomies. Like him, we believe that social tagging minimizes cost and maximizes user participation. So we do support social tagging in SweetWiki, but we also think that these tags must be organized. The system we have implemented helps users build a better folksonomy while relying on standard semantic web technologies for organizing and maintaining the folksonomy. SweetWiki uses folksonomies and social tagging as a better way to categorize the wiki documents [9, 10].



Java Gui : Swing

The panel is an intermediate container. Its only purpose is to simplify the positioning of the button and label. Other intermediate Swing containers, such as scroll panes and tabbed panes, typically play a more visible, interactive role in a program's GUI.

As the figure shows, even the simplest Swing program has multiple levels in its containment hierarchy. The root of the containment hierarchy is always a top-level container. The top-level container provides a place for its descendent Swing components to paint themselves.

How to use components ?

JavaJPanel

JavaJTable

add here new courses about component :

JTextArea

**Fig 2.** Tags are suggested as the user enters a keyword, the number of pages sharing that tag is displayed as well as the related category.

SweetWiki integrates a standard WYSIWYG editor (Kupu) that we extended to directly support semantic annotations following the "social tagging" approach. As shown in Figure 2, when editing a page, the user can *freely* enter some keywords in an AJAX-powered textfield. As the user types, an auto-completion mechanism proposes existing keywords by issuing SPARQL queries to the semantic web server in order to identify existing concepts with compatible labels and shows the number of other pages sharing these concepts as an incentive to use them. Furthermore, related categories are also displayed in order to address the ambiguity of homonymy. With this approach, tagging remains easy (keyword-like) and becomes both motivating and unambiguous. Unknown keywords are collected and attached to new concepts to enrich the folksonomy. Later on, community experts may reposition them in the ontology, edit them, etc. The feedback coming from the tags is useful for improving the ontology.

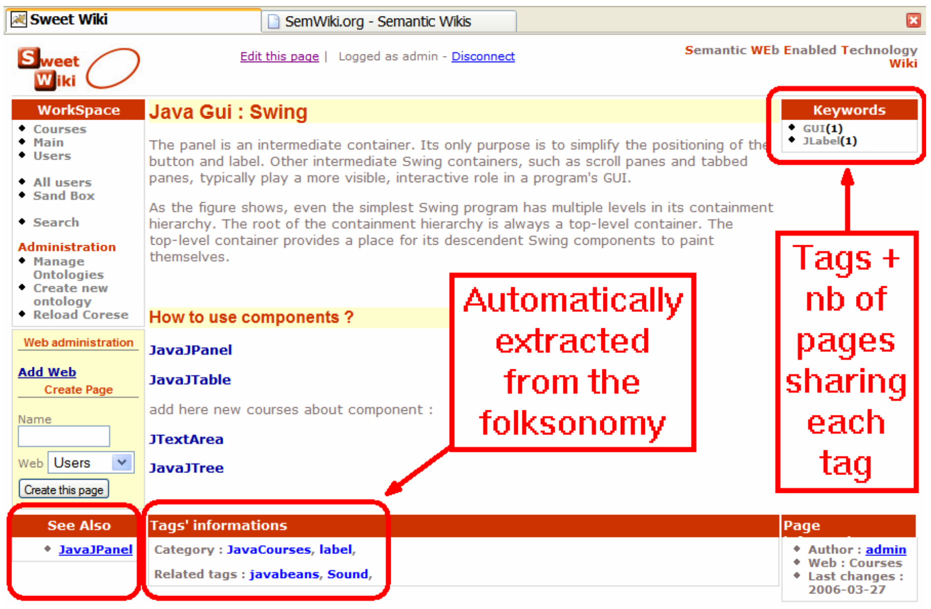
When the page is saved in XHTML the associated metadata are saved inside using the RDF/A syntax, as illustrated by Figure 3. Besides the topic tags (keywords and see also), metadata include contextual information (e.g. author, last modification, etc.). Thus the page stands ready to be served by a web server.

```
<head>
<link href="http://www.essi.fr/sweetTwiki/wiki.rdfs" rel="schema.WIKI"/>
<meta content="text/html; charset=ISO-8859-15" http-equiv="Content-
Type"/>
<meta content="JavaGui" name="WIKI.name"/>
<link href="#admin" rel="WIKI.author"/>
<meta content="Description - non implemente" name="WIKI.description"/>
<meta content="2006-3-27" name="WIKI.modification"/>
<link href="#Courses" rel="WIKI.hasForWeb"/>
<link href="#JavaJPanel" rel="WIKI.forwardLink"/>
<link href="#JavaJTable" rel="WIKI.forwardLink"/>
<link href="#JTextArea" rel="WIKI.forwardLink"/>
<link href="#JavaJTree" rel="WIKI.forwardLink"/>
<link href="http://www.inria.fr/acacia/java-ontology#GUI"
      rel="WIKI.hasForKeyWord"/>
<link href="http://www.inria.fr/acacia/java-ontology#JLabel"
      rel="WIKI.hasForKeyWord"/>
<link href="#JavaJPanel" rel="WIKI.seeAlso"/>
</head>
```

**Fig 3.** How the tags and SeeAlso metadata are described in the wiki page file. This sample comes from the SweetWiki page in Figure 4. The metadata that use the wiki ontology are automatically generated and never visible by a user. All the RDF code is hidden from regular users.

During the save process, the newly saved page metadata are extracted using the semantic web server API. This API uses a GRDDL XSLT stylesheet to extract the metadata in RDF/XML format and feed them to the CORESE engine. Other wiki pages that hold “create links” (links created before the current page existed) are also updated and their metadata extracted using the same process. The CORESE engine is then used to generate faceted navigation widgets: the semantics of the tags is used to derive related topics, query the engine on similar pages using SPARQL queries, etc. (see Figure 4).

The pages content is saved in pure XHTML and is ready to be served (without any further translation as required with a wikiML variant). When a SweetWiki document is requested by a web browser, templates are used in order to integrate the faceted navigation widgets around the page content. These templates may be changed like the skins of TWiki for example, they are just used for decorating the final document.



Java Gui : Swing

The panel is an intermediate container. Its only purpose is to simplify the positioning of the button and label. Other intermediate Swing containers, such as scroll panes and tabbed panes, typically play a more visible, interactive role in a program's GUI.

As the figure shows, even the simplest Swing program has multiple levels in its containment hierarchy. The root of the containment hierarchy is always a top-level container. The top-level container provides a place for its descendent Swing components to paint themselves.

How to use components ?

JavaJPanel

JavaJTable

add here new courses about component :

JTextArea

JavaJTree

Category : JavaCourses, label,

Related tags : javabeans, Sound,

Author : admin
Web : Courses
Last changes : 2006-03-27

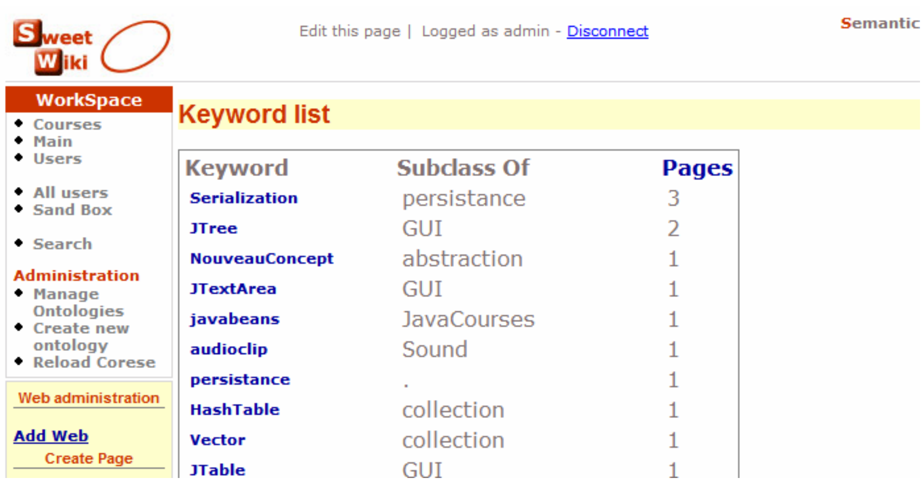
**Fig 4.** Faceted navigation links extracted from the tags.

## 3.4 Ontology editor for maintaining and re-engineering the folksonomy

Supervising tools are integrated in SweetWiki by relying on the semantic web server SeWeSe. They are used to monitor the wiki activity itself running SPARQL queries over the metadata e.g. usage frequency for tags (See Figure 5), new tags, orphan pages, etc.

In order to maintain and re-engineer the folksonomy, SweetWiki also reuses web-based editors available in SeWeSe. In our examples we tagged some Java courses, using a Java ontology. Selecting this ontology in the editor, one can add/remove/edit concepts (Figure 6). In particular, if a tag/concept has been recently added it may be inserted in the hierarchy. Figure 7 shows the concept editing tool.

Using these editors, the folksonomy and the annotations may be updated. For instance, community experts can pick a couple of tags and declare semantic relations between them such as `subClassOf`. They may also merge concepts when two tags are synonymous, etc. Enhancements of the ontology seamlessly improve content sharing: search and faceted navigation benefit directly from the updates. The way the system is designed, versioning cannot break the annotations. If a tag is suddenly missing it is just treated as a new tag and if many pages exist with the old tag (pages are not touched in tag editing process), the tag would re-appear (with a high number of tagged pages, encouraging other people to use it). It is also possible to merge tags as several tags may be labels of the same concept.



Sweet Wiki

Edit this page | Logged as admin - Disconnect

Semantic

WorkSpace
- Courses
- Main
- Users

- All users
- Sand Box

- Search

Administration
- Manage Ontologies
- Create new ontology
- Reload Corese

Web administration

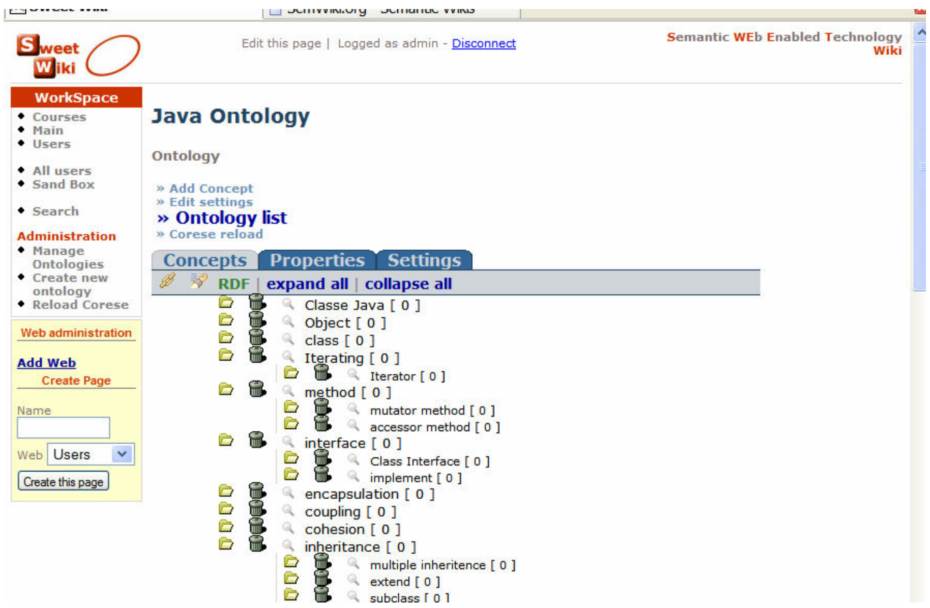
Add Web

Create Page

## Keyword list

| Keyword | Subclass Of | Pages |
|---|---|---|
| Serialization | persistance | 3 |
| JTree | GUI | 2 |
| NouveauConcept | abstraction | 1 |
| JTextArea | GUI | 1 |
| javabeans | JavaCourses | 1 |
| audioclip | Sound | 1 |
| persistance | . | 1 |
| HashTable | collection | 1 |
| Vector | collection | 1 |
| JTable | GUI | 1 |

**Fig 5.** tags sorted by popularity



Sweet Wiki

Edit this page | Logged as admin - Disconnect

Semantic WEb Enabled Technology Wiki

WorkSpace
- Courses
- Main
- Users

- All users
- Sand Box

- Search

Administration
- Manage Ontologies
- Create new ontology
- Reload Corese

Web administration

Add Web

Create Page

Name

Web Users

Create this page

## Java Ontology

Ontology

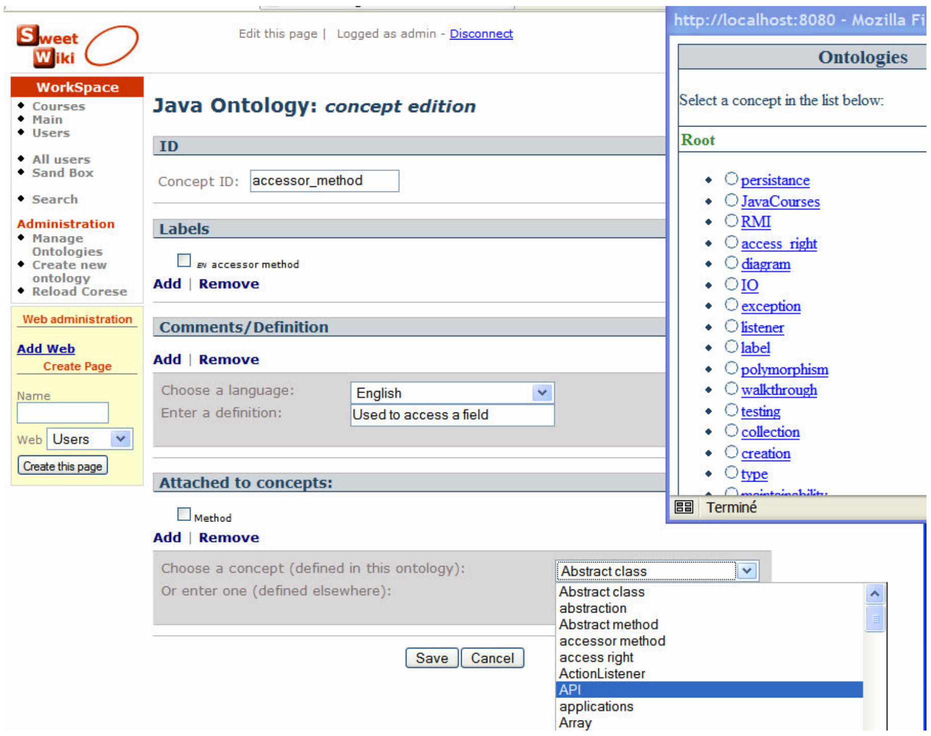
» Add Concept
» Edit settings
» Ontology list
» Corese reload

Concepts | Properties | Settings

RDF | expand all | collapse all

```
Classe Java [ 0 ]
Object [ 0 ]
class [ 0 ]
Iterating [ 0 ]
    Iterator [ 0 ]
method [ 0 ]
    mutator method [ 0 ]
    accessor method [ 0 ]
interface [ 0 ]
    Class Interface [ 0 ]
    implement [ 0 ]
encapsulation [ 0 ]
coupling [ 0 ]
cohesion [ 0 ]
inheritance [ 0 ]
    multiple inheritence [ 0 ]
    extend [ 0 ]
    subclass [ 0 ]
```

**Fig 6.** The ontology editor, here illustrated with the Java courses folksonomy. It is possible to add/edit/remove concepts, properties/settings from here, even import ontologies and merge.



Java Ontology: *concept edition*

ID

Concept ID: accessor_method

Labels

☐ EN accessor method

Comments/Definition

Choose a language: English

Enter a definition: Used to access a field

Attached to concepts:

☐ Method

Choose a concept (defined in this ontology): Abstract class

Or enter one (defined elsewhere):

Abstract class
abstraction
Abstract method
accessor method
access right
ActionListener
API
applications
Array

Name

Web Users

Ontologies

Select a concept in the list below:

Root

- persistance
- JavaCourses
- RMI
- access_right
- diagram
- IO
- exception
- listener
- label
- polymorphism
- walkthrough
- testing
- collection
- creation
- type

**Fig 7.** Editing a concept.

## 4 Related Work and Positioning

Many semantic wiki projects are being developed. Looking at the state of the art we can distinguish between approaches considering "the use of wikis for ontologies" and approaches considering "the use of ontologies for wikis" (while a few engines merge both approaches)..

Most of the current projects on semantic wikis fall in the first category *i.e.* they consider wiki pages as concepts and typed links (in the page content) as relations or attributes. In this model, called a "Wikitology" in [25], the Wiki becomes the front-end of the ontology.

One of the first wikis to fall into this category is Platypus [21] that imposes separately editing the metadata for each wiki page in a "Wiki Metadata page". It supports basic ontology editing but with no consistency check between the annotations and the ontology. It does not come with a reasoning engine and supports only basic queries. Semantic metadata are used for improving navigation but the main drawback is that the users have to switch between editing normal text and editing semantic annotations as these activities are done using two distinct text-based editors. Other wikis like SHAWN [27] offer similar features. The other wikis presented in this category address Platypus' shortcomings by allowing semantic annotations directly in the text of the page, usually as typed links.

Rise [25] also falls in the first category: the ontology used by the community is edited via the Wiki itself and a set of naming conventions is used to automatically determine the actual ontology from the Wiki content. A proprietary language is used for describing the metadata while RDF exportation is possible. Semantic information is used for navigation and consistency checks. The ontology is built as wiki pages are updated (rebuilt each night).

Rhizome [20] supports a modified version of WikiML (ZML) that uses special formatting conventions to indicate semantic intent directly in the page content. Pages are saved in RDF and another editor can be used to edit the RDF directly. Rhizome authors admit that this feature is dangerous as one can break the wiki behavior by entering bad RDF. To mitigate the inherent dangers of this level of openness, Rhizome Wiki provides fine-grain authorization and validation alongside the use of contexts. It is not clear how metadata improve the wiki behavior; there is no advanced search and no help for navigating the wiki so far. RDF-Wiki [29] is similar to Rhizome in that it allows RDF annotations for external processing.

SeMediaWiki [26] is based on MediaWiki. In contrast to Rise, typed links can also be used for specifying attributes of the page. For example, the following text: *San Diego is a [[is a::city]] located in the southwestern corner of [[is located in::California]]* establishes the facts "San Diego is a city" and "San Diego is located in California". While the text *Its coordinates are [[coordinates:=32°42'54"N, 117°09'45"W]]* defines an attribute named "coordinates". These data are used for faceted navigation. SeMediaWiki translates these metadata into RDF but does not use a reasoning engine. Other semantic extensions of MediaWiki are available such as [32] but are still at early stage of development.

Makna [31] is based on JSPWiki and provides semantic extensions as typed links. It comes with the JENA reasoning engine that allows complex queries. Its text-based editor proposes extra HTML forms (ajax-powered) for quering the semantic engine and look for concepts/properties/relationships. This is very useful in the case of a large ontology.

WikSar [22, 23] enables users to enter semantic annotations from the wiki text editor using WikiWords. For example: if in a page named "PrinceHamlet", there is a line "FigureBy: WilliamShakespeare", it can be seen as a RDF statement. By combining all such embedded statements, a formal ontology emerges within the Wiki. The editor is text-based and proposes neither help of any kind to the user nor any consistency check. As pages are saved, the metadata are used to propose faceted navigation. WikSar supports queries in RDQL and SPARQL and queries can be embedded in wiki pages or templates. A distinctive feature of WikSar is the "interactive graph visualisation and navigation" tool that can be used for exploring the wiki through its metadata.

Typed links are powerful but one has to remember each concept, relation, property before typing it and this is not very practical. Ace Wiki goes further: with AceWiki [40] one can add and modify sentences written using the ACE language (Attempto Controlled English [41]), through the use of an interactive Ajax-based editor. The editor is aware of the background ontology, and provides guidance to the user by proposing only valid completions. Moreover, the editor can be used to extend the ontology by creating new concepts, roles and individuals. Therefore, it is also, de facto, a simple ontology editor.

The second family of approaches focuses on "the use of ontologies for wikis". IkeWiki [24] supports both WikiML and WYSIWYG editing of page content and metadata, as well as page tagging. The editor comes with some AJAX features like auto-completion on metadata. It requires an existing ontology to be loaded. Some support for ontology editing is provided. It uses Jena and metadata are used for navigation and page rendering. Annotations can be visualized in a frame next to the wiki page. Each node is a link. IkeWiki has a very nice user interface.

SweetWiki also falls into this second category. It does not implement the Wikitology model yet but we have made provision for such an evolution. So far we support the concepts of social tagging and folksonomy. SweetWiki is close to WikSar since they share many features like usage-driven ontology building, queries embedded in the wiki pages (as JSP tags), edition of metadata and page content in the same editor. SweetWiki adds a reasoning engine and an extensible WYSIWYG editor for both content and metadata, (like IkeWiki or Makna). The SweetWiki editor is AJAX-enhanced and annotating pages leads to instant gratification for users in two ways since as they type: (a) they can see an instant display of faceted links the annotation will add to the page; (b) an auto-completion mechanism proposes existing concepts from the ontology, related categories and number of pages sharing that annotation as an incentive to reuse existing tags. Furthermore, SweetWiki comes with complete user-friendly ontology supervising and editing tools. However, SweetWiki is not dedicated to *collaborative* ontology management (e.g. OntoWiki [30]) but we are currently brainstorming on how we could add such capabilities to our engine.

## 5 Discussion

To summarize the overall scenario explored in SweetWiki, we have proposed an innovative approach that allows users to edit wiki pages and tag them using a shared conceptualization behind the scenes. In addition community experts can check the underlying model being built, look at the tags/concepts proposed by the users and (re)organize them. If this happens, annotations that users entered are not changed, but faceted navigation and search based on semantic queries are improved by new links.

As the reader may have noticed in the snapshots, our current experimentation uses an online course on Java as a test case. The learning objects are organized as wiki pages and annotated with concepts of the java language.

A number of evolutions are currently under consideration:

- *Including forms in wiki pages*: the easy creation of pages makes it tempting to extend the concept to create small web applications in particular processing small forms. SeWeSe proposes a language merging SPARQL and JSP to generate forms from the underlying ontology. We are planning on integrating this facility to ease the development of small front-ends e.g. dedicated advanced search.
- *Natural language processing for automatic tagging*: several wikis are starting to analyze the text of wiki pages to suggest potential keywords. Seamless deduction

of metadata could be achieved by applying natural language processing techniques to (semi-)automatically derive keywords from the existing content and its context.

- *Complete versioning*: support versioning of textual content, semantic annotations and underlying ontologies at the same time;
- *Collaborative management of the folksonomy*: provide groupware to assist the distributed lifecycle of ontologies; here the wikitology approach seems only natural and we need more powerful tools to implement it efficiently.

This last point brings us back to the two-way vision *wikis for ontologies and ontologies for wikis*. This division of the current approaches is only symptomatic of the early times of semantic wikis. In the long term future semantic wikis should merge these two approaches as two facets of the same coin as some projects already started to do it; the objective being to turn this two-way vision into a virtuous circle where users maintain the ontology and the wiki *at the same time* without any artificial distinction between them. For us SweetWiki is an ideal experimentation platform to test this vision. We are just starting to experiment with the possibilities on different focus groups.

## Acknowledgements

Thanks a lot to Sebastian Shaffert from the semantic wiki mailing list (swikig@aifb.uni-karlsruhe.de) for starting the “wiki of the future debate”. It was interesting to see that we were thinking about the same things at the same time. Authors would also like to thanks the ILOG company and the COLOR commission of INRIA for their scientific and financial support to the “Usable Intranet” project, in the context of which the work presented here was developed.

## References

1. Stenmark, D. (2005). "Knowledge sharing on a corporate intranet: Effects of re-instating web authoring capabilities". Proceedings of ECIS 2005, Regensburg, Germany, 26-28 May 2005.
2. C.Chat and C.Nahaboo. (2006). “Let's Build an Intranet at ILOG Like the Internet!”, submitted to the Intraweb workshop, WWW Conference 2006, Edinburgh.
3. Buffa, M. (2006). “Intranet Wikis”. Accepted to the Intraweb workshop, WWW Conference 2006, Edinburgh.
4. Buffa M., Sander P., Grattarola J.-C. (2004). “Distant cooperative software development for research and education: three years of experience”, In proceedings of CALIE’04, Grenoble, France.
5. N.Finck (Digital Web Magazine), M.Hodder, and B.Stone (Google). (2005) “Enhancing Internal Communications with Blogs, Wikis, and More”: http://www.nickfinck.com/presentations/bbs2005/01.html
6. D.Merrill’s. (2005) “A view into Google's inner workings”, audio report about presentation at Vortex 2005: http://www.citizenvalley.org/blocnotes/index.php?id_article=241401
7. Inside Google, Rough Type Blog, http://www.roughtype.com/archives/2005/10/inside_google.php
8. Cunningham, W and Leuf, B. (2001). “The Wiki Way: Quick collaboration on the web”. Addison-Wesley, Boston.
9. S. Powers, (2005), “Cheap Eats at the Semantic Web Caf”é, http://weblog.burningbird.net/archives/2005/01/27/cheap-eats-at-the-semantic-web-cafe/
10. T. Hammond, T. Hannay, B. Lund, and J. Scott, (2005), “Social Bookmarking Tools, a General Review”, D-Lib Magazine, April 2005, Volume 11 Number 4, http://www.dlib.org/dlib/april05/hammond/04hammond.html
11. Shirky. C. (2005).” Ontology is overrated”. Etech Talk. http://www.itconversations.com/shows/detail470.html

12. Smith. G. (2005). “IA Summit Folksonomies Panel”. http://atomiq.org/archives/2005/03/ia_summit_folksonomies_panel.html
13. Bird. F. (2005). “Some ideas to improve tags use in social software, flat hierarchy versus categories in social software”. http://fredbird.org/lire/log/2005-05-17-tags-structuration-proposal
14. Olsen. H. (2005). “Navigation blindness, how to deal with the fact that people tend to ignore navigation tools”. The Interaction Designer’s Coffee Break, Issue 13, Q1 2005. http://www.guuui.com/issues/01_05.php
15. Gruber. T. (2005). Folksonomy of Ontology: A Mash-up of Apples and Oranges. First on-Line conference on Metadata and Semantics Research (MTSR'05). http://mtsr.sigsemis.org/
16. A.Désilet, S.Paquet, N.G.Vinson . (2005). « Are Wikis Usable? », proceedings of the 2005 International Symposium on Wikis, Oct 16-18, San Diego, California, USA.
17. S. Shah. (2004). « The internet is Jain : How Gunslingin’ Technolibertarianism Leads to Lotus Petals », in proceedings of New Forms Festival, Technography, Vancouver, BC, 2004.
18. Cunningham. W. (2005). Ross Mayffeld’s notes on Cunningham’s Keynote at Wikisym 2005. “Ward Cunningham on the Crucible of Creativity”. http://ross.typepad.com/blog/2005/10/ward_cunningham.html
19. Wales. J, founder of Wikipedia / Presentation at Wiki Symposium 2005. http://recentchanges.info/?p=5
20. Souzis. A. (2005). Building a Semantic Wiki. EEE Intelligent Systems, vol. 20, no. 5, pp. 87-91, September/October, 2005.
21. Campanini S.E., Castagna P. and Tazzoli R. (2004). Platypus Wiki: a Semantic Wiki Wiki Web. Semantic Web Applications and Perspectives, Proceedings of 1st Italian Semantic Web Workshop. December 2004. http://semanticweb.deit.univpm.it/swap2004/cameraready/castagna.pdf
22. AumuellerD. and Auer S. (2005). Towards a Semantic Wiki Experience – Desktop Integration and Interactivity in WikSAR. Proc. of 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure, Galway, Ireland, Nov. 6th, 2005. http://www.semanticdesktop.org/SemanticDesktopWS2005/final/22_aumueller_semanticwikiexperience_final.pdf
23. Aumueller D. (2005). SHAWN: Structure Helps a Wiki Navigate. Proceedings of the BTW-Workshop, March 2005. W. Mueller and R. Schenkel editor. http://dbs.uni-leipzig.de/~david/2005/aumueller05shawn.pdf
24. Schaffert S., Gruber A., and Westenthaler R.: A Semantic Wiki for Collaborative Knowledge Formation . In: Semantics 2005, Vienna, Austria. November 2005.
25. Decker B., Ras E., Rech J., Klein B. and Hoecht C. Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis. Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE), held at the 4th International Semantic Web Conference (ISWC 2005) November 6th - 10th, 2005,Galway, Ireland
26. Krötsch M. and Vrandečić D. and Völke M. (2005). Wikipedia and the Semantic Web - The Missing Links. Proceedings of the WikiMania2005. http://www.aifb.uni-karlsruhe.de/WBS/mak/pub/wikimania.pdf
27. Aumueller, D. (2005). SHAWN: Structure Helps a Wiki Navigate. BTW-Workshop “WebDB Meets IR”, Karlsruhe, 2005-05. http://the.navigable.info/2005/aumueller05shawn.pdf,
28. WikiOnt: http://boole.cs.iastate.edu:9090/wikiont/,
29. RDF Wiki: http://infomesh.net/2001/05/sw/#rdfwiki,
30. Hepp M., Bachlechner D. and Siorpaes K. (2005). OntoWiki: Community-driven Ontology Engineering and Ontology Usage based on Wikis. Proceedings of the 2005 International Symposium on Wikis (WikiSym 2005). http://www.heppnetz.de/files/ontowikiDemo-short-camera-ready.pdf
31. Dello K., Tolksdorf R. and Paslaru E. (2005). Makna. Free University of Berlin. http://www.apps.ag-nbi.de/makna/wiki/About
32. Muljadi H. and Takeda H. (2005). Semantic Wiki as an Integrated Content and Metadata Management System. Proceedings of ISWC 2005, Galway, Ireland.
33. Merholz P. (2004). Metadata for the Masses. Adaptive Path Blog. http://www.adaptivepath.com/publications/essays/archives/000361.php
34. Kupu : http://kupu.oscom.org/
35. W3C, Semantic Web Activity, http://www.w3.org/2001/sw/ et http://www.w3.org/2001/sw/Activity
36. RDF/A Primer 1.0 Embedding RDF in XHTML http://www.w3.org/2001/sw/BestPractices/HTML/2006-01-24-rdfa-primer
37. Gleaning Resource Descriptions from Dialects of Languages (GRDDL) http://www.w3.org/2004/01/rdxh/spec
38. O. Corby, R. Dieng-Kuntz, C. Faron-Zucker, Querying the Semantic Web with the CORESE search engine. In Proc. of the 16th European Conference on Artificial Intelligence (ECAI'2004), Valencia, 2004, IOS Press, p. 705-709
39. SPARQL Query Language for RDF http://www.w3.org/TR/rdf-sparql-query/
40. AceWiki : http://gopubmed.biotec.tu-dresden.de/AceWiki/
41. Attempto Controlled English (ACE) : http://www.ifi.unizh.ch/attempto/

# iMapping Wikis— Towards a Graphical Environment for Semantic Knowledge Management

Heiko Haller, Felix Kugel, Max Völkel

Forschungszentrum Informatik (FZI), University of Karlsruhe, Germany
{`heiko.haller, max.voelkel, felix.kugel`}`@fzi.de`,
`http://www.fzi.de/ipe`

**Abstract.** iMapping is a visual technique for structuring information objects. It is based on research in the fields of visual mapping techniques, Information Visualisation and cognitive psychology. iMapping uses a Zooming User Interface to facilitate navigation and to help users maintain an overview in the knowledge space. An iMap is comparable to a large whiteboard where wiki-pages can be positioned like post-its but also nested into each other. Augmenting Wikis by providing spatial browsing and zooming facilities makes it easier to structure content in an intuitive way. Especially semantic wikis, which typically contain more fine-grained content, and stress the structure between information items, can benefit from a graphical approach like iMapping, that allows to display multiple such items and multiple wiki-pages in one view. This paper describes the iMapping approach in general, and briefly how it will be applied as a rich client front-end to the SemWiki engine.

## 1 Introduction

Wikis have proven to be useful devices to easily store and manage structured information, and are also increasingly being used for Personal Knowledge Management. Semantic technologies however have not found widespread use so far. Using wikis to also author formal (semantic) knowledge structures seems a promising approach. However for these semantic technologies to be widely used, it is crucial that they are very easy to author and that they do not constrict the user in his work. Also hypertext research has shown, that users often get "lost in hyperspace" when browsing complex hypermedia without additional navigational help [1].

When semantically formalised knowledge structures are being used, content typically becomes more fine grained and the content structure, i.e. the relations between objects gain importance. This stresses the need for user interfaces that facilitate navigating and authoring such structures without loosing orientation. Nowadays' ontology editors appear to be too complicated for every-day lightweight use. Outside the wiki and semantics world however, exist quite a number ot visual mapping techniques (like Mind-Maps, Concept Maps and others), that provide easy ways to rather intuitively structure fine grained bits of information.

iMapping is a new visual mapping approach based on Zooming User Interfaces, that tries to combine the strengths of several established mapping techniques and go beyond them. It is meant to support easy informal note taking as well as semi- and fully formalized knowledge engineering in the same powerful yet easy-to-use environment.

An iMap is like a large pin board, where wiki pages can be spatially arranged thus enabling users to gain visual overview over several wiki pages at once. Besides classical browsing, an iMapping-enabled wiki can be navigated by zooming through. We believe that the iMapping approach may well facilitate the use of wikis-especially when they go semantic.

## 2 Background

Since external knowledge repositories like wikis usually represent human knowledge and are maintained by humans, it appears sensible to make the UI as cognitively adequate as possible, to enhance the link between mental and external models. Unlike text, diagrammatic knowledge representations carry a structural analogy to the content they represent. In other words: A diagram's structure looks similar to the structure it is about. A map of Europe looks somehow like Europe from above. A flow-chart depicts the structure of a process. A text doesn't—It takes a longer way in the user's mind until it can be related to the user's mental model [2]. Enabling users to spatially arrange information items allows the creation of such diagrammatic depictions that give an intuitive overview over a subject matter. This is the very basis of iMapping.

### 2.1 Related Work

**Microcontent** Some Wikis, like e.g. SnipSnap [1], already allow including other wiki pages in a page so, instead of having to follow a link, the user can see the target page inline. This is a first step into the direction of microcontent ("content that conveys one primary idea or concept [and] is accessible through a single definitive URL" [2]), which is useful to avoid redundancies, because most pieces of information are relevant in different contexts. In the same way, pages can be nested into other pages in an iMap. This leads to a different conceptualisation of what a wiki page is: many pages will just be little snippets of text, while other pages will mainly *contain* such snippets or other resources, thus functioning as aggregators.

**Semantic Desktops and Wikis** One of the first Semantic Desktop systems, that lets the user freely specify semantic relations between typed information items on a topic maps basis, is **DeepaMehta** [3]. It provides a graph-based UI in a thin client. Once an item (or relation) has been specified (in a topic map),

[1] see `http://snipsnap.org`
[2] see `http://en.wikipedia.org/wiki/Microcontent`

DeepaMehta keeps it in a background repository on the server independent from whether they are still part of an actual topic map. This separation between the structural model and visual model makes sense, also for iMapping, because it allows multiple (visual) instances of an item to be used in different contexts or locationsmuch like hard links in a Unix file system. Some Semantic Wikis like **SemWiki** [4] work in a similar way, but browser based and without a graphical UI. Others, like SemperWiki [5], are made for local, personal use only and feature an optimized WYSIWYG UI [3].

**Visual Mapping Techniques** Visual mapping techniques are methods to graphically represent knowledge structures. Most of them have been developed as paper-based techniques for brainstorming, learning facilitation, outlining or to elicit knowledge structures. Some of them have proven to be very useful in Personal Knowledge Management, especially for tasks like gathering and structuring ideas and acquiring an overview on a certain domain. For an overview on visual mapping techniques, their cognitive psychological background and an evaluation of some existing techniques and tools, see [6]. In brief, all of these mapping techniques are quite helpful for some purposes but have constraint paradigms that make them useless for others.

**Mind-Maps** for example, provide an easy-to-understand tree-like structure useful for outlining a topic or sorting items. But it is suitable to depict the relational structure between items.

**Concept Maps** on the other hand have a graph-based structure that emphasizes these relations. But they are not as easy to handle, because explicitly specifying all these relations is too laborious e.g. for a fast gathering of keywords.

**"Spatial Hypertext"** is yet another approach. The basic idea is to view a self-contained hypertext (like a wiki is) from an overview perspective, drilling down to single pages (which tend towards microcontent). However the Spatial Hypertext paradigm expressly abandons the concept of explicitly stating relations between objects and uses spatial positioning as the basic structure. To fuzzily relate two objects, they are simply placed near to each other, but maybe not quite as near as to a third object. This allows for so-called "constructive ambiguity" [7] and is a very intuitive way to deal with vague relations and orders. While Spatial Hypertext in its pure form is not suitable to author formal knowledge structures like needed in semantic wikis, the general approach may well be used to augment them as a surface.

**Zooming User Interfaces** An early research prototype using a zooming approach was Pad and its successsor **Pad++**, both developed in Maryland [4]. It has been used in various applications and also as a web browser capable of showing the viewed web pages and their link-structure from a birds view. In a study

[3] For more information on Semantic Desktop systems in general, see `http://semanticdesktop.org`

[4] see `http://www.cs.umd.edu/hcil/pad++/`

where participants had to perform browsing tasks in order to answer some questions, subjects using Pad++ were 23% faster than those using Netscape [8]. This shows that using large zoomable information surfaces are well-suited hypertext front-ends. The work on Pad++ has later yielded its successors "Jazz" and finally **"Piccolo"**, a toolkit that supports the development of 2D structured graphics programs, in general, and Zoomable User Interfaces, in particular Piccolo [5].

A Semantic Desktop system whos UI is deeply based on zooming is **MentalSky** [6]. It uses machine-learning methods to semantically classify existing resources into clusters that can be browsed by zooming through and restructured with drag-and-drop interaction. MentalSky is currently in a prototype state of development. It strongly differs from an iMapping based semantic wiki in the respect, that it is constraint to managing external resources (like files, pictures, web-links, etc.) but is not made for authoring content neither in plain text nor in a formal way.

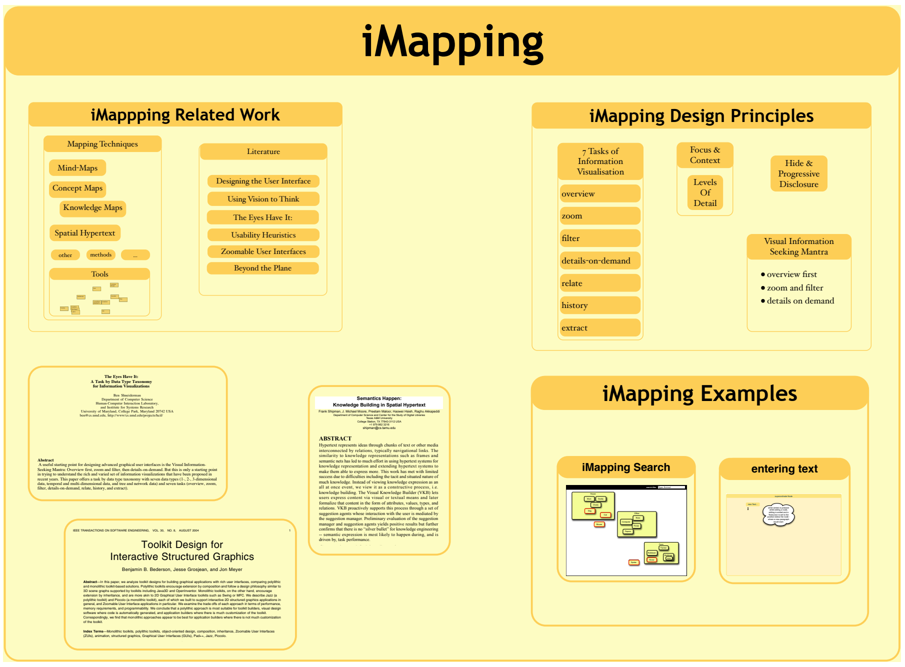


**Fig. 1.** An example iMap showing three expanded text pages and several sub-maps with collapsed items.

[5] see `http://www.cs.umd.edu/hcil/piccolo/`

[6] see `http://mentalsky.net/` and `http://cognitivetools.net/tiki-index.php?page=MentalSky`

## 3 Design

iMapping tries to combine the advantages of all the above approaches:

- basic wiki functionality (collaborative editing, easy linking, backlinks etc.)
- visual knowledge representations with structural analogy to content
- easy hierarchical overall topology
- facility for graph-based relation mapping
- support for formal semantic statements
- allowing constructive ambiguity
- provide overview by integrating context and detail through zooming

**Basic Hierarchy** The basis of the iMapping paradigm is a large two-dimensional surface, where items can be freely placed. In a wiki context, these items mainly correspond to wiki pages. Because these items can contain other items, it is encouraged to use microcontent rather than long unstructured texts. Whereas in Mind-Maps or other tree-like diagrams the lower hierarchies branch towards the outside from a central point, in an iMap hierarchy goes down into deeply nested nodes that can be zoomed into (see Figure 1). Like explained above, there can be multiple visual instances of one and the same information object, because it may be relevant in different contexts.

**Other Content** Instead of a wiki (text-) page, a node could also contain things like a picture or other structured objects. It can basically be seen as an inline link to any resource for which a display method is known. Even inter-wiki or other remote content could be included inline like that.

**Levels of Detail** Because some information objects (like most text-pages) are rather hard to recognise when they are scaled down to thumbnail size, the nodes should have at least two possible states: open and closed, which could also be seen as expanded / collapsed or being outside / inside the node. Switching between these states is done either manually per click or can take place automatically, depending on how large the object is displayed. This method is also sometimes referred to as “semantic zooming”. A wiki page could be represented by its name in collapsed state and with its content in expanded mode. A more structured article could show his title only from a distance, when zoomed larger also some additional information like authors and date and when zoomed to reasonable size, fade over to the full content. Structuring content in the ABCDE format[7], facilitates such semantic zooming.

[7] see `http://wiki.ontoworld.org/wiki/ABCDEF`

**Link Structure** On the one hand, giving an overview to the structure and relations between information items is one of the main benefits of the iMapping approach. On the other hand, just drawing arrows for every link or even every semantic relation and backlinks to any other item would result in a complete mess sometimes referred to as the “spaghetti syndrome”. The idea in iMapping is, to not show any relations by default, and only make them visible on demand (see Figure 2). This could be a subtle interaction like mouse-over or something more explicitdepending on user settings or a mode.

Not only naming or even typing links but graphically drawing them between objects in a concept map (node-and-link) style would go even further beyond common wiki functionality. However it is a common and well-evaluated technique that is useful for depicting and authoring relations between information items. Such links have of course to stay permanently visible. In the same way, it is possible to semantically interrelate items by simply drawing links between them, which can than be typed. If this is done using auto-completion, reuse of existing relation types is fostered.

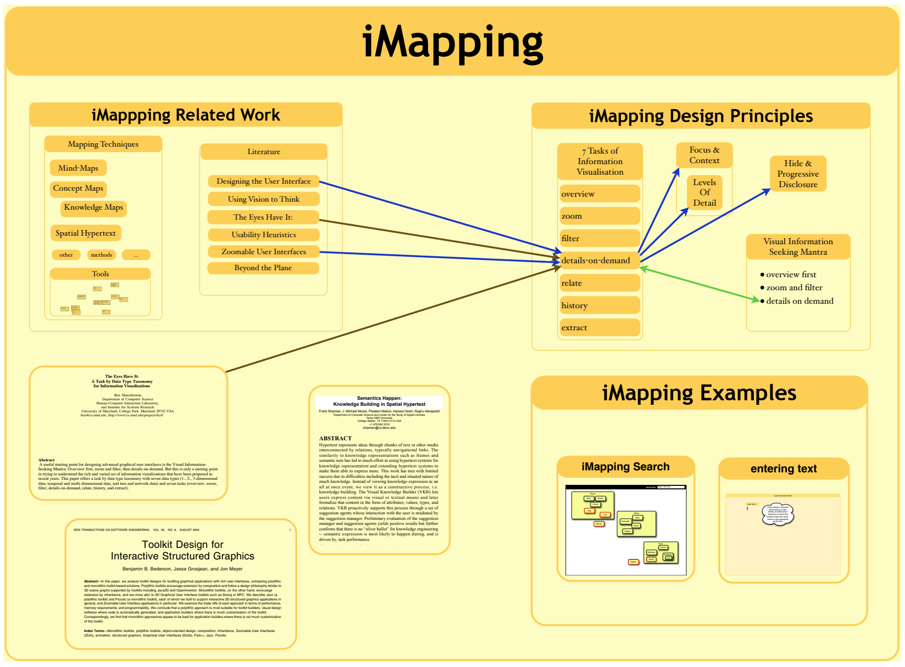


**Fig. 2.** The same iMap but with link-structure of one item made visible.

To sum up, this makes three structurally different ways of interrelating items in an iMapping environment:

- wiki-style text-links
  (linking from a particular text position to another item)
- nesting items into another
  (including the link target inline at a specified position)
- linking on an item level
  (stating a relation between two objects)

Each of these can be mere navigational links or carry formal semantics, if specified.

## 4 Discussion

Whether the iMapping approach will be successful, user studies and time will have to tell. It is a concept so far. A first prototype environment is under development and might be available by mid 2006. While our implementation is based on a java client to take advantage of the Piccolo framework (s. above), a flash- or SVG+AJAX-based version would allow browser-based usage, which would come closer to common wiki use.

Also, since the iMapping approach was initially designed for personal use, there might be unforeseen difficulties when used in a collaborative setting, like most wikis are. For example, there could be dissent on how items should be spatially arranged. But hopefully, like it is common in wiki culture, over time layouts will converge to a structure that finds consensus. Another approach would be to use personal profiles to let users make their personalised spatial arrangements of the content. The better the content and its structure represented using defined semantics, the easier it is to separate it from its visual appearance and to syndicate it to other applications.

## 5 Outlook

Wikis have started as very simple content management systems and many engines have grown immensely feature-rich by now. The step to semantic wikis is very promising and could give the realisation of the Semantic Web a significant boost. But it surely doesnt make these wikis easier to use. Focussing on user interaction and cognitive ergonomics will be an important point, if semantic wikis are to become widely used whether collaboratively or for personal knowledge management. In the Open Source Social-Semantic-Desktop Project Nepomuk[8], a first iMapping Wiki is being developed, and anticipated to become available during 2007. It will then become part of a more comprehensive knowledge workbench integrating Semantic Desktop functionalities like application integration, and semantic search with a distributed p2p-based collaboration environment.

[8] see `http://Nepomuk.semanticdesktop.org`

**Acknowledgments:**

Research reported in this paper has been partially financed by the EU in the Social Semantic Desktop project NEPOMUK (IST-FP6-027705). I would also like to thank Benjamin Heitman for fruitful discussions and last-minute technical assistance.

## References

1. Tergan, S.O.: Hypertext und Multimedia: Konzeption, Lernmöglichkeiten, Lernprobleme und Perspektiven. In: Information und Lernen mit Multimedia und Internet. third, completely revised edn. Beltz, PVU (2002)
2. Schnotz, W.: Wissenserwerb mit Texten, Bildern und Diagrammen. In: Information und Lernen mit Multimedia und Internet. third, completely revised edn. Beltz, PVU, Weinheim (2002)
3. Richter, J., Völkel, M., Haller, H.: Deepamehta - a semantic desktop. In Decker, S., Park, J., Quan, D., Sauermann, L., eds.: Proceedings of the 1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference (Galway, Ireland). Volume 175., CEUR-WS (2005)
4. Völkel, M., Oren, E.: Personal knowledge management with semantic wikis (2006) Paper submitted to ESWC2006, available at `http://semwiki.ontoware.org/`.
5. Oren, E.: Semperwiki: a semantic personal wiki. In: Proceedings of the 1st Workshop on The Semantic Desktop, Galway, Irland. (2005)
6. Haller, H.: Mappingverfahren zur Wissensorganisation (2003) Knowledge Board Europe. Availlable online at `http://heikohaller.de/literatur/diplomarbeit/`.
7. Shipman, F., Marshall, C.: Spatial hypertext: An alternative to navigational and semantic links. ACM Comp. Surveys **31** (1999)
8. Bederson, B.B., Hollan, J.D., Stewart, J., Rogers, D., Vick, D.: A zooming web browser. Human Factors in Web Development (1998)

# The ABCDE Format
## Enabling Semantic Conference Proceedings

Anita de Waard[1,2] and Gerard Tel[2]

[1] Advanced Technology Group
Elsevier, The Netherlands
anita@cs.uu.nl
[2] Department of Computer Science
Utrecht University, The Netherlands
gerard@cs.uu.nl

## Core Matter

– We believe that the best way to present a narrative to a computer is to let the author explicitly create a rich semantic structure for the article during writing.
– We propose an open-standard, widely (re)useable format, the ABCDE format for proceedings and workshop contributions that can be easily mined, integrated and consumed by semantic browsers and wikis.
– There need not be an abstract in an ABCDE document - instead, the author denotes core sentences within the B,C and D sections, which are compiled through a macro to form a structured abstract.
– We believe a LaTeX stylesheet provides a suitable input format for providing authors with a semantic structure to work from.
– We provide the `abcde.sty` LaTeX file as an appendix to this paper.
– Macros are provided to specify Dublin Core Elements, and to print a list of those that are specified.
– Our section division into Background, Contribution, and Discussion is backed by a number of emperical studies.
– We aim to work on different incarnations of this format and open it up to modification and development.

## b1 Introduction

The main problem with automatically extracting information from scientific articles is that the genre of the scientific publication has developed to be an indivisible information unit [1]. The scientific paper is a self-contained narrative, created anew in each iteration, with specific genre characteristics that minimize the potential of identification, content reuse and knowledge integration. All this rhetorical freedom comes at the expense of usability in a computer-centered environment. The linear narrative was fine when we still read and wrote on paper, but the digital environment in which scientists live and work today calls for a new fundamental unit of communication.

We believe that the best way to present a narrative to a computer is to let the author explicitly create a rich semantic structure for the article during writing. As conceptual structures become the central bearer of information, a set of structured documents can be integrated to form a knowledge network, or structured package of related knowledge regarding a topic [2]. This can be seen to form an incarnation of the intelligent data ideal, which the Semantic Web is meant to enable[3]. The purpose of our work is to examine such a new form of structured publications. Semantic Browsers such as PiggyBank[4] and semantic collaborative authoring tools such as Semantic Wiki's – as presented at this workshop[5] – are paving the infrastructural road for distributed, semantic communities to communicate. Hopefully, the ABCDE format can be a useful vehicle on this road.

*Article Outline.* This paper is organised as follows: in Section c2, the ABCDE format is described and motivated; in Section c3, the annotation and rendering of ABCDE articles in LaTeX is described. In Section d4, we discuss related work and in Section d5 some next steps.

The section numbers are consecutive, but are prefixed by a modifier: b, c, or d. These are meant as a visual cue to reflect whether the section is a part of the Background, Contribution or Discussion content of the article (described below). The reason for adding this modifier is to help the reader, but also the author, to realise which part is which - if desired, the stylesheet can be modified to make this formatting aspect invisible to humans, and only visible to computers.

## c2 The ABCDE Format

We propose an open-standard, widely (re)useable format, the ABCDE format for proceedings and workshop contributions that can be easily mined, integrated and consumed by semantic browsers and wikis. It is characterised by marking the following elements in a document:

**Annotations:** Each record contains a set of metadata that follows the Dublin Core standard[6]. This metadata is included as a part of each paper, to alleviate the annoying experience that one encounters when an article is found floating in cyberspace, without a date or any bibliographic reference information. In this sense, the DC qualifiers act as a passport that identifies the paper's date and place of birth, for future readers. Minimally required fields are Title, Creator, Identifier and Date. They can be rendered as a part of the text (see below) or left only as mark up, and not printed.

[3] "The Semantic Web is not so much about intelligent agents, but more about stupid agents and intelligent data", Berners-Lee at WWW4, Boston, 1995, personal record.
[4] http://simile.mit.edu/piggy-bank/
[5] http://www.semwiki.org/
[6] http://dublincore.org/documents/dces/

**Background, Contribution, Discussion:** The material in the main body of text is classified in one of three types:

- *Background*, describing the positioning of the research, ongoing issues and the central research question;
- *Contribution*, describing the work the authors have done: any concrete things created, programmed, or investigated;
- *Discussion*, contains a discussion of the work done, comparison with other work, and implications and next steps.

This classification must be made explicit in the metadata of the article – for details, see Section c3 on markup below.

**Entities:** Throughout the text, entities such as references, personal names, project websites, etc. are identified inside LaTeX as footnotes or references. The entities can be mined and turned into RDF, where the triple contains the section of the paper containing the entity, the entity URI, and the type of link (reference, person, project).
Identifying the contribution type will increase the quality of the property that can be inferred. For example, the mention of a project website in the Contribution probably means that the project is one of the core components of the system described in the paper. On the other hand, a project website mentioned in the Discussion probably means it is described as a Related Work.

**Core sentences as abstract:** There need not be an abstract in an ABCDE document - instead, the author denotes core sentences within the B,C and D sections, which are compiled through a macro to form a structured abstract. Upon retrieval or rendering of the article, these can be extracted to form a structured abstract of the article. This allows the author to create and modify statements summarising the article only once, and prevents that an abstract misrepresents the content of the article. This can easily happen when sections are deleted from the content, but left in the abstract, as was shown in [3] A Core summary also enables the implementation of a structured, hyperlinked abstract, where one can jump directly to the relevant part of the article from the sentence of interest.

## c3 How to semantically mark your paper

We believe a LaTeX stylesheet provides a suitable input format for providing authors with a semantic structure to work from. The `abcde.sty` style file implements the ABCDE structure for documents typeset with Springer's LaTeX `llncs.cls` class file, very commonly used for proceedings publications in computer science. We provide the `abcde.sty` LaTeX file as an appendix to this paper.

*The LaTeX style sheet.* To create a consistent layout for its proceedings, Springer makes available to authors and editors its class file `llncs.cls`[7]. It provides

[7] http://www.springer.com/sgw/cda/frontpage/0,11855,5-164-2-72376-0,00.html

the common sectioning commands `\section{..}` through `\paragraph{..}` and some theorem-like environments (`\begin{theorem} ... \end{theorem}`). To authors, the `llncs` class ressembles the common `article` class, but it is richer in marking the contribution with semantic metadata. Specifically, besides `\title` and `\author` commands, there are a `\titlerunning`, `\subtitle`, `\email`, and `\institute` commands.

If your paper was prepared with the `llncs` LaTeX package, to semantically mark it with the ABCDE format:

- Store the style file `abcde.sty` in the same folder as the paper;
- Add the line `\usepackage{abcde}` to your preamble.

Keep in mind that the purpose of semantic marking is mainly to produce meta information that goes with the document. We have chosen to render some of the markup as visual elements as well (for example, by prefixing the section numbers with b, c, or d). This was done for illustration purposes, and the style sheet can be modified to make the proposed semantic marking invisible in the printed result.

The command `\tableofcontents` and the environment `abstract` remain available, but you may choose to have the new command `\listofcore` instead (or in addition). This will produce a list of sentences that you have marked as *core* in the paper.

## c3.1 Annotations

Macros are provided to specify Dublin Core Elements, and to print a list of those that are specified. A Dublin Core element is characterized by a *name* and a *value*; it can be specified in the contribution by the command `\dublincore{..}{..}` with the name and value as first and second argument. For example, you can place `\dublincore {publisher} {Creative Commons}` anywhere in your document; the preamble would be the most logical place. (For more about Creative Commons, see http://creativecommons.org/about/licenses/.) On the other hand, `\dublincore{subject}{Dublin Core}` would be logically placed at the place where you discuss Dublin Core elements, so that if you decide to remove some material from your paper, the annotation is removed as well.

The annotations can be just used as metadata without being displayed, but a list can be printed anywhere in the document using the command `\annotations`. For this document, the result would be:

**DC Annotations**
**creator:** Anita de Waard , Gerard Tel
**title:** The ABCDE Format
**date:** May 4, 2006
**subject:** ABCDE Format
**subject:** The `llncs.sty` LaTeX style
**publisher:** Creative Commons
**subject:** Dublin Core

Some metadata already available in LaTeX-typeset documents is automatically interpreted as a DC element: specifically, the elements **creator**, **title**, and **date** will be registered as DC elements if they are provided in the preamble with the usual commands.

## c3.2 Background, Contribution, Discussion, and Core

The commands `\background`, `\contribution`, and `\discussion` declare that the material following it is the Background, the Contribution, or the Discussion of your document. The semantic marking commands do not replace sectioning commands, so you still need to name the sections in your document.

The simplest documents have these three parts consecutively, so they will have just one `\background` command at the beginning, one `\contribution` command after one or two sections, and one `\discussion` command near the end. The `abcde` package allows to switch between the three types more flexibly. The declaration implied by one of the three commands remains valid until the next `\background`, `\contribution`, or `\discussion` command. If your document contains material that does not fall in one of the three types, precede it with the `\unbcd` command.

Important statements can be marked as *core sentences* using the `\core{..}` command; these sentences can be harvested to give an overview of the content of the paper, with the possibility to jump directly to the relevant part of the paper. Using core sentences can replace an abstract, as they become part of the "Core Matter" list produced by the `\listofcore` command; this command was used on the first page of this paper. Of course, authors have the possibility of writing their abstract instead, and have the core sentences only as metadata pointers to their work.

Markup as core does not change the marked sentence visibly. We found that sometimes, sentences do not read well when taken out of their context; this may happen because of anaphors ("This is a result of ...") or because of a more complicated entanglement with surrounding sentences. If the phrasing of the sentence should differ between the text and the Core Matter list, use the form `\core[Sentence1.]{Sentence2.}` to print `Sentence1.` in the document and have `Sentence2.` in the Core Matter. For example, "She worked in Africa." can be put in the core matter as "Streep worked in Africa." using

`\core[She worked in Africa.]{Streep worked in Africa.}`.

The `\listofcore` command with the optional argument `[1]` will restrict the list to core sentences from the Contribution part of your document (and `[3]` will extend it to also contain core sentences outside of BCD-marked parts).

## c3.3 Entities

Both the Dublin Core elements (Annotation) and the (embedded) Elements, such as project websites, references, and personal names, can be extracted to port to an RDF-enabled system.

The notion of entities was described in Section c2; we are looking to expand these to the emerging standard RDF–based formats in the future, in collaboration with Semantic Wiki groups.

### c3.4 Shortcomings in the package

It is not possible to have complicated macros in a core sentence. The package was not tested against the various class options of `llncs.cls`.

The purpose of semantic marking is harvesting meta information, but the current package also produces a visible effect of the markings. The should be a possibility to switch the visual effects on and off using a style option `[draft]`; while writing draft versions, the author can keep an eye one the markings he already made, but in the final version the semantic marking would only be harvested, not shown.

## d4 Related work

There are many fields of research which offer insights, and important contributions, concerning the structuring and annotation of scientific texts. In an attempt to position the ABCDE proposal within this vast landscape, we distinguish two different dimensions of markup. First the elements marked up, which can be *entities* or *document structure* and second, the time of markup, which can be *during authoring* or *post-publication.* Combined, these describe four categories of work, which are consecutively discussed.

*Annotation phase: post-publication; marked up: entities* Leaving aside entity extraction techniques[8], an interesting body of work from the Open University revolves around the creation of a "sensemaking environment" which allows readers to manipulate, order and annotate documents. With ClaiMaker [4] and ClaimSpotter [5], they aim to create "a system that explicitly model[s] the rhetorical relations between claims in related papers". Readers can create claim-and-relationship triples according to their ontology of rhetorical relations, to make better sense of the corpus of scientific documents. The triples are identified outside the documents themselves, to improve understanding.

*Annotation phase: during authoring; marked up: entities* There are several Semantic Wiki and blogging initiatives which propose to provide semantic annotations and allow for distributed access using marked up entities. For instance, Karger and Quan [6] propose to use the innate semantics of messages and blogs to generate semantic markup and utilise it in collaborative (decision-making) systems. Mika and Klein [7] transform BibTex files into RDF, and use it to connect and disseminate bibliographic information of a research group. And Oren

[8] Technically, the huge volumes of work in entity identification and text mining belong in this category, but since the field is vast and not directly related to our research, we will omit a discussion here.

et. al [8] define six dimensions of annotation context, which helps create a faceted browsing interface to improve navigation through their Wiki environment.

In all three intiatives, the author adds (nominal) markup to improve the (RDF-based) metadata of the article.

*Annotation phase: post-publication; marked up: text structure* Simone Teufel applies a 'rhetorically defined annotation scheme', consisting of seven categories which model prototypical academic argumentation [9]. She first lets a human annotator apply one of seven 'rhetorical roles' to specific elements of a text, and uses this input to train an automatic annotator, which is then used for automated abstract generation.

Noriko Kando [10] defines a fine-grained 'text-level structure', and manually annotates a corpus of (Japanese) articles on HIV/AIDS with this structure. He finds a improved results for searching, passage extraction and browsing tasks.

Bayerl [11] adds three types of markup to a corpus of articles in psychology and linguistics, and then compares the results of the markup. Her three types are structural, which she bases on Kando's schema (above); thematic, based on the Van Dijk Macrostructure concept [12]; and rhetorical, for which she uses Rhetorical Structure Theory [13].

*Annotation phase: during authoring; marked up: text structure* In contrast, there is much less literature on building systems that allow authors of scientific publications to add markup while creating the text. Quite possibly, this is because this has traditionally been the domain of the publisher, whose methods are proprietary and not under scientific investigation.

Our section division into Background, Contribution, and Discussion is backed by a number of emperical studies.

Kando [10] did an analysis of 40 writing manuals, and came up with a text–level structure where the main headings are Problems, Evidence and Answers. Harmse and Kircz [14] performed a thorough investigation of a corpus of documents in atomic physics, and derived a set of modules, of which the three main ones are Position, Results and Interpretation.

Many journal style guidelines contain a similar division. For example, the American Institute of Physics[9] recommends using Introduction, Main Body and Conclusion as three essential parts of the paper. In the life sciences, papers are often explicitly structured in a similar way: for example, the journal Cell[10] proscribes the sections Introduction, Results, Discussion (and Experimental Procedures) and BMC Cell Biology[11] requires Background, Results, Discussion, Conclusions (and Methods). All of these tripartite divisions correspond quite well to our sections Background, Contribution and Discussion.

The idea of letting authors create markup was motivated in part by the work of Kircz and Harmsze [14], who identified a set of modular elements for a

[9] http://www.aip.org/pubservs/style/4thed
[10] http://www.cell.com/misc/page?page=authors
[11] http://www.biomedcentral.com/bmccellbiol/ifora/

corpus of papers in physics, and created authoring instructions for this modular layout. Work done at Elsevier on the online encyclopedia XPharm[12] built on these explorations, by offering a modular authoring environment, using Word templates.

*Structured abstracts* Structured abstracts are used in various settings: in medicine, they are quite common (for instance JAMA[13] and the BMJ use them) and are even a topic of study in themselves [15] — since the quality of abstracts is sometimes found to be seriously deficient [3].

Van der Tol [16] has proposed to use structured abstracts as a navigational tool, which could correspond to the 'Core matter' approach, when expanded with the right interface.

## d5 Next steps

We aim to work on different incarnations of this format and open it up to modification and development. The point is to offer a flexible structure that can live on semantic environments such as Semantic Wikis and browsers, such as Haystack[14] or Piggybank[15]. Ideally, ABCDE papers should be much easier to mine and integrate. By adding semantic markup of knowledge elements, discovery and integration of information at a structural level is improved.

It is our aim to manually mark up (a subset of) the papers presented to the SemWiki[16] workshop in LaTeX with the `abcde.sty` stylesheet, and open it up for testing before, during and after the workshop. We actively seek collaboration with groups working on Semantic Wikis to see if the format is indeed suitable for transformation to RDF, and how the the metadata can be optimally mined, stored and visualized. The ideal is to narrow the gap between publications and annotations, between doing research and talking about it. We mean to practice what we preach, and will attempt to use the ABCDE format for all relevant conference submissions. Hopefully, with concomitant RDF database and interface work, we can create a 'tipping point'[17] for the implementation of this format, and contribute to the creation of a much richer set of conference proceedings in computer science.

An example of possible developments would include the creation of a conference program, consisting of 'core-contribution' sentences, that link to contributions, as a quick way to scroll around the papers presented. Another example would be to mine all the links to a project website and connect them to the website, linked to the paragraph where the project was mentioned. This would allow the visualization of related projects, topics and co-authors.

[12] http://www.xpharm.com
[13] http://jama.ama-assn.org/
[14] http://haystack.lcs.mit.edu/
[15] http://simile.mit.edu/piggy-bank/
[16] http://semwiki.org
[17] http://www.gladwell.com/tippingpoint/index.html

For example, in the OpenAcademia.org project [7], BibTex references are turned into RDF to allow a connected set of bibliographic references utilising an Open Source extension of RSS called BuRST[18]. This rendering could be used to mine the references of an ABCDE paper, as well — and include the section of the text where the reference was made, again enhancing the quality of inferrable information. Again, the division between a paper and a discussion of a reference begins to blur, and the publication itself can become a (set of) object(s) on a Semantic Web/Wiki.

Our further work will involve the development of a more detailed model of scientific publications, and looking at the construction of meaning within scientific documents through argumentation analysis and understanding of discourse structure. The tension between the arguments or moves and the narrative of the document as a whole poses an interesting topic of study in terms of both knowledge modeling and rhetoric/discourse studies. Hopefully, it can also help create a more legible way to publish science for computer-assisted humans, and human-assisted computers.

## 6 Acknowledgement

We wish to thank Simon Pepping of Elsevier for helping think through the ABCDE format.

## References

1. Bazerman, C.: Shaping written knowledge: The genre and activity of the experimental article in science. Madison: University of Wisconsin Press (1988)
2. de Waard, A.: Science publishing and the semantic web, or: Why are you reading this on paper? European Conference on the Semantic Web 2005, Industry Forum, Alain Léger ed. (2005)
3. Pitkin, R.M., Branagan, M.A., Burmeister, L.F.: Accuracy of data in abstracts of published research articles. JAMA **281** (1999) 1110–1111
4. Li, G., Uren, V., Motta, E., Buckingham-Shum, S., Domingue, J.: Claimaker: Weaving a semantic web of research papers (2002)
5. Shum, S.B., Domingue, J., Motta, E.: Scholarly discourse as computable structure. In: OHS-6/SC-2. (2000) 120–128
6. Karger, D.R., Quan, D.: What would it mean to blog on the semantic web? Journal of Web Semantics **3**(2) (2005)
7. P. Mika, M. Klein, R.S.: Semantics-based publication management using RSS and FOAF. In: Proceedings, Semantic Wiki 2006 (Submitted). (2006)
8. Oren, E., Delbru, R., Möller, K., Völkel, M., Handschuh, S.: Annotation and navigation in semantic wikis. In: SemWiki (ESWC). (2006) Submitted.
9. Teufel, S., Moens, M.: Discourse-level argumentation in scientific articles: Human and automatic annotation. In Walker, M., ed.: Towards Standards and Tools for Discourse Tagging: Proceedings of the Workshop. Association for Computational Linguistics, Somerset, New Jersey (1999) 84–93

[18] http://www.cs.vu.nl/ pmika/research/burst/BuRST.html

10. Kando, N.: Text-level structure of research papers: Implications for text-based imformation processing systems (1997)
11. Bayerl, P.S.: Methods for the semantic analysis of document markup (2003)
12. Dijk, T.: Macrostructures: An interdisciplinary study of global structures in discourse, interaction, and cognition. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1980)
13. Mann, W.C., Thompson, S.A.: Rhetorical structure theory: Toward a functional theory of text organization. Text **8**(3) (1998) 243–281
14. Kircz, J., Harmsze, F.: Modular scenarios in the electronic age. In: CS-Report 00-20. Proceedings Conferentie Informatiewetenschap 2000. De Doelen Utrecht, 5 april 2000. (2000)
15. Wong, H., Truong, D., Mahamed, A., Davidian, C., Rana, Z., Einarson, T.: Quality of structured abstracts of original research articles in the british medical journal, the canadian medical association journal and the journal of the american medical association: a 10-year follow-up study. Curr Med Res Opin. **21**(4) (2005) 467–73
16. van der Tol, M.: Abstracts as orientation tools in a modular environment. Document Design **2**(1) (2001) 76–88

## A The `abcde.sty` style sheet

```
%
% File: abcde.sty
% Created 16/03/2006 by Gerard Tel
% Defines semantic annotations following the ABCDE proposal
% for documents already in llncs format.
%
%
% A is for Annotations
% An annotation in the Dublin Core format has two characteristics:
% the NAME of the element and its VALUE.
% A possible third argument is the SCHEME describing the VALUE format.
% Make a Dublin Core element using \dublincore[scheme]{name}{value}
% to write name and value to file dce:
\newcommand\dublincore[3][DEFAULT SCHEME]{
\addcontentsline{dce}{dcelt}{{#2}{#3}}}
%
% The DC annotations can, but need not, be printed in the text:
\def\annotationsname{DC Annotations}
% How to print a name/value combination:
\def\core@nameval#1#2{{\bf #1:} #2}
\def\@dceltline#1#2{% #1 = {name}{value}, #2 = pageno
  \par\noindent\core@nameval#1 \par}
\def\l@dcelt{\@dceltline}
% Print the core sentences with \annotations
\newcommand\annotations{%
  {\noindent\bf\annotationsname}\par
  \@starttoc{dce}}
```

```
%
% Harvesting the annotations
% Grab as much as you can from \maketitle:
\let\orig@maketitle=\maketitle
\renewcommand\maketitle{
  % within the DC element, \and and \inst#1 have different meaning
  \let\orig@and=\and\def\and{, }
  \let\orig@inst=\inst\def\inst##1{}
  \dublincore{creator}{\@author}
  \def\and{\orig@and}\def\inst{\orig@inst}
  \dublincore{title}{\@title}
  \dublincore{date}{\@date}
\orig@maketitle}

% There are four BCD-types.  Initially it is bcdu (undefined)
\newcommand{\bcd@type}{bcdu}
% The \background, \contribution, and \discussion commands will
% 1. Change the section numbering by prefixing a letter
% 2. Change the abs entries by changing \bcd@type
\newcommand{\bcd@swap}[2]{
  \def\thesection{#1\arabic{section}}
  \def\thefigure{#1\arabic{figure}}
  \def\bcd@type{#2}}
\newcommand{\background}{\bcd@swap{b}{back}}
\newcommand{\contribution}{\bcd@swap{c}{cont}}
\newcommand{\discussion}{\bcd@swap{d}{disc}}
\newcommand{\unbcd}{\bcd@swap{}{bcdu}}
%
%
% The command \core{TEXT} will print TEXT and save it in the file
% basename.abs as a contentsline
% Two args, first is optional with default second
% Print argument normally in text:
\newcommand{\core}[2][\undefined]{
\ifx\undefined#1#2\else#1\fi
\addcontentsline{abs}{\bcd@type}{#2}}
\def\listofcorename{Core Matter}
\def\@coreline#1#2#3{% #1: importance, #2: text, #3: Page no
  \ifnum#1>\c@coredepth
  \else \item#2\vskip 3\p@\fi}
\def\l@bcdu{\@coreline{3}} % Relatively unimportant
\def\l@back{\@coreline{2}} % Moderately important
\def\l@cont{\@coreline{1}} % Important
\def\l@disc{\@coreline{2}} % Moderately important
% Print the core sentences with \listofcore[i],
% Where i=1 prints only contribution core, 2 prints B&D as
% well, and i=3 prints "unlabeled" core.
\newcommand\listofcore[1][2]{%
  \def\c@coredepth{#1}
  \@restonecolfalse\if@twocolumn\@restonecoltrue\onecolumn\fi
  \section*{\listofcorename\@mkboth{{\listofcorename}}{{\listofcorename}}}
  \begin{itemize}\@starttoc{abs}\end{itemize}
  \if@restonecol\twocolumn\fi}
```

# Learning with Semantic Wikis

Sebastian Schaffert, Diana Bischof, Tobias Bürger, Andreas Gruber, Wolf Hilzensauer, and Sandra Schaffert

Salzburg Research Forschungsgesellschaft
Jakob Haringer Str. 5/II, A-5020 Salzburg, Austria

**Abstract.** The knowledge society requires life-long learning and flexible learning environments that allow learners to learn whenever they have time, whereever they are, and according to their own needs and background knowledge. In this article, we investigate how Semantic Wikis – a combination of Wiki and Semantic Web technology – can support learners in such flexible learning environments. We first summarise common features of Wikis and Semantic Wikis and then describe different aspects of Semantic Wikis for learning. We also introduce our Semantic Wiki system called IkeWiki and show why it is particularly promising as a learning tool.

## 1 Introduction

Recently, many different Semantic Wikis have been described in the literature (cf. e.g. *Platypus* [1], *Semantic MediaWiki* [2], *SemWiki* [3], *SemperWiki* [4], *WikSAR* [5], *IkeWiki* [6,7]). All or most of these articles target knowledge management or aim at enhancing the Wiki "experience" by improved navigation, browsing, and searching.
In this article, we aim to investigate another area where Semantic Wikis might play an important role: learning. A possible separation from knowledge management is that whereas knowledge management focusses on the *result*, learning focusses on the *process* that leads to the result.

In our dynamic society, learning plays an increasingly important role, and the way learning is perceived has changed significantly. Whereas traditionally learning was limited to school and university, it is now seen as a life-long process (*"life-long learning"*) requiring learners to constantly update and adapt their knowledge to new developments [8]. Whereas school and university teaching often used to be rather transfer of information from the teacher to the students without much student

**Fig. 1.** "Schulmeister von Esslingen": traditional teacher-centred learning

interaction (Fig. 1), learning in modern learning theory is considered an active process where learners participate and teachers are merely coaches in the learners' learning process. And whereas individual skills used to be in the centre of traditional learning processes, cooperative and group learning are becoming more and more important as problems are usually solved by teams and not by individuals.

Such learning requires more flexible learning environments where learners can develop their skills as needed ("on-demand learning") and when they have time ("just-in-time learning"). This, in turn, requires that content can be accessed, authored, reused, and combined easily. Social Software (i.e. *Weblogs*, *Wikis*, *ePortfolios*, *Instant Messaging*) and Semantic Web technology could play an important role in such learning environments. Where Social Software gives users freedom to choose their own processes and supports the collaboration of people anytime, anywhere, Semantic Web technology gives the possibility to structure information for easy retrieval, reuse, and exchange between different systems and tools.

In this article, we focus on a very specific technology that combines Social Software and the Semantic Web: Semantic Wikis. In Section 2, we sketch the development from traditional Wikis to Semantic Wikis. Section 3 summarises the current usage of Wikis in learning and investigates how Semantic Wikis can be beneficial. We then introduce our own Semantic Wiki called *IkeWiki*, which we believe is particularly well suited for learning (Section 4). We conclude with an overview over related work (Section 5) and perspectives for further research (Section 6).

## 2 From Wikis to Semantic Wikis

### 2.1 Traditional Wiki Systems

"Wiki" is the short form for "WikiWikiWeb" and is derived from the Hawaiian expression "wiki wiki" meaning "fast" or "quick". A Wiki is essentially a collection of Web sites connected via hyperlinks. While there is a wide range of Wiki systems available (e.g. MediaWiki, MoinMoin, TWiki) with different purposes and audiences, all of them share the following common properties:

**Editing via Browser.** Content is usually edited via a simple browser interface that can be used without installing any additional (expensive) software. This makes editing simple and allows to modify pages from everywhere in the world with only minimal technical requirements. As a consequence, content creators can access and update the Wiki from wherever they are, e.g. at work, at home, at conferences, nowadays even while travelling.

**Simplified Wiki Syntax.** Content is usually expressed in a simplified hypertext format ("Wiki syntax") that is much easier to use for non-technical users than e.g. HTML. Formatting thus does not require knowledge of HTML.

**Rollback Mechanism.** Changes to the content of a Wiki are *versioned* each time they are stored, i.e. previous versions of pages are kept. This allows to revert to earlier versions of a page e.g. in case important parts have been accidentally deleted or undesirable modifications have been made by someone else. Also, most Wiki systems allow to compare two versions of a page, making it possible to identify changes between edits quickly.

**Strong Linking.** Pages in a Wiki are usually strongly linked with each other using hyperlinks. The reason for this is that the simplified Wiki syntax makes it very easy to define a link to another page in the Wiki. For example, in many Wikis a link is defined by enclosing a word in square brackets, or by using a so-called "CamelCase" where a word contains several upper-case letters. Links to non-existing pages are usually rendered in a different colour. If a user clicks on such a link, the system redirects him to a view where he can create the non-existing page. In many Wikis, this is even the only way to create a page.

Links in a Wiki are the most important tool for navigation. Therefore, many systems allow not only to follow links in the direction they are defined but also in reverse direction ("back-links").

**Unrestricted Access.** In most Wiki systems, access is completely unrestricted – i.e. anyone can correct, modify, complete, or even delete anything. While this might seem strange, and even dangerous, from a traditional perspective, practice shows that the system works: on the one hand, ill-meaning users are rather rare; on the other hand, all changes can easily be undone using the rollback mechanism. Note that some Wikis still allow to apply further access restrictions using users and groups as found in traditional content management systems.

**Collaborative Editing.** The above-mentioned properties combined make Wikis an ideal tool for collaborative editing. As soon as someone creates content, others can contribute to it, extend it, correct it, etc. Many Wiki systems provide further support for collaborative editing, e.g. by means of discussion forums, summaries of changes, and list of last updates.

Unlike other groupware or content management tools, a Wiki gives users almost complete freedom over the content development process without rigid workflow, access restrictions, or predefined structures. Users need not adapt their practice to the "dictate of the system", but can allow their own practice to define the structure. This is important, because different domains often have – or even require – different kinds of workflow.

As a recent survey on the popular technology site *Slashdot* showed[1], Wiki systems are currently used for a wide variety of purposes, including:

– *encyclopaedia systems:* collect information in a certain area (e.g. Wikitravel) or unrestricted (e.g. Wikipedia) in a community effort with contributions from a wide range of users

[1] http://ask.slashdot.org/article.pl?sid=06/01/21/1958244

– *software development:* collaboratively create documentation, collect ideas, track bugs; most of today's high-profile Open Source projects (e.g. Apache, Mozilla, OpenOffice) use Wikis for coordination
– *project knowledge management:* project tracking, brainstorming and exchange of ideas, coordination of activities, agenda tool for collecting topics of meetings, project notes repository, knowledge base, staff directory
– *personal knowledge management:* sketchpad to collect and elaborate personal ideas, addresses, dates, tasks, bookmarks, etc. [3]
– *collaborative writing:* authors work collectively on a writing (short story, novel, etc.) which is immediately accessible by readers for their enjoyment
– *CMS:* collect and connect content, simple publication tool

## 2.2 Semantic Wiki Systems

A "Semantic Wiki" extends a Wiki by "semantic technologies" like RDF, OWL, Topic Maps, or Conceptual Graphs. The main idea is to make the inherent structure of a Wiki – given by the strong linking between pages – accessible to machines (agents, services) beyond mere navigation. This is generally done by annotating existing navigational links with symbols that describe their meaning. A link from *Mozart* to *Salzburg* could e.g. be annotated with *lived in* or *born in.*

Such annotations are useful for many purposes, e.g. for enhanced presentation by displaying contextual information, enhanced navigation by giving easy access to relevant related information, and enhanced "semantic" search that respects the context in addition to the content. Note that presentation, navigation, and search can be done in a rather generic manner, but often profit greatly from an adaptation to the represented context.

Semantic Wikis exist in many different flavours (e.g. Semantic MediaWiki [2], SemWiki [3], IkeWiki [7], PlatypusWiki [1], SemperWiki [4]). Some systems are still primarily focused on the page content and see annotations as optional "added value". For others, the semantic annotations are in the foreground and sometimes even more important than the actual content. Different systems serve different purposes, e.g. extending existing content by annotations to allow for better navigation, collaborative ontology engineering, etc. Commonly found features are:

**Typing/Annotating of Links.** Virtually all Semantic Wikis allow to annotate links by giving them certain types. The idea behind this is that every link carries meaning beyond mere navigation, as given in the example in the beginning of this section. The way link annotations are edited differs from system to system. Some Semantic Wikis include the annotations as part of the Wiki syntax (e.g. *Semantic MediaWiki* [2]), while others provide a separate editor for adding annotations (e.g. *IkeWiki*).

**Context-Aware Presentation.** Many Semantic Wikis can change the way content is presented based on semantic annotations. This can include enriching pages by displaying of semantically related pages in a separate link box, displaying of information that can be derived from the underlying knowledge base (e.g. a box with a graphical tree presentation, license information), or even rendering its content of a page in a different manner that is more suitable for the context (e.g. multimedia content vs. text content).

**Semantic Navigation.** Whereas a traditional Wiki only allows to follow a link, a semantic Wiki offers additional information on the relation the link describes. Such information can be used to offer additional or more sophisticated navigation. For instance, links are more independent from the textual context and can be displayed e.g. in a separate "related information" box. The page describing Mozart could e.g. offer a separate box with references categorised by "lived in", "composed", etc.

**Semantic Search.** Most Semantic Wikis allow a "semantic search" on the underlying knowledge base. Usually, queries are expressed in the language SPARQL, an RDF query language recently proposed by the W3C. Using "semantic search", users can ask queries like "retrieve all pieces composed by Mozart" or "retrieve all documents where the license permits derivative works".

**Reasoning Support.** Reasoning means deriving additional, implicit information from the facts entered into the system using predefined or user-defined rules in the knowledge base. For example, from the fact that "Mozart" composed "Die Zauberflöte", a system capable of reasoning could deduce that "Mozart" is a "Composer". Although reasoning is an important feature, it is only supported by few Wikis. Reasons might be that it is time-consuming, memory intensive, and can yield results that are not expected and/or traceable by the user.

## 3 Learning with Semantic Wikis

In this section, we characterise the potential and relevance of traditional and Semantic Wikis in learning environments. Furthermore we aim to give a short overview about the possibilities and advantages of Wikis in practical use. The use of Wikis in learning environments has only recently attracted attention and is rapidly gaining interest. We begin this section with a brief introduction into learning concepts aimed at the technical reader. We then introduce the current use of (traditional) Wikis in learning environments. We conclude with a discussion of possible benefits of Semantic Wikis over traditional Wikis.

### 3.1 Learning Concepts

Everyone has an informal notion of what learning "means", but different disciplines and traditions have developed their own understandings and perspectives. Before describing the potential of Wikis for learning, we briefly summarise the concepts that have been developed in recent years and are now widely accepted.

**Different Perspectives.** Learning is a natural, birth-given ability. Psychologists define learning as "a process that results in relatively consistent change in behaviour, or behaviour potential, and is based on experience" [9]. From the neuro-scientific and cognitive perspective, learning is a process of modification of cognitive structures (thinking). In the pedagogical perspective, these changes should be "good" in a normative way: after learning, the learner should have improved his skills or competencies, extended or corrected his knowledge, etc. Philosophers discuss the epistemological preconditions for knowledge and learning.

**Learning: State-of-the-Art.** In the understanding developed in the last 20 years, *learning* means to construct one's own understanding of the world, i.e. (in a cognitivist view) we interpret new information with the help of prior knowledge and experience. Learners interact with the environment, select and transform information, and construct their own knowledge. Learning is (in a constructivist view) a recursive, self-referential process and needs the stimulus and challenge through others [10].

It is nowadays commonly accepted that the possibilities of *teaching* are limited: depending on prior knowledge, biography, learning abilities, motivation, emotional arousal, interest, understanding, etc., pedagogical goals might be achieved, but success is not guaranteed. Teachers can not directly transfer their knowledge. Instead, they should act as facilitators, encouraging students to discover principles on their own.

### 3.2 Wikis in Self-directed Learning

Most learning takes place outside the formal boundaries of a class room or learning system. We learn e.g. by searching for information on the Internet, by reading newspapers, books, articles, by trying to solve problems on the job, etc. Such learning is called *self-directed* or *informal* learning. Malcolm Knowles [11] describes self-directed learning as a process "in which individuals take the initiative, with or without the help of others, in diagnosing their learning needs, formulating learning goals, identifying human and material resources for learning, choosing and implementing appropriate learning strategies, and evaluating learning outcomes" (p. 18).

**Reflection.** Writing text requires taking different views and perspectives on knowledge in order to make it explicit and understandable for readers, e.g. by integrating with contextual information and telling "a story". This results in additional reflection about the knowledge and thus learning. While this aspect is not specific to Wikis, the possibility to repeatedly update the content and structure gives success quickly and allows refinements later on. Furthermore, Wikis support the constructive process by allowing to embed content in a larger context using hyperlinks, and by considerably simplifying the restructuring of content. The hypertext structure of Wikis can also reflect and promote network thinking of the learners.

**Stimulative Nature.** A significant difference between a normal HTML page and a Wiki page is the possibility to directly change content. Even more so, most Wikis *invite* readers to edit the content. This possibility – when communicated rightly to readers – *challenges* the readers to question and rethink some or even all the content of a page ("is the author right? – what is my view? – is something missing? – can the text be improved?"), and thus mobilises critical thinking, which in turn initiates learning. Also, the dissociation from others and their perspectives on a topic is part of the (constructivist) learning process ("learning as experience of differences", [10]).

Additionally, as links to non-existent pages are usually rendered differently, Wikis also challenge readers to contribute to pages that do not yet exist, leading to active construction of knowledge and additional reflection about the content.

**Personal Learning and Knowledge Tool.** Self-directed learning also involves personal knowledge management: taking personal notes, collecting references, ideas, etc. The usefulness of Wikis for knowledge management has been investigated [3,12], so we do not go into details here.

### 3.3 Wikis in Educational Environments

Formal Learning describes learning activities that happen in an organised way, e.g. in schools, in university, in adult education centres, etc. Wikis can be used in different ways in formal learning situations. One use is as a knowledge repository for students to search in (see above). More interesting, however, is the use of an initially "empty" Wiki that is filled by the students themselves.

Wikis can play an important role both in blended learning (which combines traditional presence learning with technology-supported learning) and in "pure" eLearning. A particular advantage over other tools is that – prepared with the knowledge about a Web browser – the necessary technical knowledge can be acquired quickly. In the following, we describe different approaches to modelling the learning environment and show how Wikis can be supportive within these approaches.

**Cognitive Apprenticeship** [13] is based on the knowledge and skill transfer in the traditional master-apprentice education. In cognitive apprenticeship, learning is always situated in the context and happens via interaction with the environment and other individuals. The apprentice learns by working closely together with the master on real-world problems of the respective craft. The master acts as an archetype for the learner by doing work instead of trying to make explicit his knowledge.

Wikis provide an important benefit in this learning model: their collaborative features allow teachers and students to work closely together on a topic, e.g. writing a text or article, collecting information on a topic, etc. – regardless of the whereabouts of students and teachers. They thus aid in an important learning task of the emerging knowledge society.

**Cooperative Learning** differs from traditional curriculum-driven education in that students work in purposely heterogeneous groups to support the learning of their individuals. Important aspects of cooperative learning are positive interdependence of group members, individual accountability, face-to-face interaction, appropriate use of collaborative skills, and regular self-assessment of team functioning [14]. In cooperative learning, learners gain a realistic self-perception by looking at the other group members.

The collaborative features (collaborative editing, versioning, discussion next to the content) of Wikis make them particularly well-suited for cooperative learning environments, with no corresponding tool in traditional learning. Whereas collaborative working with twenty people using traditional methods – e.g. a chalk board, brainstorming sessions, etc. – is not viable because it takes a lot of time, is tiresome, and causes many learners to withdraw, collaborative working using a Wiki can easily function with hundreds or even thousands of people (cf. Wikipedia). Wikis are already used for a number of different tasks in cooperative learning. Examples are:

*Communities of Practice* are groups of persons pursuing common goals and interacting with other individuals [15]. In communities of practice, learning is a collaborative process of a group. Rather than looking to learning as the acquisition of certain forms of knowledge, communities of practice define learning as a situated process through the participation in the community. In order to fully participate in the community, members need to adopt to the communities shared knowledge and practices.

Wikis can serve as a knowledge platform for a community of practice where members of the community can share their knowledge with the group, put up interesting pieces of information, work together, discuss issues, etc. New members can use the Wiki for learning about the community and its practices. In a sense, even Wikis like Wikipedia can be considered as (large) communities of practice.

*Project-Based Learning.* Projects play a large role in the context of educational environments. Whether project weeks, project days, class-independent annual topics or smaller projects, the project method applies nearly everywhere. Project classes correspond to the general education programme of schools. The project method is seen as a way to reach the education goals. Wikis can represent – in case a project encloses many individuals or several classes – a very effective tool for project planning and documentation.[2]

*Collaborative Story Writing.* Wikis give learners completely new possibilities for creative writing. A Wiki can be seen as an interactive writing book, where students write together on an essay or story. The story does not necessarily have only one end; it can branch out like a tree or even graph with a lot of different

[2] Example: Planning of a musical project: `http://www.prowiki2.org/glarnerschulen/wiki.cgi?OrdnerMusical`

paths and ends. Talented students also have the possibility of supplementing illustrative figures or photographs to the story.[3]

*Interdisciplinary and Intercultural Learning.* The collaboration features and independence of Wikis from the actual whereabouts of the learners make them well-suited for interdisciplinary and intercultural learning, bringing together learners with different cultural and educational backgrounds. For example, religion can be discussed among christian, muslim, and hindu students, languages can be learned from other students with different mother tongues, etc.

### 3.4 Wikis as ePortfolios

Learning in general implies two different processes: on the one hand, one has to collect artefacts and pieces of information; on the other hand, these artefacts need to be integrated with ones existing knowledge space by "reflecting" the learning / knowledge building process on a meta-level. The sum of documentation and reflection illustrates the learning process as a whole and can therefore be used as an process oriented ePortfolio.

ePortfolios in general are a "structured personal digital collection of information describing and illustrating a person's learning, career, experience, and achievements."[4] Depending on the purpose, ePortfolios cover 3 major processes: *collection and presentation* (presentation portfolio), *reflection and communication* (learning- and process portfolio), and *evaluation* (assessment portfolio).

Wikis (in a learning context) can on the one hand obviously be used as a supportive technology for *collection and presentation* processes. These processes can support both, self-directed learning or community learning. On the other hand, *reflection and communication* purposes are covered by Wikis in terms of using the history function and the discussion function.

*History function:* a community, working together on one topic might investigate on a similar topic and find different aspects of an issue. By editing earlier versions of an item or a topic on the Wiki, the development process can be documented and reconstructed easily by the system itself. By using the *discussion feature* of a Wiki, the process of collaborative content generation combines the *collection and presentation* issues of ePortfolio work, the discussion function coveres the *reflection and communication* aspects.

The value of ePortfolios can be seen not only in the collection and presentation of artefacts. Too much information gets lost by only reproducing results of learning processes without documenting the meta-level, described in the development process in a holistic way. Wikis can support this process, but should be part of a bigger framework, dealing with personnel development / learning documentation and reflectional processes of this documentation.

[3] Example: Creative and Cooperative Writing with Wiki – Geschichtenwald: http://www.wikiservice.at/buecher/wiki.cgi?GeschichtenWald

[4] Definition by National Learning Infrastructure Initiative of EDUCAUSE, 2003

### 3.5 Possible Benefits of Semantic Wikis

In the following, we describe a number of benefits that Semantic Wikis may provide over traditional Wikis in the context of learning that go beyond those frequently mentioned for knowledge management (cf. e.g. [3,7]). The benefits are categorised in *Learning Process* and *Content Creation and Reuse*.

**Learning Process.** Semantic Wikis can participate in many ways in a learning process. In the following, we mention three possibilities:

*Semantic Annotations lead to Reflection about Knowledge.* The possibility to add semantic annotations in a Semantic Wiki and create a background model has a stimulating effect for reflecting about the learning content. Constructing a formal model requires structuring the content thoroughly. Hence, the learner needs to reconsider and possibly reorganise the content in the Wiki, leading to improved reflection about the content. The result of structuring the content and the changes history can additionally be used by the coach to assess the learner's progress.[5]

*Sharing of Perspectives.* Semantic Wikis also offer the possibility to share formal models between teachers and students and among students, and to participate in the collaborative building of a common model within a group. Sharing of knowledge gives learners the possibility to benefit from different perspectives stemming from different cultural, social, or educational backgrounds. This is particularly true in an open learning environment like the Web.

*Reasoning Provides Additional Insight.* Reasoning and inference capabilities of Semantic Web technologies can lead to unexpected interesting results that provide additional insight without requiring active search by the user. For example, a link from *Die Zauberflöte* to *Mozart* annotated with *composedBy* could instantly lead the user to the information that *Eine Kleine Nachtmusik* was also composed by *Mozart*, a piece of information possibly entered by someone else.

**Content Creation and Reuse.** Learning comprises not only the actual learning process but also the actual outcome of the learning process. This can include content objects represented traditionally as text oriented document (e.g. thesis, project report, paper, article), enriched with figures, images and some hierarchical structure.

*Typing of links adds value to content creation process.* Apart from its collaborative aspects, the hypertext and the Wiki paradigm added another layer of "structure": multiple paths/links within documents. However, these entities are fairly complex to maintain within a learning process, because the semantics of

[5] Note that currently existing Semantic Wikis do not support versioning of metadata.

the paths/links are not explicit. Semantic technologies support the formalisation of links by allowing to type objects and associations between them. This adds an additional dimension to the content creation process and allows further exploitation.

*Reusability of Learning Content.* Using Semantic Web technologies, learning content can be annotated using standardised knowledge models like IMS Learning Design[6] or Learning Objects Metadata (LOM)[7]. Semantically annotated content allows the designers of a curriculum to more easily reuse and combine existing content to create new course material. A Semantic Wiki provides an intelligent way for content creators to add and use such metadata.

*Interoperability.* A "learning environment" consists of a plethora of "tools" ranging from traditional classrooms and chalk boards over ePortfolios, Weblogs, and Wikis to sophisticated learning management systems. Naturally, there is a desire that such tools are widely interoperable to share information between them. The Semantic Web technologies used by Semantic Wikis provide the chance to re-use significant parts of the gained knowledge model within other applications, e.g. to export a specific knowledge model built for within the Wiki as the background model for the own competency portfolio within a specific domain, or to interact with a learning management system.

## 4 IkeWiki

A number of Semantic Wiki systems are currently under development.[8] In the following, we introduce our own system called *IkeWiki*. We believe that IkeWiki has many of the features desirable for Wiki use in learning environments.

### 4.1 Design Principles

Although now also considered in different settings, IkeWiki has originally been developed as a prototype tool to support knowledge workers in collaboratively formalising knowledge [7]. Although holding in other areas as well, IkeWiki's design principles are influenced by this idea:

**Easy to Use, Interactive Interface.** IkeWikis interface (Figure 2) resembles as closely as possible the Wikipedia interface which people are familiar with. Furthermore, IkeWiki offers an interactive WYSIWYG[9] editor (using AJAX[10]

[6] http://www.imsglobal.org/learningdesign/index.html
[7] http://ltsc.ieee.org/wg12/
[8] http://wiki.ontoworld.org/index.php/Semantic_Wiki_State_Of_The_Art
[9] WYSIWYG: "what you see is what you get"
[10] AJAX: "asynchronous JavaScript and XML" – used for interactive web applications

technology to communicate with the server backend) in addition to the traditional structured text editor, as WYSIWYG editors generally have a better acceptance among non-technical users.

The WYSIWYG editor also supports interactive typing of links and resources. The interface is designed in a way that users are invited to annotate their content with semantic annotations, instead of hiding them in the syntax. We consider this an important aspect, particularly in learning environments.

**Immediate Exploitation of Semantic Annotations.** An important motivating aspect of Wiki systems is that content is immediately available to the public when a user clicks on "save". Similarly, IkeWiki allows immediate exploitation of semantic annotations for enhanced editing, presentation, navigation, and searching, even if the knowledge base is not yet fully formalised.

**Support for Different Levels of Experience.** IkeWiki is designed as a tool for collaborative working. In such a process, it is common that non-technical people (e.g. learners) work together with experts (e.g. teachers). Therefore, IkeWiki supports all levels of experience. This means that certain advanced functionalities can be hidden from novice users but are available to experienced users.

**Support for Different Levels of Formalisation.** Different application areas need different levels of formalisation [7], and as Jim Hendler said:[11] "a little semantics goes a long way". One of the goals of IkeWiki is thus to support formalisation of knowledge all the way from informal texts to formal ontologies. Also, this means that parts of the knowledge base might be more formalised than others, and that formal knowledge is in constant evolution.

**Support for Reasoning.** Unlike most other Semantic Wikis, IkeWiki supports reasoning on the knowledge base. Reasoning is important as it allows to derive knowledge that is not explicit; it is thus the true power of Semantic Web technology. At the moment, IkeWiki supports only OWL-RDFS reasoning, but an extension with a user-accessible rule engine is planned. As mentioned earlier, reasoning can be an important supportive service in learning environments.

**Compatibility with Semantic Web standards.** To be able to exchange data with other applications (e.g. other Wikis, learning management systems, ePortfolio systems, Web services), IkeWiki is based on existing Semantic Web standards like XML, RDF and OWL. Note that other knowledge representation formats like conceptual graphs are conceivable but not investigated at this point.

[11] as conference chair in the opening speech of the 2003 International Semantic Web Conference; Sanibel Island, Florida, USA, October 2003

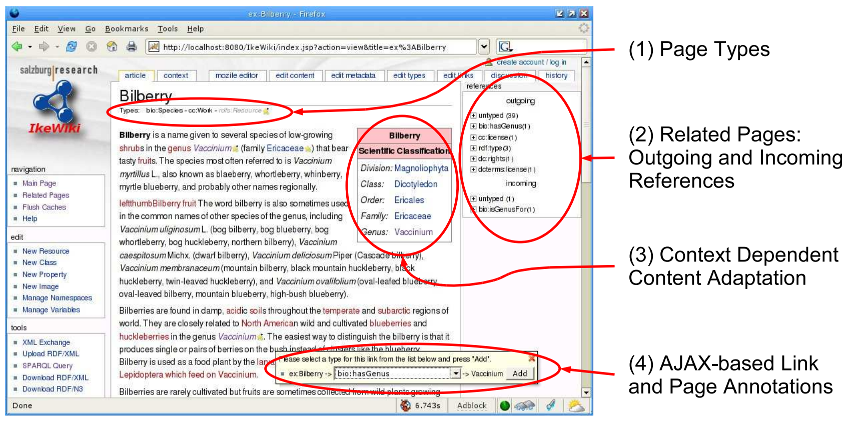

(1) Page Types
(2) Related Pages: Outgoing and Incoming References
(3) Context Dependent Content Adaptation
(4) AJAX-based Link and Page Annotations

**Fig. 2.** Sample page in IkeWiki; type information below the title (1); incoming and outgoing links are displayed in a box on the right (2); context-dependent rendering: automatically generated taxonomy box (3); interactive editing (4).

**Compatibility with Wikipedia/MediaWiki.** A significant amount of information is available in Wikipedia. To reuse it, IkeWiki supports the Wikipedia syntax. This allows users to import existing content from Wikipedia into IkeWiki (e.g. via simple copy and paste) and directly begin working.

## 4.2 Interface

IkeWiki uses a purely browser-based interface (cf. Figure 2). The current implementation only supports the Mozilla browser family due to its standards compliance and free availability.

**Page View.** A sample page view is shown in Figure 2. In the figure, you can see a sample article (copied from Wikipedia) about the "Bilberry". Type information is shown below the page title (1). Links to (semantically) related pages are displayed in a separate "references box" on the right hand side (2). The taxonomy box (3) showing the biological classification of the described plant is automatically generated from existing semantic annotations (i.e. Bilberry *has-Genus* Vaccinium) and is an example for context adaptation. Finally, (4) shows interactive typing of links using AJAX technology.

**Content Editor.** The content editor is available in two flavours: as a traditional structured text editor and as a WYSIWYG editor. The structured text editor is aimed at expert users that are familiar with other wiki systems, and allows to directly copy content from Wikipedia. The WYSIWYG editor is aimed at novice users creating new content. The WYSIWYG editor interacts with the server backend: links are automatically recognised and verified, and semantic annotations can be done directly in the editor (as also shown in Figure 2, (4)).

**Semantic Annotations Editor.** Semantic annotations are separated into three editors: the *metadata editor* allows to fill in textual metadata related to a page (like Dublin Core metadata or RDF comments). The *type editor* allows to associate one or more of the types available in the system with a page. The *link editor* allows to annotate outgoing and incoming links with type information. In the editors, available annotations are determined by the reasoner based on the page and link types; for example, if a link from "Mozart" to "Die Zauberflöte" is annotated by "composerOf", the system will automatically associate the type "Composer" with the page describing "Mozart".

## 5 Related Work

A number of studies investigating the use of (traditional) wikis in learning environments is available. A good survey over the use of wikis in teaching is given in [16]. A recent study by Beat Döbeli Honegger [17] investigated the use of wikis in schools and provided interesting results regarding the stimulative nature of wikis. The study conducted a project to create hypertext on Greek mythology and observed the effects on student learning. A visualisation of the content was done via TouchGraph technology. The EduCause fact sheet [18] further provides a very concise overview over the potential of wikis for learning.

To the best of our knowledge, this article is the first to consider Semantic Wikis as a tool for learning, as existing related work is mostly concerned with knowledge management [3,5] and knowledge engineering [7].

## 6 Perspectives and Conclusion

We investigated the use of Semantic Wikis as a tool in learning environments. The potential of using wikis, especially Semantic Wikis, in learning environments appears to be significant, but has not yet been proven in real learning environments and – although well-founded – only reflects our own considerations. In the near future, we will therefore develop learning scenarios involving Semantic Wikis and try them in real-world settings. In addition, we will investigate the possible integration of Semantic Wikis with other learning tools like ePortfolios and learning management systems.

The prototype system IkeWiki also presented in this article is under active development. Plans for the near future are a more efficient reasoning support and improvements of the user interface. On the long term, we would like to implement a Semantic Wiki system with enhanced collaboration and editing features like synchronous editing, improved WYSIWYG editing, and a tighter integration of content and metadata editing.

IkeWiki is available as OpenSource software licensed under the GNU General Public License at `http://ikewiki.salzburgresearch.at`.

## References

1. Tazzoli, R., Castagna, P., Campanini, S.E.: Towards a Semantic WikiWikiWeb. In: 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan (2004)
2. Krötsch, M., Vrandečić, D., Völkel, M.: Wikipedia and the Semantic Web - The Missing Links. In: Proceedings of the WikiMania2005. (2005)
3. Völkel, M., Oren, E.: Personal Knowledge Management with Semantic Wikis. (2006)
4. Oren, E.: SemperWiki: a semantic personal Wiki. In: 1st Workshop on The Semantic Desktop, colocated with ISWC05, Galway, Ireland (2005)
5. Aumueller, D., Auer, S.: Towards a Semantic Wiki Experience – Desktop Integration and Interactivity in WikSAR. In: Semantic Desktop Workshop 2005 at ISWC'05, Galway, Ireland (2005)
6. Schaffert, S.: IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. (2006)
7. Schaffert, S., Gruber, A., Westenthaler, R.: A Semantic Wiki for Collaborative Knowledge Formation. In: Semantics 2005, Vienna, Austria (2005)
8. Reding, V., Diamantopoulou, A.: Making a European Area of Lifelong Learning a Reality. Communication from the European Commission (2001)
9. Zimbardo, P., Gerrig, R.: Psychology and Life. 14th edn. HarperCollins, New York (1996)
10. Siebert, H.: Konstruktivismus. Konsequenzen für Bildungsmanagement und Seminargestaltung. Deutsches Institut für Erwachsenenbildung (1998)
11. Knowles, M.S.: Self-Directed Learning. A guide for learners and teachers. Englewood Cliffs: Prentice Hall, Cambridge (1975)
12. Wagner, C.: Wiki: A Technology for Conversational Knowledge Management and Group Collaboration. Communications of the Association for Information Systems **13** (2004)
13. Collins, A., Brown, J.S., Newman, S.E.: Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics. In Resnick, L.B., ed.: Knowing, Learning, and Instruction. Essays in Honor of Robert Glaser. Lawrence Erlbaum Associates, New Jersey (1989) 453–494
14. Johnson, R.T., Johnson, D.W.: An Overview of Cooperative Learning. In Thousand, J., Villa, A., Nevin, A., eds.: Creativity and Collaborative Learning. Brookes Press, Baltimore (1994)
15. Wenger, E.: Communities of Practice. Learning as a social system. Systems Thinker (1998)
16. Lamb, B.: Wide Open Spaces: Wikis, Ready or Not. EDUCAUSE review (2004)
17. Honegger, B.D.: Wikis – a rapidly growing phenomenon in the german-speaking school community. In: International Symposium on Wikis (WikiSym05), San Diego, USA (2005)
18. EduCause Learning Initiative: 7 things you should know about . . . Wikis. (2005)

# Harvesting Wiki Consensus - Using Wikipedia Entries as Ontology Elements

Martin Hepp[1,2], Daniel Bachlechner[1], Katharina Siorpaes[1]

[1]Digital Enterprise Research Institute (DERI), University of Innsbruck
[2]Florida Gulf Coast University, Fort Myers, FL, USA
{martin.hepp|daniel.bachlechner|katharina.siorpaes}@deri.org

**Abstract.** One major obstacle towards adding machine-readable annotation to existing Web content is the lack of domain ontologies. While FOAF and Dublin Core are popular means for expressing *relationships* between Web resources and between Web resources and literal values, we widely lack unique identifiers for common concepts and instances. Also, most available ontologies have a very weak community grounding in the sense that they are designed by single individuals or small groups of individuals, while the majority of potential users is not involved in the process of proposing new ontology elements or achieving consensus. This is in sharp contrast to natural language where the evolution of the vocabulary is under the control of the user community. At the same time, we can observe that, within Wiki communities, especially Wikipedia, a large number of users is able to create comprehensive domain representations in the sense of unique, machine-feasible, identifiers and concept definitions which are sufficient for humans to grasp the intension of the concepts. The English version of Wikipedia contains now more than one million entries and thus the same amount of URIs plus a human-readable description. While this collection is on the lower end of ontology expressiveness, it is likely the largest living ontology that is available today. In this paper, we (1) show that standard Wiki technology can be easily used as an ontology development environment for named classes, reducing entry barriers for the participation of users in the creation and maintenance of lightweight ontologies, (2) prove that the URIs of Wikipedia entries are surprisingly reliable identifiers for ontology concepts, and (3) demonstrate the applicability of our approach in a use case.

**Keywords.** Wikis, Ontologies, Reuse, Collaborative Ontology Building, RDF, RDF-S, OWL

## 1. Introduction

Ontologies are consensual, explicit conceptualizations of a domain of discourse [1, 2]. In short, they are unambiguous representations of concepts, relationships between concepts (for example, but not limited to, a hierarchy), instances, and axioms. Unambiguous in this sense means two things: First, the representation should allow humans to precisely grasp the meaning of any element, so that humans have a well-defined vocabulary at hand when annotating data, expressing queries, or drawing conclusions. Second, the representation should have a formal semantics, so that it supports machine reasoning. For a comprehensive overview, see [3]. However, it is important to note that ontologies are not just formal representations of a domain, but much more *community contracts* about such formal representations. Since a discourse is a dynamic social process, during which previous propositions are often modified, especially refined, or discarded, and new topics need to be added, such a community contract cannot be static, but must be able to reflect the community consensus at any point in time. Also, the respective community must be technically and skill-wise able to be involved in building the, or committing to the, ontology.

Ontologies can have a varying degree of expressivity, ranging from flat collections of consensual concepts to abundantly axiomatized models. Many ontologies have a subsumption hierarchy that allows to infer implicit class membership, but this is not a mandatory property. In its least expressive form, an ontology is a collection of named concepts with a natural language definition of their meaning, i.e. a controlled vocabulary.

Though more expressive ontologies support more sophisticated reasoning, even such flat ontologies can be extremely useful. Already having unique identifiers (e.g. URIs) assigned to concepts described in natural language is very beneficial, for it helps improve recall and precision in information retrieval by eliminating the significant amount that is caused by synonyms and homonyms.

Now, we can observe on one hand that there are very few real domain ontologies available; a large share of ontologies published on the Web are outdated, dead collections created in some academic research context. On the other hand, the English version of Wikipedia contains more than one million entries, which means it holds unique identifiers for the same number of concepts.

Currently, both ontology tools and ontology languages impose high entrance barriers for potential users, excluding the vast majority of Web users. The Web Ontology Language OWL [4], for example, is in several aspects non-intuitive for anybody who does not come from the Description Logics (DL) community, and publishing an ontology in a persistent manner requires infrastructure (e.g. a HTTP server) that is not available to an average user.

This in combination likely contributes to the fact that the most popular approach of creating ontologies is engineering-oriented, i.e., a small number of skilled individuals carefully constructs the representation of the domain of discourse, and releases the results at some point in time to a wider community of users. However, (1) the sequential paradigm of this approach and (2) the fact that a small group constructs the ontology for a bigger group has several weaknesses:

First, the *ontology evolution is not under the full control of the ontology user community*. For example, missing entries cannot be added by any user who reveals the need for a new concept, but has to be added by the small group of creators. This is slow and incomplete, for it may be too much a burden for the users to report missing entries. Also, the addition may take too long if the domain is undergoing conceptual change. In natural language, in comparison, the evolution of the vocabulary is under the control of the user community. Anybody can invent and define a new word or concept in the course of communications.

Second, users creating annotations cannot easily grasp the intension of a concept; there is often a lack of communication between ontology creator and user. Somebody using an ontology e.g. for annotating instances or expressing queries has little help in determining whether a given concept is suitable for his or her needs, since the formal part of the ontology only *constrains* the interpretation of a concept, but does, with the exception of very expressive ontologies, not actually *define* the meaning of this concept. This leaves the ontology user with sparse natural language descriptions, e.g. in the form of the Dublin Core field `dc:description`. Such is often not sufficient to check whether the ontology creators read the concept in the same manner as the potential ontology user does, and many ontology creators with a strong formal background put little emphasis on the natural language definitions and related non-functional properties. For example, "ice cubes" in UNSPSC can be understood as any form of ice cubes or as all ice-cube-related business documents; see [5] and [6]. As a consequence, two parties referring to the same ontology might read the intension of the concept differently, which can lead to incomplete and/or inconsistent results and operations.
We propose to directly use the infrastructure and culture of Wikis as an ontology engineering workbench that fosters true collaborative ontology creation and maintenance for lightweight ontologies, in the sense that anybody can add a new element to the ontology, and refine or modify existing ones. At the same time, we want to reuse the vast amount of Wikipedia entries (more than one million in the English version) as ontology components.
We especially propose the use of multimedia elements to improve the richness and disambiguity of informal concept definitions in an ontology. Also, we regard it as beneficial if the definition of a concept is not separated from the discussion that lead to shaping the intension of this concept, since the history of a conceptualization is a valuable part of the respective definition. In many sciences, especially philosophy, the notion of a term is hard to grasp without knowing the historical debates that lead to its introduction.

### 1.1 Our Contribution

In this paper, we (1) show that standard Wiki technology can be easily used as an ontology development environment without modification, reducing entry barriers for the participation of users in the creation and maintenance of lightweight ontologies, (2) present a quantitative analysis of current Wikipedia entries and their properties, (3) prove that the URIs of Wikipedia entries are surprisingly reliable identifiers for ontology concepts, and (4) demonstrate how the entries available in Wikipedia can be used as ontology elements.

### 1.2 Research Approach

First, we developed a minimal technical solution for using Wikipedia entries as ontology elements in RDF. Second, we took a representative, random sample (n=100) from a snapshot of the English version of Wikipedia (http://en.wikipedia.org/). Third, we analyzed whether the concept represented by the URI at the time of adding this entry is still consistent with the most recent description retrievable at the respective URI, i.e. whether annotations made using the URI in the past remain correct despite the fact that Wiki entries can be easily modified by an open community. We especially analyzed the amount of disambiguation pages, which are inserted when the same terminology refers to distinct concepts in various contexts. Fourth, we quantitatively analyzed properties like average age of entries and amount of change per time. Since we know from statistics that random samples are, if designed properly, very reliable estimates for the full population (i.e. the full Wikipedia content), our approach returns precise data about the suitability of Wikipedia content as concepts.

### 1.3 Related Work

Work related to ours mainly falls into the following categories:

**Community-driven Ontology Building:** There is already significant literature about collaborative ontology engineering in general, e.g. Tadzebao and WebOnto (see [7]). [8] describe collaborative ontology building in analogy to Wikis, but (1) do not borrow more from the Wiki community than the pure name, (2) take a very rich ontology meta-model as the starting point, (3) do not elaborate on the community focus of ontology building, and (4) do not address the advantage of adding multimedia elements in the informal descriptions of concepts.
Wikispecies [9] can already be regarded as a first Wiki-centric ontology for species. It even includes a subsumption hierarchy; which is, however, a lesser challenge in this narrow application domain since there is a single consensual taxonomy in Biology, the Linnaean taxonomy. Recently, the term “Folksonomies” was brought up as a reference for on-the-fly classifications created by users [10, 11]. This work is very much related to ours, however there are main differences. First, we aim at reusing the vast amount of existing Wikipedia entries as ontology elements. Second, we do not distinguish between tags and Wikipedia pages, i.e. we propose to use each Wikipedia URI as the identifier for a concept. Third, we point to the importance of multimedia elements in Wikipedia entries for capturing the intension of such concepts. Fourth, we stress the fact that the history function of Wikipedia is an important component of a concept definition, since it reflects the discourse that has led to the most recent state. [12] points out that the entry barriers for ontology development and usage should be lowered. The “Simple Knowledge Organisation System (SKOS)” [13] is such an approach. [14] describes the DILIGENT knowledge processes which proposes ontology evolution and collaborative concept mapping and refinement as core techniques for building ontologies in order to deal better with domain dynamics and other ontology engineering challenges.

**Augmenting Wikis with Semantic Web technology:** Platypus Wiki [15] is a Wiki augmented by Semantic Web approaches, namely RDF, while we want to use Wikis for creating ontologies that can be used anywhere in the Semantic Web. [16] describes Rhizome, a Semantic Wiki system that also includes the functionality of creating arbitrary RDF resources easily. A first version of our work has been presented in [17], but this prototype aimed at deploying a modified Wiki installation as an ontology engineering platform, while we now think that Wikipedia must be the starting point due to the enormous number of existing entries and community pickup.

At a non-ontology level, the usefulness of Wikipedia as a point of reference is discussed in [18]; however, this refers more to the aspect whether all facts said about a topic are authoritative in detail, and not whether the URIs represent consensual concepts.

Especially in the last months, there is vast interest in combining Semantic Web and Wiki approaches. However, all approaches known to us are different to ours in the way that they aim at augmenting Wikis with Semantic Web components, while we propose (1) to use unmodified Wikis as a platform for collaborative ontology building on the level of named classes, and (2) harvest the wealth of concept definitions already contained in Wikipedia. In this sense, our work is complimentary to "Semantic Web Wiki" work and can be easily combined with such approaches.

## 2. Understanding Wikipedia as an Ontology Asset and Ontology Workbench for the Masses

We propose to use Wiki implementations in general and especially Wikipedia as a means for

(1) defining URIs for concepts,

(2) describing the intension of those concepts in natural language, and probably augmented by multimedia elements, e.g. drawings, pictures, videos, or sound recordings, and

(3) preserving the discourse that has led to the current version of a Wiki page as an important part of the definition of the respective URI.

In a nutshell, we understand the URIs of Wiki/Wikipedia entries as identifiers for named classes. This approach appears very straightforward and might even be perceived trivial. However, it is trivial only on the technological level, but should be quantitatively validated prior to its usage.

The motivation for this approach is based on the following aspects:

(1) Wikipedia contains more than 1,000,000 entries and is likely the biggest collection of URIs augmented by a textual definition available.
(2) Wikipedia is popular as a reference and its concepts can thus be expected to have commitment by a wide audience. Based on our analysis given below, we can estimate that the total amount of active contributions (e.g. additions or modifications) exceeds 2,465,000 per month. More than 50 % of the concepts have been changed at least once per each month of their existence.
(3) Wiki technology imposes only minimal requirements on a user and is likely the simplest way of creating a persistent URI plus informal description. Anybody can add a URI for a needed concept anytime.
(4) Most Wiki packages contain a comprehensive history function that allows referring to both the latest version as well as each past version of an entry

using unique URIs. Thus, different states of the discourse become First Order Objects (FOOs) that can also be referred to.

The main paradigm of our work is simplicity, i.e. we want to support only as much functionality as can be used productively by a large share of the community.

### 2.1. Research Challenges

When using Wiki entries as ontology elements, we see at least the following research challenges.

**Resource vs. Concept:** One can argue whether the URI

```
http://en.wikipedia.org/wiki/Let_it_be
```

refers to this specific Wikipedia entry as a resource or to the respective album by the Beatles. We propose as a minimal ontological commitment to our approach that each Wikipedia entry is to be understood as the entity that an average layman associates with this description. In this sense, the URI quite naturally reflects the Beatles album, not the Wikipedia description of the album. This is a proposed social convention and can of course be debated, but makes a lot of sense in the context of our proposal.

**Wiki Entries: Classes or Instances:** Since there is no explicit knowledge representation model in the background, a Wiki entry can be anything; it is not clear whether it refers to an instance, a concept, or a property. By social convention, Wikipedia contains mostly entries that are proper nouns and does not include relationships and properties (see also below). So it must be clarified whether a Wiki entry is to be treated as a class or as an instance, at least if the ontology model requires a choice between these two. We solve this issue by omitting this distinction between instance and class, which is no significant problem in pure RDF or in OWL Full.

**Versioning and Wiki URI Schemes:** A standard Wiki already provides all functionality necessary to create a textual definition and a unique URI. For example, anybody could have added an entry for the Republic of Austria to Wikipedia, now available at

```
http://en.wikipedia.org/wiki/Austria.
```

We could immediately use this mechanism and propose to re-use this URI not only as the resource locator for retrieval of the description, but also as the identifier for the concept “Republic of Austria”. Now the problem is that since everybody can alter the text, we never know whether the current version is a monotonic extension of any previous version. So anybody who used this URI for the annotation of instances or any other statement might find that his statement no longer holds with the modified version. We propose a very hands-on solution, based on a combination of the “history” functionality in the MediaWiki distribution, and a versioning scheme embedded in the URI for concepts, same as used by the W3C for W3C documents or the WSMO, WSMX, and WSML working groups [19]. The main idea is that the general URI, e.g.

```
http://en.wikipedia.org/wiki/Austria
```

always refers to the latest version, while all intermediate versions have an additional URI of their own.
In MediaWiki/Wikipedia, all intermediate versions already have unique identifiers in the following form:

```
http://en.wikipedia.org/w/index.php?title=Austria&oldid
=23005009
```

However, since this includes the name of the script "index.php", it is not fully compliant with the design principles of URIs, see [20].
It would be desirable if the MediaWiki software is modified in a sense that makes the history entries use persistent URIs that are not bound to implementation details (e.g. PHP). This could be achieved e.g. by adding the date and time of creation (plus probably the IP address of the originator):

```
http://en.wikipedia.org/wiki/Austria/YYYY-MM-DD-HH-MM-SS-IP
```

This allows referring either to the latest version or to any specific version. It also makes it possible to create statements *about* a specific version. This challenge is closely related to the next one.

**Conceptual Consistency of URIs over time:** Wiki entries can be modified and changed by anyone and there are no substantial institutional agreements between the users who create a new entry and the ones who modify it later. It is possible that the concept represented by a URI changes substantially over time, rendering old annotations inconsistent. This is especially a problem when so called "disambiguation pages" are introduced, which happens when the community realizes that the same word is a homonym and used in very different senses in different contexts. In such cases, the original page is turned into a disambiguation page that contains separate links to the multiple context-specific entries. A core part of our work presented in this paper deals with a quantitative analysis of this problem, i.e. whether this theoretical problem is a significant obstacle, or whether it is negligible.

**Dominance of Proper Nouns:** While Wiki packages alone can also be used to define URIs for properties, e.g.

```
http://en.wikipedia.org/wiki/isAFriendOf,
```

it is by social convention that Wikipedia does not contain such entries. This means that we cannot find properties and relationships as entries in Wikipedia. There are at least three ways of dealing with this:

(1) We use properties and relationships defined in popular existing ontologies, namely Dublin Core elements [21, 22] together with Wikipedia entries.
(2) We create complimentary property ontologies in an engineering fashion, e.g. "sells", "rents", "repairs" for e-Commerce applications.
(3) We modify Wiki packages so that they can be used for defining object properties (linking resources as subjects to Wikipedia entries as objects) and

datatype properties (linking resources to literal values) and deploy a complimentary “Property Wikipedia”.

All three approaches can be used in combination. We have already implemented the second and third approach. For reasons of simplicity, we restrict the example in this paper to the use of Dublin Core elements, though.

**Redundancy:** In collaborative ontology engineering, it can happen easily that multiple entries for the same concept are created. This has no negative impact on precision, but lowers the recall of information retrieval. Wiki contains mechanisms for merging pages in such cases. In this case, “#redirect [[PAGENAME]]” is to be inserted into the body of the discontinued page. Such links could be translated into statements of equivalence.

**Selection of a Proper Ontology Meta-Model:** We have to define an ontology meta-model that is suitable for a large audience. In our current approach, we support only plain RDF and completely leave out any kind of hierarchical order. This is not because we think this would be irrelevant; we are rather still researching proper support mechanisms that help yield consensual subsumption hierarchies. The problem with collaborative building of subsumption hierarchies is that a local modification can have lots of unwanted side effects that are not immediately obvious.

## 2.2 Example

In the following, we give an example of how Wikipedia entries can be used for describing Web resources. The example is based on the social convention that the reused Wikipedia entries are understood as the entity or concept that an average layman associates with this description, not as the Web resource itself. In this sense,

```
http://en.wikipedia.org/wiki/John_Lennon
```

refers to John Lennon as the singer and not to the Wikipedia entry about John Lennon.

The example below represents the facts that

- John Lennon was a contributor to the Beatles album “Let It Be”,
- the title of this Beatles album is “Let It Be (Beatles Album)”,
- John Lennon is related to John Lennon’s discography and
- that John Lennon can be described by *“John Winston Ono Lennon was a singer, songwriter, poet and guitarist for the British rock band The Beatles”*.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY wiki "http://en.wikipedia.org/wiki/">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">

<rdf:Description  rdf:about="&wiki;Let_it_be">
  <dc:title>Let It Be (Beatles Album)</dc:title>
  <dc:contributor rdf:resource="&wiki;John_Lennon"/>
</rdf:Description>

<rdf:Description rdf:about="&wiki;John_Lennon">
  <dc:description> John Winston Ono Lennon was a singer, songwriter,
poet and guitarist for the British rock band The
Beatles.</dc:description>
  <dc:relation rdf:resource="&wiki;John_Lennon_discography"/>
</rdf:Description>

</rdf:RDF>
```

Figure 1 shows the resulting RDF graph.

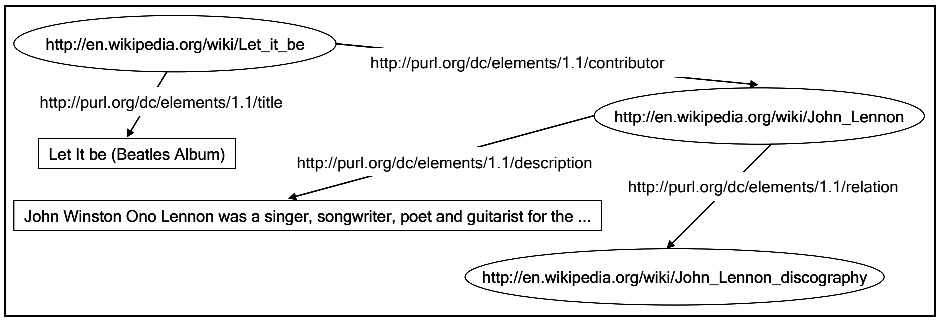

http://en.wikipedia.org/wiki/Let_it_be
http://purl.org/dc/elements/1.1/contributor
http://purl.org/dc/elements/1.1/title
http://en.wikipedia.org/wiki/John_Lennon
Let It be (Beatles Album)
http://purl.org/dc/elements/1.1/description
http://purl.org/dc/elements/1.1/relation
John Winston Ono Lennon was a singer, songwriter, poet and guitarist for the ...
http://en.wikipedia.org/wiki/John_Lennon_discography

Figure 1. RDF graph of the example.

## 3. Evaluation

In this section, we provide evidence that our approach is not only possible from a technical standpoint, but that the URIs of Wikipedia entries are surprisingly reliable identifiers for ontology concepts, despite the fact that they are yielded in a community-driven manner.

### 3.1 Methodology

We want to test whether the concepts defined by the URIs of Wikipedia entries undergo significant change during their lifespan, or whether modifications tend just to add more information, which would not change the intension of the concept, but just allow additional inferences.

For this purpose, we took a random sample (n=100) of entries in the English version of WikiPedia on November 17, 2005. For this purpose, we used the „random page“

functionality of the MediaWiki software. We assume that the random number generator employed is of sufficient quality for the purpose of this survey. We know from statistics that the mean and median of a sample is a reliable estimate for the mean and median of the full population, which frees us from the need to analyse all entries in Wikipedia.

For each of the selected Wikipedia URIs, we performed the following two tasks:

(1) We compared whether the concept or entity identified by the URI has changed significantly between the very first version and the current versions, in the sense that a layman annotation of a Web resource or a layman statement about the initial concept would hold for the first version but not for the current or vice versa. We distinguished the following cases:

    **Case 1a:** No significant change in meaning; the entry has been a stable, regular concept from its very first version to the current one.

    **Case 1b:** The entry has always been a Wiki "disambiguation page". It refers to a stable concept (i.e. all homonyms that could be referred to by this name).

    **Case 2:** A minor change in meaning has occurred. An example is that "Gloucester Courthouse" initially referred to the town and now refers to the "census designated place", which is still the same for many purposes.

    **Case 3a:** There was a major change in meaning.

    **Case 3b:** The URI was a regular entry in the beginning but turned into a disambiguation page later.

(2) For each entry, we also recorded the time and date of creation, the time and date of the last modification, the amount of editing tasks over its lifespan and per month of existence, and its age, i.e. the time lapsed between the initial creation and the date of our analysis (November 17, 2005).

Our hypothesis is that despite the ongoing change and uncontrolled editing of Wikipedia entries, there exists stable community consensus about the meaning of the respective URI.

### 3.3 Results

In the following, we summarize the results of our survey. Table 1 shows that only 3 % of the sample have turned into a disambiguation page during its lifespan. This is insofar important as this category of entry can have the most negative impact on precision in the usage of concepts for information retrieval, since initially, two communities might use the same URI to refer to distinct concepts.

Table 2 summarizes our findings with regard to the stability of concepts over their lifespan. One can see that 89 + 5 = 94 entries out of 100 were stable and could be used for annotation purposes without major problems. One entry underwent a slight change in meaning and 2 +3 = 5 entries were substantially modified. In other words, 95 % of the concepts can be used without or with only minor problems.

Table 1. Amount of URIs in the sample (n=100) that have turned into disambiguation pages

| **Disambiguation pages** | | |
|---|---|---|
| **URI refers to a regular concept** | **URI has always been a disambiguation page** | **URI became a disambiguation page during its lifespan** |
| 92 | 5 | 3 |

Table 2. Amount of significant changes in meaning between an initial and the current version of Wikipedia entries

| **Significant changes in meaning between initial and current version of Wikipedia entries** | | | | |
|---|---|---|---|---|
| Case 1: None | | Case 2: Minor | Case 3: Major | |
| 1a: Stable, regular concept | 1b: Always a disambiguation page | Slight change in meaning | 3a: Major change in meaning | 3b: URI became a disambiguation page |
| 89 | 5 | 1 | 2 | 3 |

Table 3 shows the distribution properties of the number of modifications per Wikipedia URI. The median (i.e. the element in the middle of the sample) was changed 9.5 times during its lifespan. In other words, 50 % of the entries are changed 9.5 times or less. In relation to the duration of their existence, 50 % were changed 1.2 times a month or less. A look at the quartiles Q1 through Q4 reveals that the lowest 25 % of entries in Wikipedia was changed between 1 and 5 times (Q1), the next 25 % were changed between 5 and 9.5 times, the third 25 % were changed between 9.5[1] and 19 times and the 25 % of entries that were modified most frequently underwent between 19 and 233 modifications.
If we multiply the mean of modifications per month of existence (2.9) with the total number of Wikipedia entries at the time of the survey (850,000 in November 2005), we reveal that there are on average 2,465,000 changes to entries in the English Wikipedia each month, which points to quite an active user community.

[1] The reason why the median value is not an integer number is that we have an even sample size. In this case, if the two elements in the middle of the population have different values, per definition, the *mean* of these two is the median. Thus the 50st entry underwent 9 changes and the 51st underwent 10 changes.

Table 3. Distribution properties of the number of modifications per Wikipedia URI

| Number of modifications per Wikipedia URI | | |
|---|---|---|
| | Absolute number | Modifications per month of existence |
| Mean | 21.8 | 2.9 |
| Median | 9.5 | 1.2 |
| Standard Deviation | 37.5 | 7.1 |
| Q1 | 5.0 | 0.6 |
| Q2 | 9.5 | 1.2 |
| Q3 | 19.0 | 2.7 |
| Q4 | 233.0 | 66.6 |

Table 4 indicates the distribution of the age of entries in days. 50 % of the entries were created less than 363 days before November 17, 2005. This is an amazing indication of how Wikipedia has gained interest and user involvement. 75 % (see the third quartile, Q3) were created less than 610 days before November 17, 2005, and only 25 % of the entries have been created more than 609 days ago.

Table 4. Lifespan in days (from creation until Nov 17, 2005)

| Lifespan in days (from creation until Nov 17, 2005) | |
|---|---|
| Mean | 412,6 |
| Median | 362,7 |
| Standard Deviation | 348,9 |
| Q1 | 102,7 |
| Q2 | 362,7 |
| Q3 | 609,0 |
| Q4 | 1360,7 |

## 4. Discussion

The data from our survey shows quite clearly that for the vast majority of Wikipedia entries, there is community consensus about the meaning of the URI from the very beginning to the most recent version. In other words, communities seem to be able to achieve consensus about named classes as very lightweight ontological agreements in an unsupervised fashion and with only the known mechanisms for preventing destructive changes of standard Wiki software.

As shown above, we can estimate that each month, about 2,465,000 change operations are made by Wikipedia users, but only 5 % of concepts change in a major

sense during their lifespan. We think this is a fundamental argument in favor of community-centric ontology building.
Also, our findings show that the majority of work on Wikipedia has been done in the last 20 months, since 75 % of the content of the English Wikipedia has been added in that timeframe.
Of course, there are drawbacks. First of all, what we can reuse from Wikipedia as ontology components are just named classes. There is zero support for reasoning tasks. By intention, we did not try to include any subsumption hierarchy or axioms. The reason is that other preliminary experiments which we carried out show that only very simple ontology metamodels seem suitable for collaborative ontology building. We suspect that two major reasons are the prohibitive "cost" of learning complex ontology models and the lack of transparency of effects for the average user. A simple non-consensual `rdfs:subClassOf` statement can render the annotations of a multiplicity of users incorrect, and a simple modification of `rdf:domain` can lead to class memberships that are not intended. Our future research will focus on how this skeleton can be extended towards a richer ontology meta-model without introducing new entrance barriers for users. We think for example of clever voting mechanisms with thresholds that make a subsumption relationship subject to community voting.
Also, 5% of concepts that change their meaning over time means that some annotations will become corrupt over time. However, we regard this as a trade-off decision between ontology coverage in the sense of timely addition of needed concepts, and consistency. We think that we cannot prevent the Semantic Web to break here and there. It is important to recall that a core catalyst to the success of the Web was the willingness to accept inconsistencies and broken links in return for agility and distributed evolution.
In our opinion, the delegation of ontology building to a small "elite" group of ontology engineers is conceptually flawed, since the small group has no immediate access to the representational requirements and the conceptual preferences of the community members.

## 5. Conclusion

We have shown that standard Wiki technology can be easily used as an ontology development environment without modification, reducing entry barriers for the participation of users in the creation and maintenance of lightweight ontologies. On the basis of a quantitative analysis of current Wikipedia entries and their properties we have provided substantial evidence that the URIs of Wikipedia entries are surprisingly reliable identifiers for ontology concepts. In addition, we have demonstrated how the more than one million entries in Wikipedia can be used as ontology elements, opening this enormous source of named classes for making the Semantic Web a reality.

**Acknowledgements:** The work presented in this paper is partly funded by the European Commission under the project DIP (FP6-507483), the TransIT Entwicklungs- und Transfercenter at the University of Innsbruck, and Florida Gulf Coast University.

## References

[1] N. Guarino, "Formal Ontology and Information Systems," presented at FOIS '98, Trento, Italy, 1998.

[2] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, pp. 907-928, 1995.

[3] D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, 2nd ed. Berlin etc.: Springer, 2004.

[4] W3C, "OWL Web Ontology Language Guide. W3C Recommendation 10 February 2004," available at http://www.w3.org/TR/2004/REC-owl-guide-20040210/, retrieved Nov 30, 2005.

[5] M. Hepp, "A Methodology for Deriving OWL Ontologies from Products and Services Categorization Standards," presented at the 13th European Conference on Information Systems (ECIS2005), Regensburg, Germany, 2005.

[6] M. Hepp, "Representing the Hierarchy of Industrial Taxonomies in OWL: The gen/tax Approach," presented at the ISWC Workshop Semantic Web Case Studies and Best Practices for eBusiness (SWCASE05), Galway, Irland, 2005.

[7] J. Domingue, "Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web," presented at the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1998.

[8] J. Bao and V. Honavar, "Collaborative Ontology Building with Wiki@nt. A multi-agent based ontology building environment," presented at the 3rd International Workshop on Evaluation of Ontology-based Tools (EON2004), Hiroshima, Japan, 2004.

[9] Wikimedia Foundation, "Wikispecies," available at http://species.wikipedia.org/, retrieved Nov 30, 2005.

[10] Wikipedia, "Folksonomy," available at http://en.wikipedia.org/wiki/Folksonomy, retrieved Nov 30, 2005.

[11] A. Mathes, "Folksonomies -Cooperative Classification and Communication Through Shared Metadata," available at http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html, retrieved Nov 30, 2005.

[12] M. D. Lytras, "Semantic web and information systems: An agenda based on discourse with community leaders," *International Journal of Semantic Web and Information Systems*, vol. 1, pp. i-xii, 2005.

[13] W3C, "Simple Knowledge Organisation System (SKOS)," available at http://www.w3.org/2004/02/skos/, retrieved Nov 30, 2005.

[14] D. Vrandecic, S. Pinto, C. Tempich, and Y. Sure, "The DILIGENT knowledge process," *Journal of Knowledge Management*, vol. 9, pp. 85-96, 2005.

[15] S. E. Campanini, P. Castagna, and R. Tazzoli, "Platypus Wiki: a Semantic Wiki Wiki Web," presented at the 1st Italian Semantic Web Workshop Semantic Web Applications and Perspectives (SWAP), Ancona, Italy, 2004.

[16] A. Souzis, "Building a Semantic Wiki," *IEEE Intelligent Systems*, vol. 20, pp. 87-91, 2005.

[17] M. Hepp, D. Bachlechner, and K. Siorpaes, "OntoWiki: Community-driven Ontology Engineering and Ontology Usage based on Wikis," presented at the 2005 International Symposium on Wikis (WikiSym 2005), San Diego, California, USA, 2005.

[18] Wikipedia, "Criticism of Wikipedia," available at http://en.wikipedia.org/wiki/Criticism_of_Wikipedia, retrieved Nov 30, 2005.

[19] J. de Bruijn and J. Kopecký, "Persistent URIs for WSMO and WSML deliverables. WSMO Note 2 February 2005," available at http://www.wsmo.org/TR/NOTE-URIs/20050202/, retrieved Nov 30, 2005.
[20] T. Berners-Lee, "Cool URIs don't change," available at http://www.w3.org/Provider/Style/URI.html, retrieved Nov 8, 2004.
[21] Dublin Core Metadata Initiative, "Dublin Core Metadata Element Set, Version 1.1: Reference Description," available at http://dublincore.org/documents/dces/, retrieved Nov 30, 2005.
[22] Dublin Core Metadata Initiative, "DCMI Metadata Terms," available at http://dublincore.org/documents/dcmi-terms/, retrieved Nov 30, 2005.

# From Wikipedia to Semantic Relationships: a Semi-automated Annotation Approach*

Maria Ruiz-Casado, Enrique Alfonseca and Pablo Castells

Computer Science Department, Universidad Autonoma de Madrid, 28049 Madrid, Spain
{Maria.Ruiz,Enrique.Alfonseca,Pablo.Castells}@uam.es

**Abstract.** In this paper, an experiment is presented for the automatic annotation of several semantic relationships in the Wikipedia, a collaborative on-line encyclopedia. The procedure is based on a methodology for the automatic discovery and generalisation of lexical patterns that allows the recognition of relationships among concepts. This methodology requires as information source any written, general-domain corpora and applies natural language processing techniques to extract the relationships from the textual corpora. It has been tested with eight different relations from the Wikipedia corpus.

## 1 Introduction

Wikis are environments where uploading content to the web is extremely simple and does not require the users to have a technical background. Although wikis can be intended for individual use in applications such as Personal Information Management [1], this emergent area is producing many popular collaborative environments where web users are able to share and contrast their knowledge about a certain topic. Wikis have many applications, such as building collaboratively on-line information sites (e.g. dictionaries or encyclopedias) or for coordinating and exchanging information in project management or corporate intranets [2].

The success of many public wikis is due to the interest they have arisen among potential contributors, who are eager to participate due to their particular involvement in the domain under discussion. Dating from 1995, Wiki Wiki Web[1] is dedicated to software development. ProductWiki[2] is an on-line product catalogue. Wikitravel[3] is a world wide travel guide. Other specific domains can be found for instance at Comixpedia[4], a wiki for comics, or Wookipedia[5], that collects information about the Star Wars saga. Some general-domain wiki portals are the on-line encyclopedia Wikipedia[6], the dictionary Wiktionary[7] or the questions-answers portal Answerwiki[8].

* This work has been sponsored by MEC, project number TIN2005-06885
[1] http://c2.com
[2] http://productwiki.com
[3] http://wikitravel.org
[4] http://comixpedia.org
[5] http://starwars.wikicities.com
[6] http://wikipedia.org
[7] http://www.wiktionary.org/
[8] http://answerwiki.com

Its success is mainly due to several factors: the easiness to publish and review content on the web through the web browser, usually avoiding barriers such as logins or technical knowledge; the few restrictions on what a contributor can write; and the user-friendly interface which provides flexible mechanisms to get the content published. The philosophy behind wikis is to sum up the limited effort of each contributor to produce a repository of knowledge, the size of which depends on the number of contributors and the amount of information they are willing to provide. The Wiki platforms take advantage of the synergy of the aggregated work of their individual contributors.

Given that there is usually no authorship (and no responsibilities), concerns have been raised about the quality that can be attained in public wikis. A recent claim that the quality of scientific articles in Wikipedia is equivalent to the quality of the Encyclopaedia Britannica [3], has been reported to be flawed [4]. Anyway, wikis have shown that it is possible to write collaboratively very useful content, available to everyone with little personal cost. This characteristic is specially attractive for the Semantic Web field, where the need to place semantic annotations in existing and upcoming web content constitutes a costly hindrance.

### 1.1 Blending Semantic Web and Wikis

Most of the world wide web content is written in natural language and is intended for human readers. Due to the vast amount of information contained in the web, nowadays many tasks need a certain degree of automation. When trying to search and process web content automatically, a machine has to cope with language ambiguities and implicit knowledge that it can hardly process. The Semantic Web constitutes an initiative to extend the web with machine readable content and automated services far beyond the original capabilities of the World Wide Web [5], primarily making the web content explicit through semantic annotations. This would allow an easy automated processing when searching and retrieving information. Annotation standards have been developed by the Semantic Web community giving way to the RDF[9] and OWL[10] annotation languages, among other. The annotations refer to ontologies, a knowledge representation formalism that is used to model the underlying knowledge.

But placing semantic tags in the huge amount of existing and upcoming content can also be too costly if it has to be done manually by specialised ontologists. In 2004, the size of the Semantic Web, i.e. the percentage of web pages containing metadata, was estimated to be less than 5% of the total size of the web [6]. Therefore, as some authors have recently pointed out [2, 7, 8], initiatives like Wikis, that allow easy and collaborative information exchange, can be of great help in making the Semantic Web successful. Wiki communities have proved to succeed in collaboratively producing at low cost vast information repositories. The addition of semantic annotations to documents can be achieved following the Wiki philosophy of lowering the technical barriers: the semantic annotations are presented to the user as simply assigning a type to the hyperlinks, and providing internal facilities to transform these annotations into RDF or other Semantic Web annotation language. On the other hand, semantic annotation allows that

[9] http://www.w3.org/RDF/
[10] http://www.w3.org/2001/sw/WebOnt/

Wikis benefit from the automation of management, searching and retrieval pursued by the Semantic Web.

This blend between the Semantic Web and wiki fields is called *Semantic Wikis*. Some projects developing semantic wikis are IkeWiki[11] and Semantic Wikipedia[12].

### 1.2 Adding ingredients to the blend: Information Extraction

One step further in the creation of semantic annotations through a wiki environment is pointed out by some authors [7] as the possibility of assisting the user in the editing process to facilitate the selection of the typed links. For instance, the authoring environment may generate candidates for link types automatically, by extracting semantic information from the current Wikipedia corpus.

There is a large amount of information already present in the existing Wikipediae: as for March 2006, the English Wikipedia has more than one million articles. German, Spanish, French, Italian, Japanese, Dutch, Polish, Portuguese and Swedish Wikipediae are above one hundred thousand articles each. Using semi-automatic annotation procedures, it should be possible, in a short amount of time, to offer a fully enriched Semantic Wikipedia, which already has a large set of users willing to collaborate. A large-scale annotation may be the trigger to induce many people to switch from a traditional Wikipedia to a semantically annotated one. Therefore, the Semantic Wikipedia may be accelerated in its way to maturity.

The approach that we present in this paper addresses that point. We depart from our previous work in the extraction of semantic relationships to annotate the Wikipedia [9, 10], consisting in disambiguating Wikipedia encyclopedic entries with respect to WordNet, and determining four basic relationships amongst the concepts: hyponymy (*is-a*) and its inverse hyperonymy, and holonymy (*has-part*) and its inverse meronymy. In this paper, we extend the previous work to propose a methodology to automatically extract any other kind of relationship, and describe the results obtained when it is applied to the Wikipedia corpus. This procedure can be integrated in a tool that makes suggestions to users concerning where to place tags in Semantic Wiki documents.

We would like to point out that any automatic tagging procedure will produce some amount of mistakes. However, the work in correcting a few errors will always be smaller than the amount of work needed to create all the semantic tags from scratch. In the same way that current Wikipedia users, when they identify a mistake in an article, are quite content to edit the article and correct it, the users of a semantic Wikipedia could do the same. Rather than departing from documents containing plain text and a few examples of relations, they would start with the full text of the Wikipedia, containing many semantic relationships, and, while reading the articles, they would just have to correct the wrong labels as they find them.

This paper is structured as follows: Section 2 introduces our approach to the automated extraction of patterns from textual sources; Sections 3 details the approach followed to identify new relations; Section 4 describes the experiment conducted with

[11] http://ikewiki.salzburgresearch.at/

[12] http://wiki.ontoworld.org

Wikipedia and the results obtained; finally, Section 6 concludes and points out open lines for future work.

## 2 Extracting Relationships from text: our approach

Our goal is to find semantic relationships and properties (attributes) in free text automatically. To do so, we have developed an approach based in automatic learning of lexico-syntactic patterns. The procedure starts with a seed list containing pairs of related items. It may be a list containing writers and some of their works; painters and their pictures; or soccer players and the clubs in which they have played. Next, many sentences containing the pairs of terms from the seed list are collected automatically from the web, and they are processed with the following Natural Language Processing (NLP) tools:

- Segmentation (tokeniser and sentence splitter).
- Part-of-speech tagging, i.e. identifying which words are nouns, verbs, adjectives, etc.
- Stemming, to obtain the canonical form of all the words.
- Named Entity Recognition, to identify dates, numbers, people, locations and organisations.
- Chunker (partial syntactic analyser).

The information obtained from the processed sentences can be used to study which words, syntax and entities are typically used in a human language when a particular relation between two concepts is expressed. To do so, we search in the context that surrounds the two concepts in order to find repetitive lexical patterns that appear with the concepts when the relation is present. A pattern models a possible way to convey a semantic relation in natural language, and can be applied to search and extract new pairs of concepts between which the same relation holds.

Figure 1 shows an overview of the process carried out for each type of relationship. The final version of our system is intended to take an XML dump of the English wikipedia and to produce an equivalent file with the semantic annotations added. In this way, the result can be seen directly on a web browser using standard Semantic Wikipedia software.

The following subsections elaborate on the separate steps. A complete technical exposition of the procedure can be found in [10, 11].

### 2.1 Collecting contextual patterns from the web

Given a pair of related terms appearing in a text, the context of this pair is the text fragment that encloses them. The context boundaries are sometimes expressed as a window of $N$ words to the left and to the right of those two terms, or as a syntactic constituent (e.g. a sentence) containing them both.

For the task that we address in this paper, extracting semantic relationships between words, the context can be very useful. For example, in (1) it can be seen that *tail* is a part of *dog*, because of the possessive pronoun *its*. In the context, the possessive pronoun

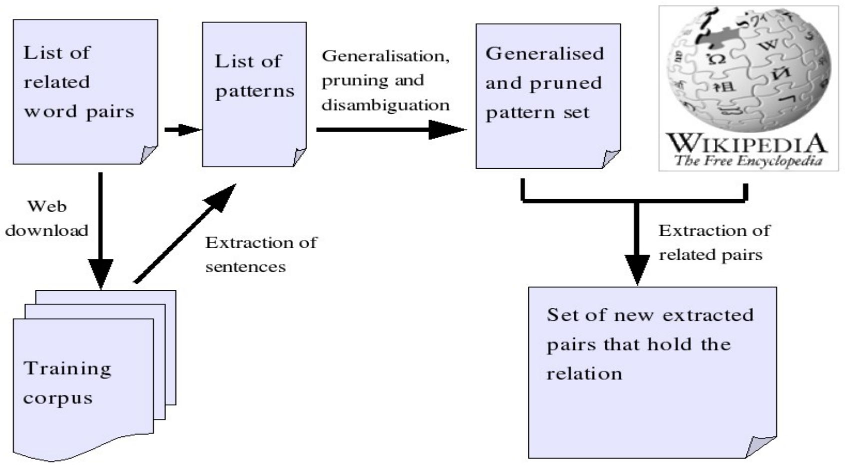

List of related word pairs
List of patterns
Generalisation, pruning and disambiguation
Generalised and pruned pattern set
WIKIPEDIA
The Free Encyclopedia
Web download
Extraction of sentences
Extraction of related pairs
Training corpus
Set of new extracted pairs that hold the relation

**Fig. 1.** Overview of the procedure.

indicates the existence of a holonymy (part-of) relationship. Also, the verb *composed* in Sentence (2) indicates a relationship *composer-song* between *John Lennon* and *Imagine*.

(1) The happy dog wagged its tail.

(2) The piano on which John Lennon composed "Imagine" is on its way back to the Beatles Story in Liverpool.

In some fields, such as Word-Sense Disambiguation and Information Retrieval, these contexts are usually characterised using a *bag-of-words* approach, where all the words in the context are put together regardless of their relative ordering or syntactic dependencies. In our approach, as we are interested in finding patterns of words that model a relationship, we keep the relative ordering of the words.

A relationship is formed by a triple: the two concepts related and the relationship itself. When the relation is modelled as a pattern, the two concepts participating are usually called *hook* and *target* [12]. So, in the previous example, the relation *composer-song* can be modelled with the pattern (3).

(3) The piano on which *hook* composed *target* is on its way back to the Beatles Story in Liverpool.

As we mentioned before, we start with a seed list containing related pairs. The patterns can be collected, for each relationship, using this list: for each pair of terms,

1. Perform a search on the Internet containing the two terms in the pair.
2. Process with the NLP tools all the sentences that contain the two terms.
3. Substitute the first term by *hook*, and the second term by *target*.

### 2.2 Pattern Generalisation

The patterns of words with which natural languages can express semantic relationships are usually manifold. Therefore, for each kind of relationship, we need to capture all this lexical variability. The purpose is to group similar patterns to obtain one that is general enough to match different contexts amongst which there are only small differences in paraphrasing. Given that many of the patterns collected in the previous step will have shared parts, that information can be used to guide the generalisation.

The core idea is the following: to generalise two original patterns, the common parts are maintained, and the differences are substituted either by disjunctions or wildcards. For instance, from Sentences (4a,b), the patterns (5a,b) are extracted, and the generalisation (6) can be obtained. The star is wildcard (representing any word), and the vertical bar | means a disjunction, meaning that any of the two adjectives can be found before the target.

(4) a. Alfred Hitchcock directed the famous film Psycho
  b. Alfred Hitchcock directed the well known film Psycho

(5) a. *hook* directed the famous film *target*
  b. *hook* directed the well known film *target*

(6) *hook* directed the * famous|known film *target*

This pattern can detect the relation director-film and determine the participating concepts in many sentences, e.g. (7a,b,c).

(7) a. Alfred Hitchcock directed the famous film The Birds
  b. Bernardo Bertolucci directed the well known film The Last Emperor
  c. Woody Allen directed the amusing and famous film Annie Hall

Note that in this example above, pattern (6), has been obtained from two patterns, one containing the word *famous*, and the other containing the words *well known*, but it is not the only possible generalisation. In total, using disjunctions and wildcards, from patterns (5a,b) the following generalisations are plausible:

- Substituting them all with the wildcard, *.
  (8) *hook* directed the * film *target*
- Creating a disjunction *famous|well* followed by the wildcard *.
  (9) *hook* directed the famous|well * film *target*
- Creating the disjunction *famous|known*, preceded by the wildcard *.
  (10) *hook* directed the * famous|known film *target*

We believe that the third one is the most suited to this task, as the terms in the disjunction both undertake the same role in the sentence, as modifiers of the target, while *well* is an adverb modifying the adjective *known*. This can be partly identified automatically if the generalisation procedure takes into account the part-of-speech (PoS) of the words. In this way, during the generalisation step, we only allow for disjunctions of words that share the same part-of-speech.

### 2.3 Patterns Pruning

In the previous subsection, it is still not clear when to stop generalising. For instance, given that the three sentences in (11) do not have any word in common, their generalisation would be pattern (12), which is clearly unusable because it matches everywhere.

(11) a. *hook* directed the famous film *target*.
  b. *hook* won an Oscar with his film *target*.
  c. *hook* 's movie *target*.

(12) *hook* * *target*

In general, very specific and long patterns tend to be very accurate when extracting new relations from text, as they impose many restrictions to the contexts they match. But they are applicable in few cases and hardly extract new relationships. On the contrary, short and very general patterns have a good recall because they match frequently in the texts, but their precision may be low, extracting wrong relationships. It is desirable, therefore, to evaluate the set of patterns as they are generalised, in order to stop as soon as the result of a generalisation produces too many results with a low precision.

A possible way to calculate the precision of the generalised patterns is the one described in [12]. In this approach, after all the patterns have been collected and all their possible generalisations calculated, a development corpus is collected again from the web, searching for pages that contain the hooks from the seed list. The precision of each pattern is estimated as the percentage of times that it extracts, from the development corpus, a pair existing in the seed list.

In our approach [11], we used an improved evaluation methodology that mainly consists in testing the patterns, not only for a development corpus collected with their original hooks, but also for the development corpora collected with the hooks from different relationships, which altogether constitutes what we call the *development super-corpus*. The performance of each individual pattern for each particular relation is tested in the super-corpus.

Based on the scores calculated through this automatic evaluation, the set of patterns can be refined discarding those that do not reach a predefined threshold in precision, or that are not able to match in the super-corpus a minimum number of times.

### 2.4 Pattern Disambiguation

Up to this point, the patterns have been extracted and generalised separately for each relationship. Hence, we have some patterns that, supposedly, appear between writers and their works, or between people and their birth place. The problem is that it may be the case that the same pattern is extracted for different relations. This is specially relevant in the case of a few short patterns such as the genitive construction (13). As can be seen in Sentences (14a-e), it can be used to convey semantic relationships as dissimilar as *scientist-theory*, *painter-painting*, *writer-work*, *architect-work* or *location-sublocation*, among many others.

(13) *hook* 's *target*

(14) a. Einstein's Theory of General Relativity
  b. Bosco's The Garden of Delights
  c. Tolkien's Lord of the Rings
  d. Gaudi's Sagrada Familia
  e. Barcelona's Sagrada Familia

In order to overcome this difficulty, we propose two solutions:

- To take into account the Named Entity (NE) tag of the hook and the target, whenever it has been annotated by the NE recogniser during the NLP processing. In this way, people, locations, organisations and dates would be marked as such in the examples above. If we annotate that a pattern is only applicable if the *hook* is of type *location*, then it will only apply to Sentence (14e) and not to Sentences (14a-d). This can be used to somewhat mitigate the problem.
  Though not completely error-free, patterns including NER are more expressive and present a better ability to differentiate relationships.
- Even so, it may be the case that a pattern still appears in the lists of patterns for different relationships. Currently, many of these patterns are ruled out in the pruning step, because we are taking into consideration the test corpora from all the relationships when we build the *development super-corpus*. Hence, if pattern (13), extracted for the relationship *scientist-theory*, is applied to the test corpus collected for *author-work*, it will erroneously mistag all the book as scientific theories, which will be detected because they do not appear in the seed list for theories, and the precision of the rule will be low.

## 3 Extraction Procedure

Once we have a set of patterns for each semantic relationship, the extraction procedure is simple. Each pattern will contain:

- A flag indicating whether the hook appears before the target, or vice-versa.
- A left-hand part, consisting of the words at the left of the hook or the target, whichever appears first.
- A middle part, with the words between the hook and the target.
- A right-hand part.

Given a set of patterns for a particular relation, the procedure to obtain new related pairs is as follows:

1. Download the corpus that should be annotated from the web.
2. Clean the HTML code, remove menus and images
3. Process the textual corpus with NLP tools:
   (a) Tokenise the text and split sentences.
   (b) Apply a part-of-speech tagger.
   (c) Stem nouns and verbs.
   (d) Identify Named Entities.
   (e) Chunk Noun Phrases.

4. For every pattern,
   (a) For each sentence in the corpus,
      i. Look for the left-hand-side context of the pattern in the sentence.
      ii. Look for the middle context.
      iii. Look for the right-hand-side context.
      iv. Extract the words in between, and check that either the sequence of PoS tags or the entity type are correct. If so, output the relationship.

The above procedure is repeated for all the relations considered.

## 4 Experiment and results

### 4.1 Experimental settings

The experiment carried out consists in extracting several relationships and properties from a test corpus downloaded from the English Wikipedia. The relations and properties considered are:

– Person's birth year
– Person's death year
– Person-birth place
– Actor-film
– Writer-book
– Football player-team
– Country-chief of state
– Country-capital

In order to collect the corpus of the Wikipedia, to ensure that many entries contain the indicated relationships, we have performed a recursive web download starting from the following entries:

– *Prime Minister*, that contains hyperlinks to Prime Ministers from many countries.
– *Lists of authors*, that contains hyperlinks to several lists of writers according to various organising criteria.
– *Lists of actors*, that contains hyperlinks to several lists of actors.
– *List of football (soccer) players*, containing hyperlinks to many entries about players.
– *List of national capitals*, containing the names of national capitals from countries in the world.

From those initial pages, all the hyperlinks have been followed up to a depth of 3–4 hyperlinks, having collected in total 20,075 encyclopedia entries totalling roughly 460 Megabytes after cleaning the HTML files. The NLP toolkit used to process them was the Wraetlic tools v. 2.0[13].

The pruned patterns for the mentioned relations have been produced using the procedure described above. For this experiment, only those patterns showing a precision

[13] Available at http://www.eps.uam.es/∼ealfon/eng/research/wraetlic.html

| Pattern |
|---|
| `On time_expression TARGET HOOK was baptized\|born` |
| `" HOOK ( TARGET -` |
| `-\|-- HOOK ( TARGET -` |
| `AND\|and\|or HOOK ( TARGET -` |
| `By\|about\|after\|by\|for\|in\|of\|with HOOK TARGET -` |

**Table 1.** Some of the patterns obtained for the relationship *birth year*.

.

| Relation | No. of patterns | No. of results | Precision |
|---|---|---|---|
| Birth-year | 16 | 15746 | 74.14% |
| Death-year | 8 | 5660 | 90.20% |
| Birth-place | 3 | 154 | 27.27% |
| Actor-film | 11 | 4 | 50.00% |
| Country-Chief of state | 109 | 272 | 50.00% |
| Writer-book | 176 | 179 | 37.29% |
| Country-capital | 150 | 825 | 11.45% |
| Player-team | 173 | 315 | 7.75% |

**Table 2.** Number of patterns obtained for each relationship, number of results extracted by each pattern set, and precision.

higher than 0.90 in the development super-corpus and that matched at least 3 times are used. Table 1 shows some of the patterns obtained for the property *birth date*. In the second column of Table 2 the total number of patterns that were finally obtained for each of the semantic relationships under consideration is shown.

## 4.2 Results and discussion

Table 2 shows the number of results (pairs of related terms) that each of the pattern sets has extracted, and the precision attained. This precision has been estimated by correcting manually least 50 results from each relationship.

As can be seen, the precision for birth year and death year is very good, because they are usually expressed with very fixed patterns, and years and dates are entities that are very easily recognised. The few errors are mainly due to the following two cases:

– Named Entity tagging mistakes, e.g. a TV series mistagged as a person, where the years in which it has been shown are taken as birth and death date.
– Names of persons that held a title (e.g. king or president) during a period of time, that is mistakenly considered their life span.

On the other hand, as expected, the other examples have proven more difficult to identify. We have observed the problem, mentioned in the previous sections, that some patterns are applicable for many kinds of relationships at the same time. This phenomenon is specially relevant in the case of the *player-team* relation. The precision of the patterns is 92% when they are applied only to the entries about soccer players, but the figure falls down to 7.75% when applied to the whole Wikipedia corpus collected.

This means that they are patterns that, in the domain of soccer, usually indicate the relationship between the player and its club, but in other contexts they may be conveying a different meaning. One of these patterns is the already mentioned genitive construction. In sports articles, when this construction is found between an organisation and a person is usually expressing the *player-team* relation, as in *Liverpool's Fowler*. But it also extracted many wrong pairs from documents belonging to different topics.

The same also applies to the case of countries and capitals. During training, from phrases such as *Spain's Madrid*, the system extracted the genitive construction as indicating a relationship of capitality, but it is a source of errors because it can also express a part-of relationship between a country and any of its cities.

In the case of *actor-film*, we have observed that in the actors' entries in the Wikipedia, there is usually a section containing all the filmography, expressed as an HTML bullet list. In this way, because the information is already semi-structured, the textual patterns cannot apply. It should be easier to extract that data using other simpler procedures that take benefit of the structure of the entry.

### 4.3 Automatically generated list pages

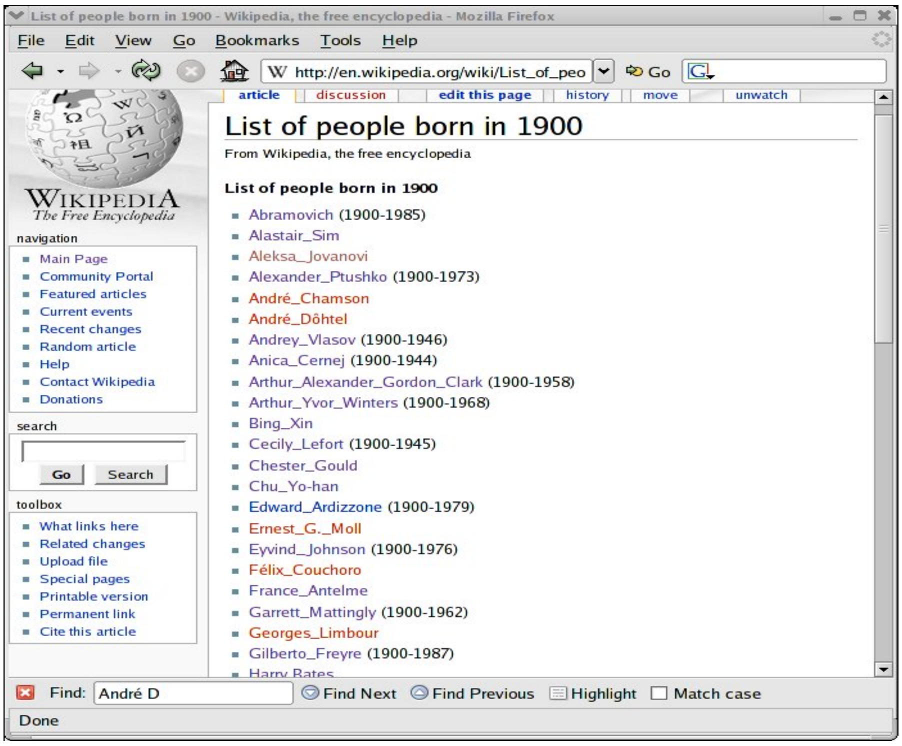
To test a possible application of this procedure in a real case, the algorithm has been used to generate automatically a list page using the data collected from Wikipedia. The example chosen is the list of people born the year 1900. A total of 57 people had been annotated with that birth date. The system was able to obtain the list automatically and, in the cases in which the death date was available as well, it was added as additional information. A manual evaluation of the generated list revealed the following:

– Two out of the 57 people were not people, due to errors in the Named Entity recogniser. These were eliminated by hand.
– One birth date was erroneous, and it was really the date of publication of a book about that person. That entry was also removed.
– One person had been extracted with two different (but correct) spellings: Julian Green and Julien Green, so they were merged in one.
– From the remaining 53 people, 14 did not have an associated Wikipedia entry. They had been extracted from other lists in which they were mentioned, but their biography is not available yet. These were left inside the list, as it is useful information.
– Finally, three people in the list had ambiguous names, so they were directed to disambiguation pages. It was easy to modify the hyperlink so they pointed directly to the particular person with that name that had been born in 1900.

Figure 2 shows a possible layout for the generated page. Note that, in the Wikipedia, there are special categories to group all the people born every year. So, the category called *Category:1900_birth* contains all the people for which someone has categorised their entries as having been born in 1900.

Before our experiment was performed, this category contained 640 people[14] born in that year, all of them with an entry in the Wikipedia. From our list, we have been

[14] Note that, in our experiment, we do not process the whole English wikipedia (more than one million entries) but a small subset containing around 20,000 entries.



List of people born in 1900

From Wikipedia, the free encyclopedia

**List of people born in 1900**

- Abramovich (1900-1985)
- Alastair_Sim
- Aleksa_Jovanovi
- Alexander_Ptushko (1900-1973)
- André_Chamson
- André_Dôhtel
- Andrey_Vlasov (1900-1946)
- Anica_Cernej (1900-1944)
- Arthur_Alexander_Gordon_Clark (1900-1958)
- Arthur_Yvor_Winters (1900-1968)
- Bing_Xin
- Cecily_Lefort (1900-1945)
- Chester_Gould
- Chu_Yo-han
- Edward_Ardizzone (1900-1979)
- Ernest_G._Moll
- Eyvind_Johnson (1900-1976)
- Félix_Couchoro
- France_Antelme
- Garrett_Mattingly (1900-1962)
- Georges_Limbour
- Gilberto_Freyre (1900-1987)
- Harry_Bates

**Fig. 2.** A possible layout for the page generated automatically containing people that were born in 1900, after a manual correction was performed.

able to identify and categorise four people born in that year that were still not inside the category, and 14 people that are not listed because yet nobody has written their entries.

In the same way that this list has been gathered, it should be easy to create or extend list pages using other criteria, such as famous people born in a certain city, or people that died at a certain age. If the Wikipedia were extended with semantic links, then all these list pages would not need to be stored as static pages inside the Wikipedia server, but ideally they would be generated on-the-fly using the semantic metadata.

## 5 Related work

To our knowledge, there is no other work reported addressing the task of annotating semi-automatically wiki content for the Semantic Web. However, there is already much research on automatically identifying relationships in unrestricted text. In particular, the use of lexical or lexicosyntactic patterns to discover ontological and non-taxonomic relationships between concepts has been proposed by [13–15], all of whom manually define regular expressions to extract hyponymy and part-of relationships. [16] learns patterns that express company merge relationships. [17] quantifies the error rate of a similar approach for hyponymy relationships at 32%.

Systems that learn these lexical patterns from the web have the advantage that the training corpora can be collected easily and automatically. Several approaches have been proposed recently [18, 19, 12], having various applications in mind: Question-Answering [12], multi-document Named Entity Coreference [20], and the generation of biographical information [21].

Other systems that automatically extract ontological knowledge from text are Text-ToOnto [22], OntoLT [23] and KIM [24].

## 6 Conclusions and Future Work

In this paper, we propose the use of Natural Language Processing techniques to automatically extract semantic relationships from the Wikipedia. We have shown that, for some relationships, the precision obtained is acceptable, and with a brief manual revision good-quality metadata can be obtained.

We believe that these procedures can contribute in increasing the size of current Semantic Wikis semi-automatically. Even though it will always be necessary a manual revision to identify the wrong results, the work involved in correcting the generated metadata is smaller than creating it all from scratch, as we have shown with the example of generating list pages.

Furthermore, we foresee applications of this work for automatic ontology building and population, as hinted by [7].

Concerning future work, we plan to continue exploring ways to improve the precision of the patterns for the relationships with poorer performance. In particular, the use of patterns that may express different relationships, depending on the context, needs to be further enhanced.

We also plan to try this procedure with yet more relationships, and in other languages.

## References

1. Kiesel, M., Sauermann, L.: Towards semantic desktop wiki. UPGRADE special issue on The Semantic Web **6** (2005) 30–34
2. Schaffert, S., Gruber, A., Westenthaler, R.: A semantic wiki for collaborative knowledge formation. In: Proceedings of SEMANTICS 2005 Conference., Vienna, Austria (2005)
3. Giles, J.: Internet encyclopaedias go head to head. Nature, Special Report **438** (2005) 900–901
4. Britannica, E.: Fatally flawed: Refuting the recent study on encyclopedic accuracy by the journal nature (2006)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web - a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American **284** (2001) 34–43
6. Baeza-Yates, R.: Mining the web. El profesional de la información **13** (2004) 4–10
7. Krötzsch, M., Vrandecic, D., Völkel, M.: Wikipedia and the semantic web - the missing links. In: Proceedings of WIKIMANIA 2005, 1st International Wikimedia Conference. (2005)

8. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic wikipedia. In: Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland (2006)
9. Ruiz-Casado, M., Alfonseca, E., Castells, P.: Automatic assignment of wikipedia encyclopedic entries to wordnet synsets. In: Proceedings of the Atlantic Web Intelligence Conference, AWIC-2005. Volume 3528 of Lecture Notes in Computer Science. Springer Verlag (2005) 380–386
10. Ruiz-Casado, M., Alfonseca, E., Castells, P.: Automatic extraction of semantic relationships for wordnet by means of pattern learning from wikipedia. In: Natural Language Processing and Information Systems. Volume 3513 of Lecture Notes in Computer Science. Springer Verlag (2005) 67–79
11. Alfonseca, E., Castells, P., Okumura, M., Ruiz-Casado, M.: A rote extractor with edit distance-based generalisation and multi-corpora precision calculation. In: In Proceedings of the Poster Session of ACL-2006. (2006)
12. Ravichandran, D., Hovy, E.: Learning surface text patterns for a question answering system. In: Proceedings of the ACL-2002. (2002) 41–47
13. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of COLING-92, Nantes, France (1992)
14. Hearst, M.A.: Automated discovery of wordnet relations. In: Christiane Fellbaum (Ed.) WordNet: An Electronic Lexical Database. MIT Press (1998) 132–152
15. Berland, M., Charniak, E.: Finding parts in very large corpora. In: Proceedings of ACL-99. (1999)
16. Finkelstein-Landau, M., Morin, E.: Extracting semantic relationships between terms: supervised vs. unsupervised methods. In: Workshop on Ontologial Engineering on the Global Info. Infrastructure. (1999)
17. Kietz, J., Maedche, A., Volz, R.: A method for semi-automatic ontology acquisition from a corporate intranet. In: Workshop "Ontologies and text", co-located with EKAW'2000, Juan-les-Pins, France (2000)
18. Brin, S.: Extracting patterns and relations from the World Wide Web. In: Proceedings of the WebDB Workshop at the 6th International Conference on Extending Database Technology, EDBT'98. (1998)
19. Agichtein, E., Gravano, L.: Snowball: Extracting relations from large plain-text collections. In: Proceedings of ICDL. (2000) 85–94
20. Mann, G.S., Yarowsky, D.: Unsupervised personal name disambiguation. In: CoNLL-2003. (2003)
21. Mann, G.S., Yarowsky, D.: Multi-field information extraction and cross-document fusion. In: ACL 2005. (2005)
22. Maedche, A., Staab, S.: Semi-automatic engineering of ontologies from text. In: Proceedings of the 12th Internal Conference on Software and Knowledge Engineering, Chicago, USA (2000)
23. Buitelaar, P., M. Sintek, M.: OntoLT version 1.0: Middleware for ontology extraction from text. In: Proc. of the Demo Session at the International Semantic Web Conference. (2004)
24. Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., Kirilov, A.: Kim - a semantic platform for information extaction and retrieval. Journal of Natural Language Engineering **10** (2004) 375–392

# Extracting Semantics Relationships between Wikipedia Categories

Sergey Chernov, Tereza Iofciu, Wolfgang Nejdl, and Xuan Zhou

L3S Research Centre, University of Hannover, Expo Plaza 1, 30539, Hannover, Germany,
{chernov, iofciu, zhou, nejdl}@l3s.de

**Abstract.** Wikipedia is the largest online collaborative knowledge sharing system, a free encyclopedia. Built upon traditional wiki architectures, its search capabilities are limited to title and full-text search. We suggest that semantic information can be extracted from Wikipedia by analyzing the links between categories. The results can be used for building a semantic schema for Wikipedia which could improve its search capabilities and provide contributors with meaningful suggestions for editing the Wikipedia pages. We analyze relevant measures for inferring the semantic relationships between page categories of Wikipedia. Experimental results show that inlinks provide a better evidence of semantic connection in comparison to outlinks.

## 1 Introduction

The Wikipedia [1] is a freely accessible Web encyclopedia. The Wikipedia project started in 2001 as a complement to the expert-written Nupedia and it is currently run by the Wikipedia Foundation. There are Wikipedia versions in 200 languages, with more than 3,700,000 articles and 760,000 registered users. It is built on the expectation that collaborative writing improves articles over time. Users do not need to be experts on the topics they are writing on, but they are warned that their contributions can be modified by anyone, all the modifications are logged and the information about the evolution of each page is available online. Because of its writing openness, "edit wars" and disputes may easily appear when users do not reach an agreement on a given topic. The Wikipedia project uses Wiki software, which was invented by Ward Cunningham [8] in 1995. The English Wikipedia version is the world's largest wiki, followed by the German Wikipedia version.

An especially interesting property of Wikipedia is the semantic tagging and linkage of its content. Pages are heavily interlinked and many of them explicitly assigned to one or more semantic *Categories*. Categories should represent major topics and their main use within Wikipedia is in finding useful information. There are two types of categories. The first type is used for classification of pages with respect to topics. They can have hierarchical structure, for example the page can be assigned to the category *Science* or one of its subcategories like *Biology* and *Geography*. The second type of categories is *Lists*, they usually contain links to instances of some concept, for example *List of Asian Countries* points to 54 Asian countries. Each page may be assigned to several categories.

Like in most of the wikis, the search capabilities on Wikipedia are limited to traditional full-text search, while search could benefit from the rich Wikipedia semantics and may allow complex searches like *find Countries which had Democratic Non-Violent Revolutions*. Using categories as a loose database schema, we can enrich Wikipedia search capabilities with such complex query types. Wikipedia categories could be organized in a graph, where the nodes are categories and the edges are hyperlinks. For example, if some page from the category "Countries" points to a page from the category "Capitals" we can establish a connection "Countries to Capitals". However, not all hyperlinks in Wikipedia are semantically significant such that they can be used to facilitate search. The problem is how to distinguish strong semantic relationships from irregular and navigational links.

In this paper we propose two measures for automatic filtering of strong semantic connections between Wikipedia categories. They include number of links between categories and Connectivity Ratio, which could be applied to inlinks or outlinks. For evaluation, we apply these heuristics to the English Wikipedia and perform user study to assess how semantically strong the extracted relationships are. We observe that, for a given category, inlinks provide good evidence for semantic relationships, while outlinks have poor performance.

The rest of the paper is organized as follows. The related work is given in Section 2. In Section 3 we describe in details the problem of discovering strong semantic relationships between categories and the possible use of semantic scheme in Wikipedia. Later, in Section 4 we describe our analysis of factors, relevant for discovering semantic links and present our experiments in Section 5. We conclude and outline future research directions in Section 6.

## 2 Related Work

The idea to bring semantics into Wikipedia is not new, several studies on this topic have been carried out in the last few years.

The semantic relationships in Wikipedia were discussed in [7]. The authors considered the use of link types for search and reasoning and its computational feasibility. Its distinctive feature is the incorporation of semantic information directly into wiki pages. Later, the semantic links proposal was extended in [10] to the Semantic Wikipedia vision. According to this model, the pages annotations should contain the following key elements: categories, typed links, and attributes. Typed links in form of *is capital of* are introduced via markup extension [[is capital of::England]], each link can be assigned multiple types. They also proposed the usage of semantic templates, based on the existing Wikipedia templates. We follow this approach, but concentrate on automatic extraction instead of manual link assignment. Also, our goal is to enable better search on Wikipedia, but not to provide means for full-fledged reasoning. So we can tolerate higher level of inconsistency in annotations and use ill-defined schemas. The system for semantic wiki authoring is presented in [2]. It aids users in specifying link types, while entering the wiki text. This approach considers ontology-like wiki types, using "is a" or "instance of" relationship types. Since the prototype supports manual editing, it does not discuss automatic relationship assignment. Our approach can be used as an additional feature in this system.

One of the first attempts to automatically extract the semantic information from Wikipedia is presented in [6], which aims at building an ontology from Wikipedia collection. This work focus on the extraction of categories using links and surrounding text, while we aim at extracting semantic links using assigned categories. The paper [5] shows the importance of automatic extraction of link types, and illustrates several basic link types, like synonyms, homonyms, etc. It also suggests to use properties for dates and locations. However, it does not propose any concrete solutions or experimental results. Studies of *history flow* in Wikipedia are presented in [9]. The work is focused on discovering collaboration patterns in page editing history. Using an original visualization tool they discovered editing patterns like statistical corroboration, negotiation, authorship, etc. This work does not consider semantic annotation of Wikipedia articles.

The link structures in Wikipedia have been studied recently. The work from [11] presents an analysis of Wikipedia snapshot on March 2005. It shows that Wikipedia links form a scale-free network and the distribution of in- and outdegree of Wikipedia pages follow a power law. In [3] authors try to find the most authoritative pages in different domains like *Countries*, *Cities*, *People*, etc., using PageRank and HITS algorithms. It is reported in the paper, that Wikipedia forms a single connected graph without isolated components or outliers.

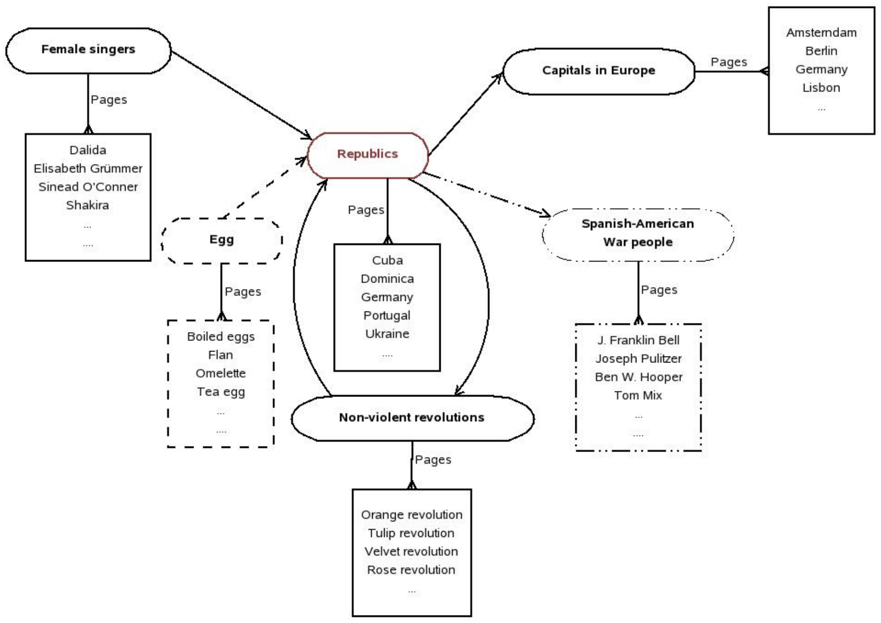
## 3 Problem

The usage of semantic links can be illustrated by the example we have mentioned in Section 1. Consider the query *find Countries which had Democratic Non-Violent Revolutions*. When we search in full-text for *Country Revolution Democracy* we get a lot of pages, which contain all the keywords, but most of them do not talk about particular countries. In a database-like view, the target page of our query should belong to the *Countries* category, and it should have a connection to a page in the category *Revolutions* which mentions the word *Democracy*. In current Wikipedia, there is actually a link between the pages *Ukraine* and *Orange Revolution*. If we put into a separate inverted list[1] all pages with *Country to Revolution* link type, we can force the previous query to return more relevant results.

However, it is infeasible to maintain and index all possible links between Wikipedia categories. An example of typical Wikipedia linkage between categories is shown in the Fig.3. Ovals correspond to categories, squares contain the lists of pages and arrows show existence of at least on hyperlink between categories. The category *Republics* is pointed by the *Female Singers*, *Egg*, and *Non-violent Revolutions* categories. It also points to *Capitals in Europe*, *Spanish-American War People* and *Non-violent Revolutions* categories. Some of these links can be converted into strong semantic relationships, like "*Republics to Non-violent Revolutions*" categories, while relationships like "*Egg to Countries*" are not regular semantic connections and only used for navigation or some unimportant purposes. It is useless to type and index such "LinkSourceCatergory to LinkTargetCategory" relationships, as they cannot help users in search. Instead,

[1] Inverted indices are used in information retrieval for keyword search, for details see [12]

we need to filter out unimportant links and extract semantically significant relationships from Wikipedia. This could be achieved by analyzing the link density and link structures between the categories.



Besides indexing purposes, the prominent semantic relationships can be of use for template generation and data cleaning. For example, if we have some pages in *Countries* without link to pages in *Capitals*, the system could suggest users to add missing link. One may want to create more precise link types and distinguish between type "Country has Capital" and "Country changed Capital", but this task is much more challenging and it is not the focus of this paper, we concentrate on selecting only coarse-grained semantic relationships.

## 4 Approach to Extracting Semantic Links

This section presents our approaches to extract semantically important relationships from the links in Wikipedia. This task can be seen as an automatic construction of a database schema, where we want to emphasize the meaningful relations between categories and disregard unimportant ones.

It seems reasonable, that highly connected categories represent strong semantic relations. For example, if a considerable percentage of pages from category "Country" has links to category "Capital", we can infer that there must be a "Country to Capital" relationship between the two instances categories. On the other hand, if there are only several links between two categories like “Actor” and “Capital”, it seems that there is no regular semantic relationship like “Actor to Capital”.

We conduct experiments to test this filtering method. In the experiments, we extract a core set of pages which have a common topic (in our case the common topic is *Countries*). For this pages we extract all the categories they belong to, and also two lists of categories, one for the pages with links toward *Countries* (inlink pages) and one for the pages referred by *Countries* (outlink pages). The experiments with these lists can give an idea about what link direction is more important for semantic relationship discovery. During the experiments we test two measures used for finding the strong semantic connections:

1. **Number of links between categories**. The more links we have between categories, the stronger should their semantic connection be. We study separately the effect of outgoing links and incoming links, so only links in one direction are considered each time.
2. **Connectivity Ratio**. We can normalize the number of links with the category size, to reduce the skew toward large categories. We call this normalized value *Connectivity Ratio*, and it represents the percentage of linkage between two sets (in one direction). Namely
$$ConnectivityRatio_i = \frac{NL_{ij}}{NP_i}$$
where $NL_{ij}$ is the number of links from category $i$ to category $j$[2], and $NP_i$ is the total number of pages in category$_i$.

## 5 Experimental Studies

In this section we describe our experimental setup and discuss the results.

### 5.1 Collection

For experiments we used the Wikipedia XML corpus [4] which is available for the participants of INEX 2006 evaluation forum[3]. This corpus is based on the English Wikipedia dump and has about 668,670 pages. We exported the dataset into a MySQL 4.1 database, the data size was about 1,2 Gigabytes . The pages belong to 63,879 distinct categories[4]; only pages from article namespace are included. Some of the pages in the dataset were empty, so after an export we got a slightly different corpus statistics, from the one reported in [4].

For the experiments we selected three sets of pages, which we called *Countries*, *Inset* and *Outset*. The *Countries* set consists of 257 pages devoted to countries, they

[2] In current experiments $j$ always corresponds to a *Countries* set.

[3] `http://inex.is.informatik.uni-duisburg.de/2006/`

[4] Some categories names differ only by space character before the names, or slightly different spelling. Our experimental setup does not assume use of NLP techniques, so we did not remove these inconsistencies and treated these categories as distinct.

were manually extracted from the "List of countries" Wikipedia page, this set represents the *Countries* category. We also built the *Inset*, which contains all Wikipedia pages that point to any of the pages in the *Countries*, and *Outset* contains pages pointed from the *Countries*. The statistics summary for the selected sets is presented in Table 1.

| | # of pages | # of assigned categories |
|---|---|---|
| Countries | 257 | 405 |
| Inset | 289,035 | 60,277 |
| Outset | 30,921 | 14,587 |
| Total (distinct entries) | 290,893 | 63,879 |

**Table 1.** The Statistics from Experimental Collection

Each page consists of the name of the page, a list of associated categories, and a list of links that can be internal links (pointing to Wikipedia pages) or external links (pointing to pages from the Web), for experiments we only considered internal links. We also maintain separately the information about links between pages and the connection between pages and categories.

## 5.2 Results

The main evaluation criteria for our task is the quality of extracted semantic relationships. To enable quantitative comparison between semantic connection we introduced the **Semantic Connection Strength** measure (**SCS**). It receives 0, 1, or 2 values, where grade 2 corresponds to a strong semantic relationship, value 1 to average and 0 indicates weak semantic connection[5]. Two assessors were given the following instruction: "category A is strongly related to category B (value 2) if every page in A should have at least one link to B. It should be considered as an average relationship (value 1), if 50% of pages in A should have at least one link to B and should be given weak relationship (value 0) otherwise." It turned out, that it is difficult even for human to decide, whether a category should be connected to Countries or not. The level of disagreement between accessors sometimes reached 40%, it shows that SCS metrics is very subjective and should be improved in future.

First, we tested how number of pages linked from source to target category indicates a level of semantic relationship. We ranked the categories from *Inset* and *Outset* by the number of pages in them, which are linked to pages in *Countries* set. Using a fixed interval, we selected separately 100 sample categories from both rankings, so they are uniformly distributed across each ranking. These sample categories with corresponding numbers of links are listed in the Table 2. The SCS measure for sampled sets was averaged over every 20 categories, it is shown in the Fig. 5.2. On the ordinate we put the average of SCS metrics, the abscissa shows categories intervals. We see from the plot, that using Inset we have obtained the most strong semantic relationships, while Outset has poor performance.

The better performance of *Inset* is also observed while using Connectivity Ratio as a ranking factor, these results are given in Table 3 and Fig. 5.2. The performance of the

[5] The intermediate values are also possible, for example when averaging the assessment results.

| # | # of links | Inset | # of links | Outset |
|---|---|---|---|---|
| 1 | 3272 | American actors | 193 | Country code top-level domains |
| 2 | 99 | German poets | 21 | Governorates of Egypt |
| 3 | 67 | People from Arizona | 15 | South American history |
| 4 | 52 | 1988 albums | 12 | Antigua and Barbuda |
| 5 | 43 | Rapists | 10 | Cote d'Ivoire |
| 6 | 37 | People from Hawaii | 9 | Yugoslavia |
| 7 | 32 | geography of Egypt | 8 | Ancient Japan |
| 8 | 29 | 1974 films | 7 | Cross-Strait interactions |
| 9 | 26 | Stanford alumni | 7 | Empire of Japan |
| 10 | 24 | Camden | 6 | History of Mongolia |
| 11 | 22 | Pre-punk groups | 6 | Theology |
| 12 | 20 | Nuremberg Trials | 5 | Islands of Singapore |
| 13 | 19 | Video storage | 5 | Energy conversion |
| 14 | 17 | Neighbourhoods of Buenos Aires | 5 | Westminster |
| 15 | 16 | Dutch mathematicians | 4 | Geography of New Zealand |
| 16 | 15 | German currencies | 4 | Lists of lakes |
| 17 | 14 | National parks of Kenya | 4 | Yugoslav politicians |
| 18 | 13 | Cities in the United Arab Emirates | 4 | Encyclopedias |
| 19 | 13 | Egg | 3 | Subdivisions of Afghanistan |
| 20 | 12 | Swedish military commanders | 3 | Geography of Lebanon |
| 21 | 11 | Eurovision Young Dancers Competitions | 3 | Ecuadorian culture |
| 22 | 11 | Basketball at the Olympics | 3 | Rivers |
| 23 | 10 | Communes of Charente-Maritime | 3 | Nova Scotia |
| 24 | 10 | 1846 | 3 | Political parties in Sweden |
| 25 | 9 | New Zealand Reform Party | 3 | Roman Catholic Church |
| ... | ... | ... | ... | ... |
| 76 | 1 | Australian sport shooters | 1 | Hindi |
| 77 | 1 | Canadian pathologists | 1 | Canadian television |
| 78 | 1 | Danish archbishops in Lund | 1 | Abstraction |
| 79 | 1 | Football in Uganda | 1 | Trinidad and Tobago writers |
| 80 | 1 | Ice hockey in China | 1 | Singaporean people |
| 81 | 1 | Latin_American_cuisine | 1 | Scythians |
| 82 | 1 | Mountains of Libya | 1 | Tetraonidae |
| 83 | 1 | Paradox games | 1 | Historic United States federal legislation |
| 84 | 1 | Road transport in Switzerland | 1 | Water ice |
| 85 | 1 | Spanish military trainer aircraft 1970-1979 | 1 | Hiberno-Saxon manuscripts |
| 86 | 1 | Transportation in Manitoba | 1 | Belgian cyclists |
| 87 | 1 | Yom_Kippur_War | 1 | Business magazines |
| 88 | 1 | 62 BC | 1 | British fantasy writers |
| 89 | 1 | Defunct Northern Ireland football clubs | 1 | Victims of Soviet repressions |
| 90 | 1 | Missouri Pacific Railroad | 1 | Empresses |
| 91 | 1 | U.S. generals | 1 | Medieval music |
| 92 | 1 | Creator deities | 1 | Soviet dissidents |
| 93 | 1 | Television stations in the Caribbean | 1 | University of Edinburgh alumni |
| 94 | 1 | Manitoba government departments and agencies | 1 | Signers of the U.S. Declaration of Independence |
| 95 | 1 | Libraries in Illinois | 1 | Microbiology |
| 96 | 1 | Star Wars Trade Federation characters | 1 | Communities in New Brunswick |
| 97 | 1 | Sin City | 1 | Electrical engineers |
| 98 | 1 | Ritchie County, West Virginia | 1 | Thomas the Tank Engine and Friends |
| 99 | 1 | Arapahoe County, Colorado | 1 | California Angels players |
| 100 | 1 | Mega Digimon | 1 | Towns in New Hampshire |

**Table 2.** The 100 sample semantic connections extracted using number of links

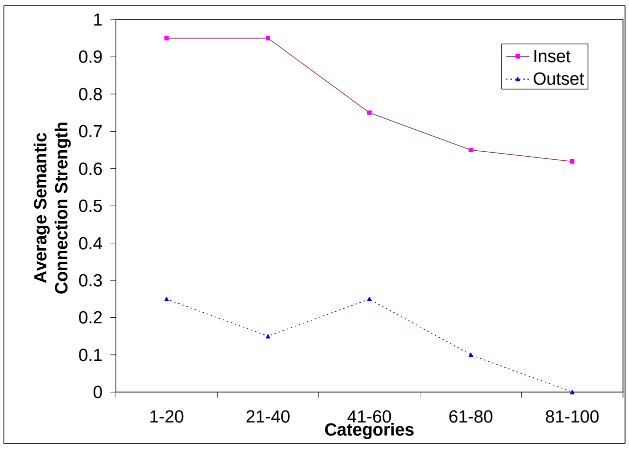


**Fig. 1.** Average semantic connections strength for 100 sample categories, extracted using number of links

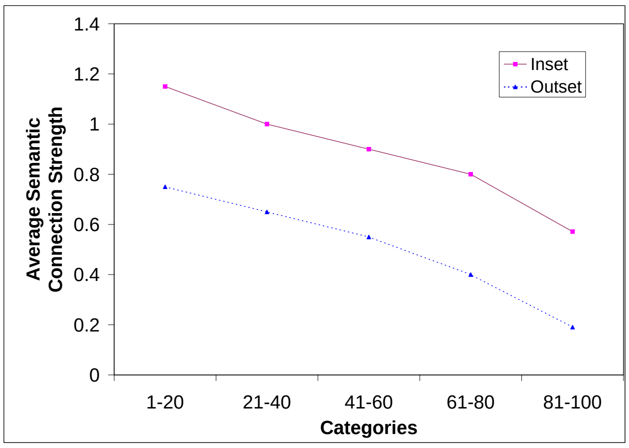


**Fig. 2.** Average semantic connections strength for 100 sample categories, extracted using connectivity ratio

| # | connectivity ratio | Inset | connectivity ratio | Outset |
|---|---|---|---|---|
| 1 | 1 | Johannesburg suburbs | 1 | Provinces of Vietnam |
| 2 | 1 | Cities in Burkina Faso | 1 | New Zealand-Pacific relations |
| 3 | 1 | The Outlaws albums | 1 | Transportation in Lebanon |
| 4 | 1 | Gackt albums | 1 | 9th century BC |
| 5 | 1 | North Carolina Sports Hall of Fame | 1 | Education in Belgium |
| 6 | 1 | Airlines of Liberia | 1 | Lake Kivu |
| 7 | 1 | Tongan rugby league players | 1 | Nepalese law |
| 8 | 1 | Hong Kong radio personalities | 1 | Sport in Lithuania |
| 9 | 1 | Yorb | 0.928571 | Republics |
| 10 | 1 | Education in Qatar | 0.75 | Economy of Greece |
| 11 | 1 | North African music | 0.666667 | Commonwealth Universities |
| 12 | 1 | Zara class cruisers | 0.666667 | World War II European theater |
| 13 | 1 | Airports in Shanghai | 0.571429 | Cities in Kosovo |
| 14 | 1 | Croatian athletes | 0.5 | Languages of Ukraine |
| 15 | 1 | Iranian photographers | 0.5 | Iraqi culture |
| 16 | 1 | Paleozoologists | 0.5 | 1287 |
| 17 | 1 | Swedish sportspeople | 0.5 | Bolivian music |
| 18 | 1 | Ceylon cricketers | 0.5 | Foreign relations of Hungary |
| 19 | 1 | Peanuts | 0.5 | Moroccan society |
| 20 | 1 | 1997_films | 0.5 | Sri Lankan literature |
| 21 | 1 | Archaeological sites in Kazakhstan | 0.461538 | Politics of Macau |
| 22 | 1 | British make-up artists | 0.416667 | Ajaria |
| 23 | 1 | Coscoroba | 0.4 | Peninsulas of Russia |
| 24 | 1 | Farragut class destroyers | 0.361111 | Forced migration |
| 25 | 1 | High_schools_in_Florida | 0.333333 | Bessarabia |
| ... | ... | ... | ... | ... |
| 76 | 0.346154 | BBC | 0.0508475 | Unitarian Universalists |
| 77 | 0.333333 | Hydrography | 0.0487805 | File sharing networks |
| 78 | 0.333333 | Porn stars | 0.047619 | Spanish Civil War |
| 79 | 0.333333 | Komsomol | 0.0447761 | Swedish nobility |
| 80 | 0.32 | 1973 American League All-Stars | 0.0425532 | Battles of France |
| 81 | 0.3 | Roman Republic | 0.04 | Babylonia |
| 82 | 0.285714 | Dacian kings | 0.0384615 | Cantons of Switzerland |
| 83 | 0.272727 | University of San Francisco | 0.037037 | Scales |
| 84 | 0.25 | Esperantido | 0.0344828 | Agriculture organizations |
| 85 | 0.25 | Cooking school | 0.0325203 | Alcoholic beverages |
| 86 | 0.242424 | Church architecture | 0.03125 | New Testament books |
| 87 | 0.222222 | Danny Phantom | 0.0294118 | Marine propulsion |
| 88 | 0.2 | Buildings and structures in Cardiff | 0.0273973 | British politicians |
| 89 | 0.2 | Prediction | 0.025641 | Food colorings |
| 90 | 0.181818 | Media players | 0.0238095 | Christian philosophy |
| 91 | 0.166667 | Computer animation | 0.0222222 | Governors of Texas |
| 92 | 0.153846 | Kroger | 0.0208333 | West Indian bowlers |
| 93 | 0.142857 | Scottish (field) hockey players | 0.0186916 | Ancient Greek generals |
| 94 | 0.125 | Free FM stations | 0.017094 | Spanish-American War people |
| 95 | 0.111111 | Palm OS software | 0.0155039 | English ODI cricketers |
| 96 | 0.09375 | Stagecraft | 0.0136986 | Presidents of the Cambridge Union Society |
| 97 | 0.0769231 | Transportation in Texas | 0.0117647 | Municipalities of Liege |
| 98 | 0.0625 | Guessing games | 0.00952381 | Medical tests |
| 99 | 0.0434783 | Massachusetts sports | 0.00714286 | Food companies of the United States |
| 100 | 0.0163934 | data structures | 0.00414938 | Telecommunications |

**Table 3.** The 100 sample semantic connections extracted using connectivity ratio

Connectivity Ratio measure is up to 25% better comparing it to the plain number of links, this proves the advantage of the normalizing factor.

We presume our results can be explained similarly to the linkage analysis algorithms, where inlinks are considered as sources of authority propagation. The authority of a category increases when it gets links from any other category. This corresponds to a vote from another author that there is a semantic connection between two categories. We also need to check if the difference in size of the two selected sets can explain the discrepancy in the performance.

## 6 Conclusions and Future Work

We have observed that, for a given category, inlinks provide good evidence for semantic relationships, but outlinks have poor performance. We also show that normalized Connectivity Ratio is a better measure for extracting the semantic relationships between categories. We consider this result might be skewed toward our core *Countries* category. It is natural that there are a lot of inlinks for pages representing countries, basically everything is or has happened in a country. The results we obtained are also influenced by the ranking scheme we chose. We have to improve the connectivity ratio formula so that it brings out the relevant relations and removes the trivial ones.

For our future experiments we want to select more categories as a starting set and remove bias introduced by the *Countries* categories. The assessment of semantic relationship should be improved by taking into account possible information need. It would be interesting to study a cardinality of link types relationships. For example, "Actor to BirthDate" is a n:1 relation, while "Actor to Film" is a n:n relation. Another interesting aspect is to investigate bidirectional relationships, categories size and their indegree, we are also going to apply link analysis algorithms for establishing the semantic authorities among categories.

## 7 Acknowledgments

We would like to thank Paul Chirita for numerous discussions and Michal Kopycki and Przemyslaw Rys for their invaluable help with the experimental setup.

## References

1. Wikipedia, the Free Encyclopedia. `http://wikipedia.org`, accessed in 2006.
2. David Aumueller. SHAWN: Structure Helps a Wiki Navigate. In *Proceedings of BTW Workshop WebDB Meets IR*, March 2005.
3. Francesco Bellomi and Roberto Bonato. Network Analisis for Wikipedia. In *Proceedings of Wikimania 2005, The First International Wikimedia Conference*. Wikimedia Foundation, 2005.
4. Ludovic Denoyer and Patrick Gallinari. The Wikipedia XML Corpus. Technical report, 2006.
5. Daniel Kinzler. WikiSense — Mining the Wiki. In *Proceedings of Wikimania 2005, The First International Wikimedia Conference*. Wikimedia Foundation, 2005.

6. Natalia Kozlova. Automatic Ontology Extraction for Document Classification. Master's thesis, Saarland University, Germany, February 2005.
7. Markus Krötzsch, Denny Vrandecic, and Max Völkel. Wikipedia and the Semantic Web - The Missing Links. In *Proceedings of Wikimania 2005, The First International Wikimedia Conference*. Wikimedia Foundation, 2005.
8. Bo Leuf and Ward Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
9. Fernanda B. Viegas, Martin Wattenberg, and Kushal Dave. Studying Cooperation and Conflict between Authors with History Flow Visualizations. In *Proceedings of SIGCHI 2004, Vienna, Austria*, pages 575–582. ACM Press, 2004.
10. Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic Wikipedia. In *Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland*, 2006.
11. Jakob Voss. Measuring Wikipedia. In *10th International Conference of the International Society for Scientometrics and Informetrics*, Stockholm, 2005.
12. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

# Wiki and Semantics: Panacea, Contradiction in Terms, Pressure for Innovation?

## Some experiments and tracks towards *Intelligence Amplifiers*

Jean Rohmer

Centre des Nouvelles Technologies d'Analyse de l'Information
Thales Communications, France
jean.rohmer@fr.thalesgroup.com

**Abstract.** This paper examines the relative characteristics of wiki principles and of semantic systems. It first stresses some oppositions between these two approaches, and exposes the challenge of their reconciliation. We then make a detailed description of Ideliance, a „pure“ semantic tool, and we set criteria to compare several existing semantic wiki systems. After a critical look at some of their features, we propose precise directions for cross-fertilisation of semantics and wikis, advocating for solid, long-term foundations.

## 1 Semantic Wiki: an oxymoron which raises many questions

We must first realise that the existence of the „Semantic Wiki“ concept, comes — like many other Information Systems concepts — from our inability to build machines or programs which automatically understand natural language, either in the form of documents or in the form of spoken or written conversations (as quoted in [1], best analysis programs fail to understand a sentence in more than 70% of the cases). Machines cannot help us without „semantics inside“, and today we have to strenuously feed them with this semantics.

In practice, we permanently have to waver between textual document management and structured database applications. *Is there a life between Word and SQL ?* That is the question Semantic Wiki is about.

There are two ways to approach this question. The first one is extremely theoretic, this is *semantics*, the second one is extremely practical, this is *Wiki*. At first glance, they seem to be much too distant to hope that any fusion is possible. Semantics has been endlessly studied for millenniums in literature, philosophy, philology, linguistics, and Wikis sometimes look like an odd tinkering from an idle programmer's week-end. Moreover, while Wiki is the Hawaiian word for „quick“, semantics refers to things like syntax, study, grammar, school ... and school comes from „skole“ which is the Greek word for „being slow“, „not to hurry“ !

In other words:

**Semantic Wiki means Slow Quick ( SlowQuick ?)**

In this paper, we would like to address various facets of this contradiction, at the light of our experience in the development of a Semantic Network Manager: IDELIANCE [2] and of Information Technology usage in large and small Business environments.

In fact, Semantic Wiki is an ambitious endeavour: it aims at increasing the synergy between people intelligence and the power of the grid of networked computers; and in this period, where it is still embryonic, we should take time -*skole*- to be sure to start with the right groundwork.

In this paper, we first describe the main features of IDELIANCE at the Wiki light, then we review the main characteristics of significant Wiki implementations. This will help us to reformulate the Semantic Wiki challenge, and to make proposals for sound directions for the future.

## 2 Ideliance: a „pure“ semantic network manager with some wiki properties

As reported in [2], Ideliance was initially an attempt to bring the best ideas of Artificial Intelligence on everybody's desktop. In the same way as Ward Cunningham [3] wanted to develop the „simplest online database“, we wanted to develop the „simplest personal knowledge base“. For that purpose, we chose to develop a user-friendly Semantic Networks [4] editor for everyday use in professional life. The first Ideliance prototype was available in November 1993. Commercial usage started in 1996 for France Telecom. Off-the-shelf personal versions were bought in 1998 by French Atomic Energy Commission (CEA), and French Army. The server version was installed in 2000 in large corporations like L'Oreal and Merck Pharmaceuticals. [2] reports on lessons learnt from Ideliance applications.

Although the question of finding the right balance between textual and structured representation was perpetual during Ideliance design, the initial choice was clearly in favour of *full structured semantic networks*. Ideliance proposes users to create **subjects**, belonging to zero, one or more **categories**, and to write **statements** of the form (subject / **relation** / subject), where each relation has an **inverse**. A complement in a statement can be not only another subject, but any resource (file, email, URL). A subject is any character string (including numbers and spaces), without comma, and without length limitation. A set of Ideliance objets is called a **collection**. Example of a statement and its inverse:

John Paul Wagner **works for** United Nations (UN)

United Nations (UN) **is the employer of** John Paul Wagner

In fact, and this is a main difference with wikis, *there is no textual format for statements*: all statements are built from a graphical interface which let users manipulate only names of subjects, relations and categories in a controlled way. Nevertheless, we authorise a free text area associated with each subject, with zones which may point to other subjects, without label/ relation on this link.

The basic mode of information display is the Ideliance „Card“: given a subject, it collects and displays all the statements having it in subject position. This allows for an immediate navigation mode from Card to Card, each being built dynamically. In the server mode, a card displays all up-to-date statements about a subject written by any user. More precisely, each statement has a signature (author, date of creation, visibility rights). User see only statements they are allowed to, either directly or through group membership.

Conversely, several tools allow to publish Ideliance contents (e.g. cards) in useful standard formats like Word, HTML, XML, Excel. Utilities permit also to convert XML and Excel files into Ideliance statements. All reasonable features we can expect on objects are available: delete, rename, duplicate, merge, extract. For instance we can extract and merge collections.

A difference with most of Semantic Wikis is that there is no reference to a standard like RDF in its implementation. Not only because it simply did not exist at this time, but because we wanted *to make symbolic statements the unique, atomic concept for information representation*, totally in the hands of end-users, so as to close all the backdoors to Software Engineers for any underground traffic on information. This does not impede the usage of RDF as a *standard interoperability format* between Ideliance and other semantic applications. (There is no mandate to choose RDF as an *internal* implementation feature). A WSDL Web Service has been developed to let other applications read, write, query an Ideliance server. These services could be reused to provide a more Semantic Web compliant interface to Ideliance.

The main **radical** idea of Ideliance is to **try to** get rid of the notion of document. Each atom of information is a statement, and the tool collects and displays them on demand.

The card feature is one of these tools, but many others are available:

– semantic queries
– textual queries (à la Google)
– simple or complex tables in OLAP-like style
– simple or complex graphs

None of theses features has a textual format accessible by the users, who can only go through an interface. (Internally, the objects corresponding to queries, tables, graphs are represented as system-visible Ideliance statements).

Textual queries simulate the notion of document through the dynamic cards. They retrieve all subjects whose card content matches the query. An option is to extend this textual search to the content of documents used as complements to the card subject. For this purpose, Ideliance embeds the Wilbur search engine [1].

Ideliance interaction with the user fully relies on the notions of **emergence and suggestions**, which can be generally stated as: when an user starts an

[1] see `http://wilbur.redtree.com`

action, the system suggests the usual ways to continue / complete it. We will develop this point later in the discussion.

All these specifications and implementations where not done overnight. Ideliance is the result of may years of evolutions with many kinds of users, either individual or collective ones.

Currently, we are experimenting the addition of information analysis tools: data mining, clustering, knowledge discovery, rules, in particular for Military Intelligence and Business Intelligence applications.

## 3 A survey of some current Semantic Wiki proposals

Existing semantic wikis can be compared using some alternative options:

**Approaches to the challenge of accomodating both free text and structured semantics:**

- put structure / formalism inside text (option A1)
- put text inside structure / formalism (option A2)
- exclude text (option A3)

Note that the second approach is the one adopted by HTML, as the essence of the Web, and later by XML.

**Global design approach:**

- grounded in technical architecture choices (option B1)
- driven by an end-user perspective (option B2)

As we have seen before, Ideliance illustrates Options A3 and B2.

Platypus [5], is a good illustration of option B1 : the idea is that a Wiki page is annotated -in hidden fields- by RDF metadata statements, and that an HTTP server can selectively download these metadata to the client, allowing a natural chain of navigation. In Ideliance, we implemented a similar feature „Ideliance Inside“ for HTML pages, where the hidden (and proprietary at that time) Ideliance format could be extracted from an HTML page, along with a symmetric mechanism allowing to generate Ideliance cards in human-readable HTML, accompanied with the embedded corresponding Ideliance collection.

WikiSAR [6] is an illustration of option A1: the subject is implicitly the page name, and, on a text line, verb and complement are indicated, separated by a colon, each of them following the WikiWord conventions. This very simple structuring scheme is complemented by the capability to insert formal queries *forged with the verbs* in the text. They then are replaced by their result *always up-to-date*. Moreover, WikiSAR can visualise the network of sentences though a friendly graphical interface.

Rhizome [7] is clearly a technical architecture (option B1), with the idea to implement a sort of algebra on subsets of RDF triples. It introduces specific formal languages -ZML, RxML, which make Rhizome a flexible workbench to manipulate semantic nets.

Ikewiki [8] outlines general, ambitious goals for a collaborative semantic environment. Authors analyse in a systemic way the relationship between users acceptance, expressiveness, generality of semantic applications, following a B2 option. More than a Wiki, this project is a general purpose, users-oriented platform for semantic information processing.

Semperwiki [9] is a simple, personal semantic system, with an option A1 structure-in- text approach similar to WikiSAR, and a strong emphasis on bringing incentive to the semantic effort of the users (a B2 option).

A broader discussion of the Semantic Wiki field should also encompass the relationship with the *very close* Semantic Desktop domain [10].

## 4 Some surprising things about Wikis and Semantic Wikis

As stated in a previous paragraph, the Semantic Wiki is today a tiny domain, but, in the same time, it sets some key challenges for the evolution of Information Processing systems. It is de-facto a domain which harnesses all studies conducted since more than 40 years in Software Engineering, Artificial Intelligence, Formal Logic, Natural Language Processing. We must not take a narrow, short term view of it. Instead, we must analyse this domain lucidly.

The tinkering aspect of some Wiki implementations (see for instance TheManySetsOfRulesToBuildWikiWordsAlsoCalledCamelCase) could at the end of the day cause more harm than benefit, even if they contribute to the very nature and initial success of wikis. A long term perspective for Semantic Wikis needs a joint, sustainable effort from the community. At this point the *semantic side* of Semantic Wiki should help: it is striking to realise that explaining the basics of a semantic tool like Ideliance finally takes less time and space than detailing all the folklore around Wikis. (WikiTag, WikiCategories, QuickSurvey, ReverseIndex, RoadMap, WikiSingleWordProblem, WikiNamePluralProblem, WikiKeyWords, etc, etc). Finally the written documentation on the Wiki unformal conventions is thicker than the formal one on semantic tools. *In this respect, we think it is safer to carefully inject the Wiki spirit inside Semantic (Desktop) systems than the opposite.*

A surprising thing about Semantic Wikis is the quasi absence of reference to the tools used daily by all professionals: Excel, PowerPoint (or their open source equivalent). Such tool are in themselves *excellent structuring tools which embed a lot of semantics* as compared to textual documents. We should make efforts to bridge their *actual* semantics with the *potential* semantics of wikis, all the more if we target personal or collaborative applications. A side point would be the support of figures and simple arithmetic, which are ubiquitous in any kind of business. If we do not address these points, the risk is to limit ourselves to the development of encyclopaedia-like usages.

I am always very surprised to notice that the notion of *inverse relation* is nearly absent in semantic tools. It was one of our first decisions in the design of Ideliance. This trivial mechanism has nevertheless the capability to build automatically symmetric cross-referencing and navigation from one card / page to the other. It is also extremely useful to express queries and rules in a symmetric way.

Finally, the constant reference as a kind of creed to RDF in most Semantic Wiki designs seems to me overrated: A Semantic Wiki is *neither more semantic nor more wiki* simply because it gives visibility to the RDF standard. A Semantic Wiki user in the future should ignore the existence of RDF, as well as PowerPoint users ignore that there exists a Metafile format which allows them to cut / paste schemas from / to other office documents formats. *RDF is key to provide interoperability among the semantic applications, but is an irrelevant concept for the end-user.* For instance, the rock-bottom notion of blank node is a semantic nonsense for the end-user. In the same spirit, the notion of unique URI, often quoted as an advantage when placed inside a RDF statement, is a concept which becomes less important for users, more familiar with information retrieval through search engines than through the URI of their object of interest. *The semantics as perceived by users should differ from the one perceived by programs.* An argument for this statement is that many wiki designers forge new friendly formal dialects, softer than RDF, to let users manipulate formal semantics. (See for instance ZML [7])

## 5 Proposals for a sustainable fusion of Semantic and Wikis

In the previous paragraph, we were voluntarily critical at some aspects of wikis, semantic wikis and semantic systems. We would like now to make proposal to cross-fertilise their advantages while minimising their potential weaknesses.

**Transpose to semantics the wiki culture of a community taking care of a knowledge-base** The success of Wikipedia is the proof that some users are diligent towards informal semantics. Let us encourage the same sort of people to become as diligent towards formalised semantics. There exist *potential semantic (re)writers / translators* which could translate natural language pages of a wiki into a semantic format (in the same way publishers are used to have papers translated from one natural language to another). Such added value would generate a networked snowball effect : reading the wiki would become more pleasant and efficient, and casual writers would be tempted to turn themselves into *semantic writers* to raise the level of retrievability and understandability of their contribution.

**Design carefully a specific level of semantic formalism for the end-user** As exposed above, this level should definitely depart from the Semantic

Web standards (RDF, OWL) which, by essence, were semantic constructs for machines, and not for people. Of course, this user-level semantics could made operational and interoperable through the use of the Semantic Web standards, but in a hidden way, in lower layers of the Internet machinery. This level should also differ from the first generation of wiki conventions, while keeping their freshness.

Our experiment with the simple Ideliance formalism is a proof of existence of this level. Designing and agreeing upon such a user-level semantic will not be immediate, but it is a long term key success factor.

**Make Semantic Wikis a companion of office tools instead of a substitute** This is mandatory for the acceptance of Semantic Wikis in most of economic sectors. Not only Semantic Wikis should import / export their contents from / to office documents formats, but they also should be able to capture in real time the semantics of graphical editing a presentation or a spreadsheet. In the same way current wikis have a Text Processor-like face, there must exist Semantic Wikis with a Spreadsheet-like face, a Presentation Editor face.

With the emerging XML standards to describe the contents of office documents, this objective is not out of reach.

**Encourage a „Semantic Inside“ policy for HTML pages** The idea of populating HTML pages with semantic statements, along with the capacity of Semantic Wikis or Semantic Desktops to selectively download them seems very simple, and capable of initiating a viral propagation of semantic statements. It has been implemented as an „Ideliance Inside“ feature, and also in [5]. We should analyse why it does not exist in reality, and what conditions should be met to start such a proliferation.

**Use human-driven discovery and emergence mechanisms for vocabulary / ontology congruence** The whole semantic game is complex : it is a continuum of interactions between a continuum of levels, e.g. from personal, to workgroups, to corporate, to global level. The notion of ontology, ubiquitous in the Semantic Web along the idea of URI uniqueness, needs to be reformulated to meet this complexity.

„But where danger is, grows the saving power also.“ cited from „Patmos“, by Friedrich Holderlin.

Here the „danger“ is „people complexity“, as compared to the simple „machines“ world of the Semantic Web. And thus the saving power is „people“ too.

The alignment / congruence tools in the semantic wiki world should *empower users with the ultimate decisions concerning the meaning of terms*, instead of being blind black boxes trying to make some „optimal“, global ranking of the „best“ meaning. These tools must compute / discover emerging properties from the whole knowledge base, let users make their choice, observe theses choices, and capitalise from them for further recommendations.

In Ideliance, a first modest implementation of this emergence principle consists of maintaining statistics on relations and complements of subjects of a given category. This simple mechanism has already the capability to give, in real time, an up-to-date view of the „data model“ of the collection, without any a-priori declaration.

In the same way the Sun Microsystems slogan is *„The Network is the Computer“*, we tend to say: *„Information is The System“*. The system structure emerges from the information it contains.

**Take into account higher levels of semantics: Discourses, Argumentation, Rhetoric** Any *semantic* writing act has an intention, an objective, expects some results or benefits. Personal or collective semantic tools should help users in such high-level tasks. This is for instance by the ABCDE format for scientific publishing [1]. This paper is a first attempt to cleave the monolithic document, by explicitly labelling paragraphs as **B**ackground, **C**ontribution and **D**iscussion, along with general **A**nnotation, and collection of useful resources in an **E**ntities paragraph.

The other nice side of this *editing* effort will be the *reading* side, which, after some *thinking*, and *working*, will again yield a new *editing* activity.

We could call this proposal *„soft semantics“*, since it keeps the very meaning inside natural language sentences or paragraphs. It makes a step further than *metadata semantics* à la Dublin Core, which takes the document as a monolith. With Ideliance, we clearly are trying a *„hard semantics“* approach, which we call also *Extreme Explicit Semantics*. (The *Extreme Implicit Semantics* would be automatic natural language understanding)

If we merge *„hard semantics“* with the ABCDE approach, we could contemplate tools which not only would represent the semantics of each sentence, but also the semantics of the *making-up of all the sentences* towards the author's intention. This would permit to represent things like: „statement A is used by author B as an example of statement C which is later used as a support for statement D“. In a companion paper [11], we outline the notion of *„litteratus calculus“* as an infrastructure to underpin such an approach.

## 6 Conclusion: Towards Intelligence Amplifiers

[6] advocates for immediate gratification for semantic writing. This idea of a fast ROSI (Return On Semantic Investment) is well in pace with the Wiki world. And this point is key for the acceptance of semantic wikis.

Coming back to our built-in contradiction (Semantic Wiki = Slow Quick), we would like to point out another kind of gratification distillated by semantic writing: it is a long term, deferred, -*skole*- reward : the pleasure of thinking, of installing order, clarity in one's knowledge and thoughts. The effort to decide to create a new subject, to choose a category for it, to forge a new statement linking two subjects is a mental exercise, which, day after day, amplifies *your* intelligence. Clicking is not thinking.

**Acknowledgments:**

I would like to thank the main contributors to Ideliance : Sylvie Le Bars[2] for designing the subtle tempo between slow and quick, click and think, and Stéphane Jean and Denis Poisson for the technical design and implementation.

## References

1. Waard, A.D., Oren, E., Haller, H., Völkel, M., Mikka, P., Schwabe, D.: The abcdef format. In: Submitted to ESWC 2006. (2006)
2. Rohmer, J.: Lessons for the future of semantic desktops learnt from 10 years of experience with the ideliance semantic networks manager. In: ISWC 2005 Galway, Ireland. (2005)
3. Cunningham, W.: (Wiki design principles) available at `http://c2.com/cgi/wiki?/DesignPrinciples`.
4. Sowa, J.F.: (Semantic networks) available at `http://www.jfsowa.com/pubs/semnet.htm`.
5. Campanini, S., Castagne, P., Tazzoli, R.: Platypus wiki. In: 3rd International Semantic Web Conference (ISWC) Hiroshima, Japan. (2004)
6. Aumueller, D., Auer, S.: Towards a semantic wiki experience – desktop integration and interactivity in wikisar. In: ISWC2005 Semantic Desktop Workshop, Galway, Ireland. (2005)
7. Adam, S.: Building a semantic wiki. IEEE Intelligent Systems **20**(5) (2005)
8. Westenthaler, R., Schaffert, S., Gruber, A.: A semantic wiki for collaborative knowledge formation. (In: Semantics 2005, Vienna, Austria, 24th November 2005)
9. Oren, E.: Semperwiki: a semantic personal wiki. In: Proceedings of the 1st Workshop on The Semantic Desktop, Galway, Irland. (2005)
10. First Semantic Desktop Workshop 2005, Galway, Ireland, S. Decker and J. Park and D. Quan and L. Sauermann (2005)
11. Rohmer, J.: Litteratus calculus: a manifesto for a demographic way to build a sustainable semantic web. In: ESWC. (2006)

[2] see `www.arkandis.com`

# A Wiki as an Extensible RDF Presentation Engine

Axel Rauschmayer and Walter Christian Kammergruber

Axel.Rauschmayer@ifi.lmu.de
Walter.Kammergruber@gmail.com
Institut für Informatik
Ludwig-Maximilians-Universität München

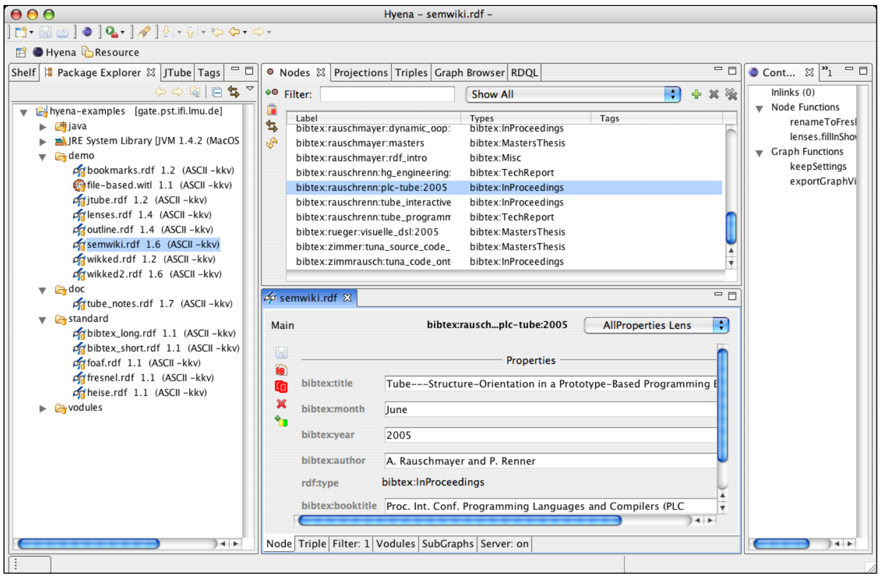
**Abstract.** *Semantic wikis* [1] establish the role of wikis as integrators of structured and semi-structured data. In this paper, we present WIKKED, which is a semantic wiki turned inside out: it is a wiki engine that is embedded in the generic RDF editor HYENA. That is, HYENA edits (structured) RDF and leaves it to WIKKED to display (semi-structured) wiki pages stored in RDF nodes. Wiki text has a clearly defined core syntax, while traditional wiki syntax is regarded as syntactic sugar. It is thus easy to convert WIKKED pages to various output formats such as HTML and LaTeX. WIKKED's built-in functions for presenting RDF data and for invoking HYENA functionality endow it with the ability to define simple custom user interfaces to RDF data.

**Keywords:** Semantic wiki, RDF, wiki engine, wiki syntax, RDF presentation.

## 1 Introduction

What would later be called WIKKED started as a project about one and a half years ago, when we discussed combining two of our favorite technologies: RDF and wikis. We never thought that this combination would carry us as far as it did (which we think bodes well for the newly-named community of "semantic wikis"): Initially, we had an RDF editor called HYENA and just wanted to mark up and link pages with it. So we implemented the small wiki engine WIKKED and embedded it in HYENA. In the end, WIKKED grew far beyond its basic functionality and is now even used for constructing small custom user interfaces that display and manipulate RDF. The current focus of WIKKED is to complement the GUI-based HYENA and not to replace it with a full-blown web-based RDF editor. One can, however, run HYENA in a web server mode and have WIKKED act as a "normal" wiki (Sect. 5).

This paper is directed at members of the semantic wiki community, and thus we neither explain RDF [2, 3] nor wikis [4]. It has the following structure: The next section describes how WIKKED fits into the whole of the HYENA picture and what Fresnel lenses are. Sect. 3 explains the core components of WIKKED, Sect. 4 walks the reader through an example. Sect. 5 illustrates WIKKED's advanced features such as an interactive command line and a web mode. We end the paper by giving related work, future research and a brief conclusion.


Hyena – semwiki.rdf –
Hyena
Resource
Shelf
Package Explorer
JTube
Tags
hyena-examples [gate.pst.ifi.lmu.de]
java
JRE System Library [JVM 1.4.2 (MacOS
demo
bookmarks.rdf 1.2 (ASCII -kkv)
file-based.witl 1.1 (ASCII -kkv)
jtube.rdf 1.2 (ASCII -kkv)
lenses.rdf 1.4 (ASCII -kkv)
outline.rdf 1.4 (ASCII -kkv)
semwiki.rdf 1.6 (ASCII -kkv)
wikked.rdf 1.2 (ASCII -kkv)
wikked2.rdf 1.6 (ASCII -kkv)
doc
tube_notes.rdf 1.7 (ASCII -kkv)
standard
bibtex_long.rdf 1.1 (ASCII -kkv)
bibtex_short.rdf 1.1 (ASCII -kkv)
foaf.rdf 1.1 (ASCII -kkv)
fresnel.rdf 1.1 (ASCII -kkv)
heise.rdf 1.1 (ASCII -kkv)
vodules
Nodes
Projections
Triples
Graph Browser
RDQL
Filter:
Show All
semwiki.rdf
Main
bibtex:rausch...plc-tube:2005
AllProperties Lens
Properties
Node
Triple
Filter: 1
Vodules
SubGraphs
Server: on
Cont...
Inlinks (0)
Node Functions
renameToFresl
lenses.fillInShov
Graph Functions
keepSettings
exportGraphVi

| Label | Types | Tags |
|---|---|---|
| bibtex:rauschmayer:dynamic_oop: | bibtex:InProceedings | |
| bibtex:rauschmayer:masters | bibtex:MastersThesis | |
| bibtex:rauschmayer:rdf_intro | bibtex:Misc | |
| bibtex:rauschrenn:hg_engineering: | bibtex:TechReport | |
| bibtex:rauschrenn:plc-tube:2005 | bibtex:InProceedings | |
| bibtex:rauschrenn:tube_interactive | bibtex:InProceedings | |
| bibtex:rauschrenn:tube_programm | bibtex:TechReport | |
| bibtex:rueger:visuelle_dsl:2005 | bibtex:MastersThesis | |
| bibtex:zimmer:tuna_source_code_ | bibtex:MastersThesis | |
| bibtex:zimmrausch:tuna_code_ont | bibtex:InProceedings | |

| | |
|---|---|
| bibtex:title | Tube---Structure-Orientation in a Prototype-Based Programming E |
| bibtex:month | June |
| bibtex:year | 2005 |
| bibtex:author | A. Rauschmayer and P. Renner |
| rdf:type | bibtex:InProceedings |
| bibtex:booktitle | Proc. Int. Conf. Programming Languages and Compilers (PLC |

**Fig. 1.** A typical Hyena window: To the left is a standard view on the file system. Center bottom is the editor for the currently open file, center top is a view that lists all RDF nodes in that file. The editor shows the content of the currently selected resource. To the right, there is a view displaying the context of the selected resource: incoming links and functions that can be applied to it.

## 2 Background

### 2.1 Hyena

Hyena is a generic RDF editor that has been implemented as a plugin for the Eclipse Rich Client Platform [5] Fig. 1 shows what a typical Hyena environment looks like. Hyena's internal architecture is depicted in Fig. 2. Support for an RDF vocabulary in Hyena is provided by plugins called *vodules* (*vo*cabulary mo*dules*). Vodules take care of three different facets of RDF editing: First, a vocabulary has associated declarative data such as OWL constraints, rdfs:comments and rdfs:labels. Accordingly, each vodule can contribute a *subgraph*[1]; every container (or *graph*) of RDF data that is managed by Hyena is composed of several subgraphs. Second, there will always be RDF data formats that cannot be meaningfully edited with a generic editing mechanism. Thus, vodules can implement their own custom widgets for node editing. Third, imperative knowledge might be needed for manipulating RDF data. Vodules add

[1] The reader might know subgraphs under the moniker *named graphs* [6]. We have emulated this feature before implementations in RDF engines (such as Sesame [7] became available, which is why we are still using this legacy name.

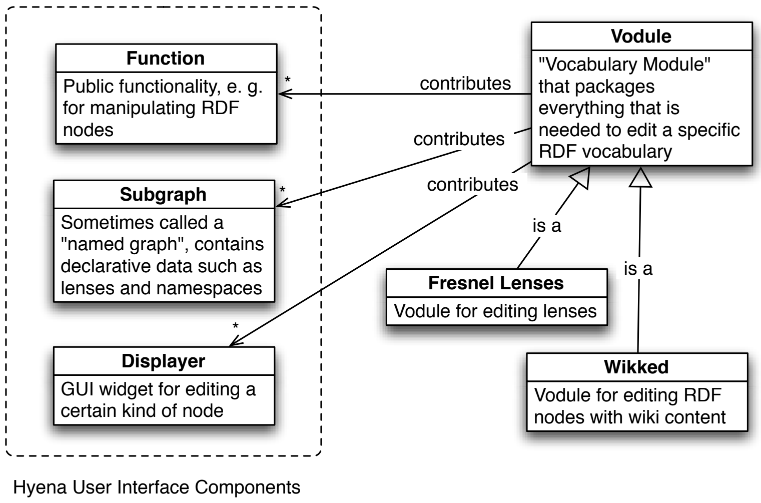

Function
Public functionality, e. g. for manipulating RDF nodes
Subgraph
Sometimes called a "named graph", contains declarative data such as lenses and namespaces
Displayer
GUI widget for editing a certain kind of node
Hyena User Interface Components
Vodule
"Vocabulary Module" that packages everything that is needed to edit a specific RDF vocabulary
contributes
contributes
contributes
is a
is a
Fresnel Lenses
Vodule for editing lenses
Wikked
Vodule for editing RDF nodes with wiki content
*
*
*

**Fig. 2.** The main components of HYENA's user interface are functions, subgraphs and displayers. A *vodule* supports editing of RDF vocabularies by making cross-cutting contributions to each group of components. Vodules for Fresnel lenses and WIKKED pages provide editing support for these vocabularies.

these to HYENA as *functions*. Note that functions can both be accessed via the GUI and as WIKKED commands.

### 2.2 Fresnel Lenses

The *Fresnel Display Vocabulary* [8] is the foundation of HYENA's generic editing facilities. This vocabulary is used to define how nodes of a given type should be displayed. One such definition is called a Fresnel *lens*. The most basic version of a lens needs to state what properties should be displayed and in what order. As an example, here is a minimal lens for the "Friend of a Friend" vocabulary (FOAF, [9]) written in Notation 3 [10] syntax (taken and abridged from [8]):

```
:foafPersonDefaultLens rdf:type fresnel:Lens ;
                       fresnel:classLensDomain foaf:Person ;
                       fresnel:showProperties ( foaf:name
                                                foaf:surname ) .
```

We have given this lens node the URI `:foafPersonDefaultLens` and assigned it the type `fresnel:Lens`. The lens applies to nodes of type `foaf:Person` and displays first and last name (in that order; the value of `fresnel:showProperties` is an `rdf:List`). Advanced Fresnel features include more sophisticated formatting (display a property value as plain text, a link, an image, . . . ) and negative definitions (display all properties except X). Fresnel lenses have the advantage that one can deliver data together with instructions on how it is to be displayed. The result of applying a lens to a node is called a *projection* in HYENA.

## 3 Wikked and WITL (Wikked Template Language)

WIKKED is the complete wiki engine, WITL the formal language used for marking up a page. In this section, we will go into details about the design decisions we have taken. They were based on several requirements we had: Initially, We had the RDF editor HYENA and wanted to add notes in rich text. The simplest solution was to start with plain text and support some wiki markup such as headings, lists, tables and text styles. Soon it became obvious that it would be nice to have the ability to display RDF data inside these nodes. For that, we needed a more complete syntax and gave WITL a bracket-based *core syntax* reminiscent of XML. We still support wiki markup by translating it to the core syntax. RDF integration gave us the ability to refer to other nodes. As wiki pages were just another kind of node, we arrived at the most basic wiki functionality: markup and linking. Having a syntax with a proper LL(k) grammar makes it easy to translate our notes to different target markup languages: Apart from the obvious choice of HTML for rendered display of a note, we can also produce LaTeX. This proved handy for situations where one does some quick brainstorming with bullet lists and headings and is able to move on to LaTeX as a more professional publishing format later. The data we presented using WITL was still static. We wanted to add more interactivity and were able to do so by making it easy to extend the WITL vocabulary in Java and by opening up the rest of HYENA to WIKKED. Now WITL commands allow one to apply functions to RDF nodes.

*The Case for Plain-Text Markup.* The most obvious reason we went with plain-text editing is that it is much easier to implement than a true WYSIWYG editor. HYENA's users are very technical, so it was to be assumed that they would be able to handle the challenge. Later, we will still be able to add a WYSIWYG editor. But as WITL is to be frequently extended, typing a plain text is paradoxically often *more* usable than visual editing. LaTeX's ongoing popularity is testament to that. Furthermore, plain text is still the most convenient, most widely used and most cross-platform data format. For example, even if you know nothing about wiki markup, you are still bound to use it if you are sending an email with a bullet list from a Windows computer to a Linux computer and store it in a version control system there. One final consideration is that WITL has to be easier to type than XML, which we otherwise could have used instead. We start this section with a quick example and follow up on it with a more detailed explanation.

### 3.1 WITL by Example

Before we properly introduce the syntax, we give an example of WITL markup:

```
The {i:core syntax} uses {b:braces} to delimit markup directions.
Simple links follow the same scheme: {http://hypergraphs.de/}.
Further arguments are separated by carets:
```

```
{http://hypergraphs.de^Hypergraphs web site}.

- Then there is also wiki-inspired syntactic sugar
  that is translated into the core syntax.
- Bullet lists, **bold**, ~~italics~~ etc. all work.
```

### 3.2 Core WITL

At heart, all generic markup languages look very similar: Be it XML, LaTeX or Lisp's S-expressions [11], there are always nested named terms. If you didn't like S-expressions being part of this last list (because they are not really a markup language), you can already guess at the hybrid nature of WITL: On one hand, commands should be easy to type. On the other hand, text will usually dominate, whereas in, say, Java, it can only appear within quotes. So what will our nested terms look like? As they have to be easy to type, XML's named *closing* brackets are out and terms where only the beginning is named are in, as in simple LaTeX commands and S-expressions. Let us start by putting braces around URLs which have to be easy to embed if WIKKED is to be worthy of the "wiki" moniker. We opted for braces, because they appear less frequently in plain text than parentheses and square brackets[2]. We then generalize this syntax to `{tag:argument}` where `http` or `ftp` of an embedded link is just another tag. To allow more than one argument, we introduce the caret "`^`" as an argument separator. Again not for esthetic reasons, but because it appears rarely in plain text (as opposed to the ampersand `&`, less-than `<` or greater-than `>`) and because we will use some of the other candidates (such as the pipe symbol `|`) for wiki markup later. This leads us to expressions such as

```
Text in {b:{i:bold italics}}
A link to {a:http://external.com^External Corp.}
```

Comments are written as `{* Comment *}`. Next, if we are to integrate Java, we need a way to express method invocations: creating a list (with the String elements `''a''` and `''b''`) and invoking method `add` on it, to append the element `''c''`, is expressed as `{{List:a^b}add:c}`.

The final two constructs are *raw text* and *pairs*. Raw text is for circumventing WITL when it isn't powerful enough: whereas typical plain text will have critical characters (such as less-than in XML) of the target markup language escaped, raw text is taken verbatim. Therefore one can directly add code in the target language. Raw text syntax is `[[[raw text]]]`. Pairs have two roles: We use them as a basic data structure, among other things to encode maps and we use them for specifying options for a function without getting lost in positional parameters. These two roles are quite related: options are passed to a function as a separate argument containing a map from string to arbitrary data (specified by the user as a sequence of pairs). Pairs are written as `{key=value}`. Fig. 3 gives an overview of the core syntax.

[2] Additionally, we want to keep the option open of using square brackets for citations.

| | |
|---|---|
| `Plain text` | is default, the following constructs are always started by special character. |
| `{a:url^text}` | a wikked function, most HTML tags are defined |
| `{http://foo.com}` | links fit naturally into the syntax scheme |
| `{{List:xxx}add:12}` | method invocation |
| `{key=value}` | a pair (e. g. for optional arguments) |
| `[[[raw]]]` | "raw" or unescaped text; use to insert HTML |
| `{* Comments *}` | ignored... |
| `{:varName}` | synonym for `{get:varName}` |

**Fig. 3.** The core syntax of WITL.

## 3.3 Wiki Markup

While the syntax we have introduced so far is very regular and easy to define, for some of the more commonly used markup, it would be nice to have more "visual" text markup, just like traditional wikis have. Note that for many constructs, this syntax is line-based and largely incompatible with our term/bracket-based core syntax. As we don't want to lose our core syntax, parsing happens in two stages in Wikked: First, we parse the core terms. Second, we parse lines inside the terms and translate traditional wiki markup to core WITL. While a sequence of lines always starts when a term opens and ends when it closes, a single line can contain both plain text and nested terms. A plain text newline finishes a line. Through this translation step, wiki markup is syntactic sugar for the core syntax. Fig. 4 shows what markup is available. We have intentionally left out breaks and separator lines as these can be cleanly expressed with normal WITL as `{br}` and `{hr}`.

## 3.4 Java Integration

We initially experimented with turning WITL into a full-blown programming language, but it turned out that that was very cumbersome. We didn't have the development tools we were used to (life without automated refactoring is painful) and WITL syntax is more suited for markup than for programming. As we needed to bridge over to Java anyway[3], we chose to stay very close to Java when it came to defining WITL's semantics and libraries: WITL is a functional language whose values are Java objects. For example, when evaluating the expression `{func:[[[<arg1>]]]^<arg2>}` the first argument is passed to the function `func` as the Java text string `''<arg1>''`, while the second argument is escaped first and then also passed as a string. Escaping depends on the currently used target language; for HTML the second argument becomes `''&lt;arg2&gt;''`. Functions cannot be defined (only applied) in pure WITL. They are defined by registering Java objects with the WITL interpreter. Every method in such an object hat has been marked with a special annotation [12] is

[3] It is, after all, the implementation language of Wikked and Hyena

| | |
|---|---|
| `==== Heading Level 1 ====`<br><br>`=== Heading Level 2 ===`<br><br>`== Heading Level 3 ==`<br><br>`= Heading Level 4 =` | **Heading Level 1**<br><br>**Heading Level 2**<br><br>**Heading Level 3**<br><br>**Heading Level 4** |
| `Paragraphs`<br>`are`<br><br>`separated by`<br>`blank lines` | Paragraphs are<br><br>separated by blank lines |
| `Tabs lead to`<br>`        indented (quoted) text` | Tabs lead to<br><br>    indented (quoted) text |
| `**bold**`<br>`~~italics~~`<br>`''teletype''` | **bold**<br>*italics*<br>`teletype` |
| `- Bullet lists`<br>`- can be`<br>`  unordered`<br>`   + or ordered.`<br>`   + One can also nest them.` | • Bullet lists<br>• can be unordered<br>  1. or ordered.<br>  2. One can also nest them. |
| `: Colons`<br>`  are for definition lists with`<br>`: term`<br>`  and definition` | Colons<br>    are for definition lists with<br>term<br>    and definition |
| `\|\| Table \| Heading \|`<br>`\| cells \| in the \|`<br>`\| table \| body \|` | **Table** **Heading**<br>cells in the<br>table body |
| `% single-line comment` | |

**Fig. 4.** Wiki markup supported by WITL.

afterwards visible as function in WITL. Before applying a function, WITL compares the number and type of the function arguments to the method signature and makes sure that both correspond. This mechanism facilitates unit-testing and building new functions by composing existing ones. In both cases one can stay in Java and does not have to invoke WITL in any way.

### 3.5 RDF Integration

A wiki goes from mere text to hypertext by allowing one to link pages. There are two directions of RDF integration in WIKKED: First, wikked pages live inside RDF, they are normal RDF nodes, with attached literals that store the page content. We are discouraging the use of blank nodes for wiki pages and automatically generate a URI node when the user wants to create a new page[4]. The reason for this is that blank nodes cannot (stably) be exported as public locations. Interestingly, RDF frees us from purely name-based identification of pages. At the cost of slightly increased complexity, we got rid of problems such as renaming, support for multiple languages and disambiguation.

Second, one can access RDF content with WITL. The most basic functionality here is to link to other nodes which can by either other WIKKED pages or arbitrary RDF content. HYENA uses the types of a node to determine whether they should be displayed as wiki pages or as something else. Now, where traditional wikis embed the links to other entities inside the text, this is not a good option for RDF; renaming a node would result in the connection being broken. The solution is to reify that connection in RDF. This happens in two complementary ways:

- Subject linking: Whenever the user has *one specific* RDF node in mind, he just pastes its URI into the WITL source; e. g. as `{subj:http://my-node.com}`. Before saving, WIKKED converts the source in the following manner: The page node is used as a `rdf:Seq` of all nodes referenced by subject links. Therefore, the first subject link is the value of property `rdf:_1`, the second one of property `rdf:_2`, etc. The argument of the `subj` function is then changed to be the Seq index. If the above example is the first subject link, it will be changed to `{subj:1}`. All this happens transparently to the user: Before the user edits the page the next time, we put the URIs back in. We have thus moved the actual link out of the source code into a triple. Inserting the link target into the source on-demand guarantees that it is always current. Fig. 5 shows subject linking in use.
- Object linking: If the user has attached *a set of* RDF nodes, he specifies the property he has used, as in `{obj:http://my-predicate.com}`. When displaying the page, WIKKED lists all values of that property. Fig. 9 is an example of object linking.

### 3.6 Skins

*Skins* are a feature that has been created for the web (or non-embedded) personality of WIKKED. When creating web sites, it is often desirable to have reusable

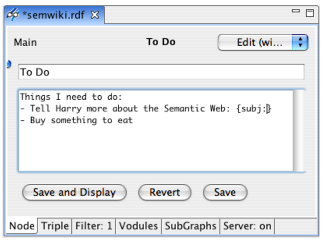
[4] A page node can be easily renamed to a neater URI at any time.

frames for certain subsets of the pages. This is what WIKKED provides with skins: The current display state is denoted as a stack of page IDs (namely, their RDF nodes). The bottom of the stack is the actual content, an arbitrary amount of skins can be pushed on top of it:

| Index | Kind of page |
|---|---|
| ... | etc. |
| 2 | meta-skin (e. g. content that is the same for the complete web site) |
| 1 | skin or meta-page (e. g. a frame for one section of a web site) |
| 0 | base page (actual content, without user interface frames) |

Rendering of a composite page is performed by going through the stack, starting with the topmost page. Each page is evaluated and passes evaluation on to the next stack element with a special function. The result of the evaluation is usually a text string. This kind of nesting can be compared to around methods in CLOS or super calls in Java. While navigating through the site via links, each transition can specify to either remove or replace elements of the page stack. That is, one can display the same page with a different skin, a different page with the same skin, (prudishly) not show any skin, etc. In case a skin wants to display properties of the content page, we allow it to access the page stack. The index of the content page is always 0, the first skin has index 1, the meta-skin has index 2 and so on. Whenever one starts up WIKKED without specifying what page to display, it fills the stack with a default URI for the start page (index 0) and the start skin (index 1).

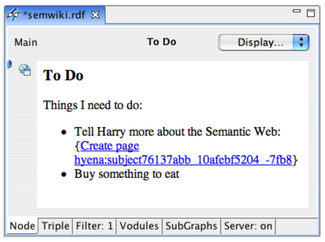
## 4 Example



To Do

```
Things I need to do:
- Tell Harry more about the Semantic Web: {subj:}
- Buy something to eat
```

(a) Editing the page



**To Do**

Things I need to do:

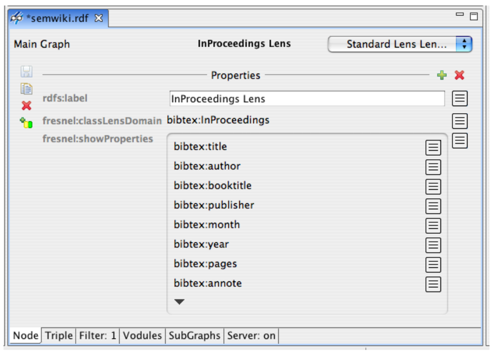
- Tell Harry more about the Semantic Web: {Create page hyena:subject76137abb_10afebf5204_-7fb8}
- Buy something to eat

(b) Displaying the page

**Fig. 5.** (Left) We pick the title "To Do" for the new page and type the actual content in the text field at the bottom. The empty subject link will get us the option to create a new page. (Right) Using the combo box in the top right corner, we switch the view from WITL source to rendered HTML.

– Let us say we want to use WIKKED to create a page with things we have to do. We use the standard HYENA command for creating a new node and assign it the type `wikked:Page`. Afterwards, we can select a (type-sensitive) displayer for entering the wiki text. We decide on a fitting title for our page and type the text containing the to-do items (Fig. 5(a)). Note that we have not used core WITL, but rather the wiki markup for bullet lists. If we leave the subject argument empty, WIKKED will display a link with which we can create a new page node. Next, we pick a different view of the currently displayed node: whereas up to now, we have looked at WITL source, we now switch to rendered HTML (Fig. 5(b)).

– Before we continue with sketching what we want to tell Harry, we decide that it makes sense to refer him to two papers of ours. This allows us to put WIKKED's RDF integration to use: We parse our BibTeX bibliography with the external BibTeX-to-RDF converter "bibtex2rdf" [13]. Then we use the *Fresnel* display vocabulary [8] to tell HYENA how to display the BibTeX entries. A Fresnel lens declares (per RDF schema class) what properties to display and in what order. HYENA has a built-in lens editor, so creating the lens only involves the following four simple steps: First, create a new lens. Second, add a new property specifying what class this lens applies to. Third, use a predefined HYENA command to collect all used properties from the RDF instances in our RDF graph. Fourth, remove those properties that are to be ignored (none in the example) and rearrange remaining ones via drag and drop until we are satisfied with their order (Fig. 6).

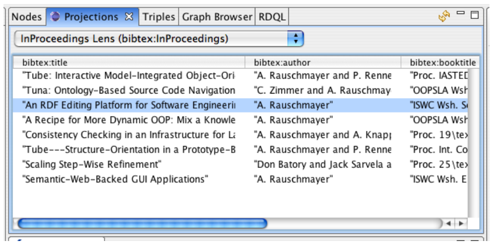


InProceedings Lens

rdfs:label: InProceedings Lens
fresnel:classLensDomain: bibtex:InProceedings
fresnel:showProperties:
bibtex:title
bibtex:author
bibtex:booktitle
bibtex:publisher
bibtex:month
bibtex:year
bibtex:pages
bibtex:annote

**Fig. 6.** HYENA has a built-in editor for Fresnel lenses. This figure shows the finished lens where the properties have been automatically filled in by looking at all instances of `bibtex:InProceedings`. The boxes to the right of the predicate names can be dragged and dropped to rearrange the order in which properties will be displayed when using the lens.

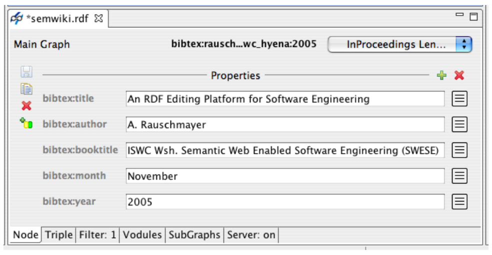
– After we have defined the Lens, we want to try it out. First, we use a HYENA view to show us all instances in the current RDF graph that our lens applies to (its *projections*, Fig. 7). Then we click on a list item and have its contents displayed as defined by the lens (Fig. 8).



InProceedings Lens (bibtex:InProceedings)

| bibtex:title | bibtex:author | bibtex:booktitle |
|---|---|---|
| "Tube: Interactive Model-Integrated Object-Ori | "A. Rauschmayer and P. Renne | "Proc. IASTED |
| "Tuna: Ontology-Based Source Code Navigation | "C. Zimmer and A. Rauschmay | "OOPSLA Wsh |
| "An RDF Editing Platform for Software Engineeri | "A. Rauschmayer" | "ISWC Wsh. S |
| "A Recipe for More Dynamic OOP: Mix a Knowle | "A. Rauschmayer" | "OOPSLA Wsh |
| "Consistency Checking in an Infrastructure for La | "A. Rauschmayer and A. Knapp | "Proc. 19\tex |
| "Tube---Structure-Orientation in a Prototype-B | "A. Rauschmayer and P. Renne | "Proc. Int. Co |
| "Scaling Step-Wise Refinement" | "Don Batory and Jack Sarvela a | "Proc. 25\tex |
| "Semantic-Web-Backed GUI Applications" | "A. Rauschmayer" | "ISWC Wsh. E |

**Fig. 7.** The "Projections" view in HYENA allows us to list all instances that can be displayed with a certain lens. Here we are displaying all instances of `bibtex:InProceedings`.



Main Graph bibtex:rausch...wc_hyena:2005 InProceedings Len...

Properties

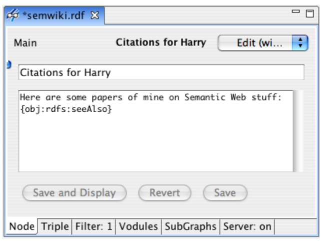
| | |
|---|---|
| bibtex:title | An RDF Editing Platform for Software Engineering |
| bibtex:author | A. Rauschmayer |
| bibtex:booktitle | ISWC Wsh. Semantic Web Enabled Software Engineering (SWESE) |
| bibtex:month | November |
| bibtex:year | 2005 |

**Fig. 8.** One projection of our InProceedings lens. In the top right corner you can see that using the lens is just another way of displaying a node.

- Now we can create the new WIKKED page to hold the citations for Harry, add `rdfs:seeAlso` properties referencing the BibTeX entries, and write a corresponding object link in WITL (Fig. 9(a)). The rendered result is shown in Fig. 9(b).
- To conclude, we do something a little more fancy: we want to have the same set of citations as before, but this time we want them displayed in a bullet list. Furthermore, if someone clicks on a link, we want to search Google for the paper title, in an external browser. This looks as follows:

```
Click on any of these papers to search for them via Google:
{ul:
    {obj:rdfs:seeAlso^^
        {li: {evalLater:{present:{:ANCHOR}}
              ^{util.google:{literal:{:bibtex:title}}}} } } }
```

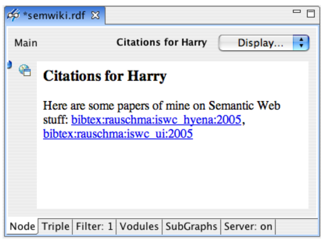
  We use an extended version of the object-linking function `obj` which has three arguments where the first one contains a property URI, the second options and the third an expression that will be re-evaluated once for each object. During each evaluation, the property values *of the current object* will be bound to WITL variables. Within the third argument, function `evalLater` displays a link (whose text is the first argument) and postpones evaluating its second argument until after the link has been clicked. The function we postpone is `util.google` which opens an external browser window with the



Citations for Harry

```
Here are some papers of mine on Semantic Web stuff:
{obj:rdfs:seeAlso}
```

(a) Editing the page



**Citations for Harry**

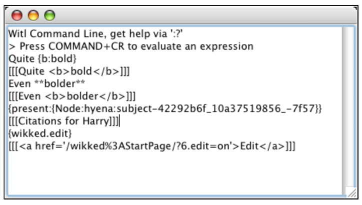
Here are some papers of mine on Semantic Web stuff: bibtex:rauschma:iswc_hyena:2005, bibtex:rauschma:iswc_ui:2005

(b) Displaying the page

**Fig. 9.** (Left) We are using an object link to encode the list of citations we want to tell Harry about. (Right) The citation list rendered via HTML.

result of a Google query. Furthermore, the helper function `present` lets us display a node as "prettily" as possible, i. e. it considers display aids such as an attached `rdfs:label`. Function `literal` extracts the plain text of a literal. Note that we cannot use wiki markup for iteratively creating the bullet list, because the scope of wiki markup does not extend beyond a single function brace.

## 5 Command Line and Web Serving

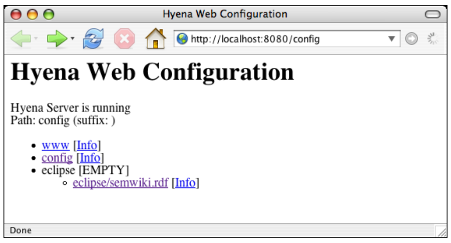


```
Witl Command Line, get help via ':?'
> Press COMMAND+CR to evaluate an expression
Quite {b:bold}
[[[Quite <b>bold</b>]]]
Even **bolder**
[[[Even <b>bolder</b>]]]
{present:{Node:hyena:subject-42292b6f_10a37519856_-7f57}}
[[[Citations for Harry]]]
{wikked.edit}
[[[<a href='/wikked%3AStartPage/?6.edit=on'>Edit</a>]]]
```

**Fig. 10.** An interactive session with the WITL command line: We evaluate bold markup in two variations; present a prettified version of a wiki page node; and show how editing is added to a page (it can be added to sets of pages via a skin). Note that one can make out in the produced HTML that the start page is currently selected in Hyena. Output is shown as it would be encoded in WITL.

Hyena also comes built-in with a Wikked command line where one can interactively execute WITL code (Fig. 10). What node is currently selected in Hyena is also reflected in the evaluation environment of the command line, so one can explore what source code will look like when evaluated "inside" a page.

Hyena, and thus Wikked, can be started in different modes: It can be started with Eclipse, as a plug-in. But it can also be started stand-alone, obviously with limited functionality. That last mode makes more sense when one



Hyena Web Configuration

http://localhost:8080/config

Hyena Web Configuration

Hyena Server is running
Path: config (suffix: )

- www [Info]
- config [Info]
- eclipse [EMPTY]
  - eclipse/semwiki.rdf [Info]

Done

**Fig. 11.** HYENA's document hierarchy: First is a container `www` that serves static web pages, second is the configuration container `config`, last is an Eclipse container with all currently open documents (currently, only "semwiki.rdf" is open). Every container is accessible via its URI, for example, you are currently seeing the answer to a GET request to the configuration container.

considers that HYENA can also be used for web-serving: either via Jetty, a small embeddable web server or as a servlet[5]. Jetty can be activated in conjunction with Eclipse. As a web server, HYENA manages the currently open files in a named hierarchy of *containers*. A unique name plus the server address results in a component having a unique URI. These URIs are used in HYENA's ReST-based web service [14, 15] API. While each container can provide a customized reaction to web requests the most common use cases are: GET requests display web pages and download data, PUT requests upload RDF files (including wiki definitions), POST requests allow one to react to form input and DELETE requests remove containers. Several standard containers[6] are available: container `www` serves static web pages stored in the standard HYENA directory, container `config` provides configuration and status information about currently active containers (Fig. 11) and, if Eclipse is running, there is an `eclipse` container that publishes all currently open HYENA documents. For example, we can display the running example in the web browser by clicking on `semwiki.rdf`.

## 6 Related Work

EclipseWiki [16] is a plugin for Eclipse that allows one to edit text files as wiki pages and to embed links to Eclipse resources. In contrast, WIKKED is more specialized: It has RDF integration and can even link to markers *inside* files.

ZML [17] from the Rhizome Wiki has markup rules that are similar to ours, but differs in two important aspects: First, it can "natively" express full HTML which WIKKED foregoes in order to support both LaTeX and HTML; one can still use raw text as a last resort in WIKKED. Second, ZML's syntax is purely line-based where WITL's syntax is a hybrid of bracketed and line-based constructs.

XSDoc Wiki [18] is used for integrating different development artifacts inside a wiki. Instead of RDF, it normalizes all data as XML and has powerful import

[5] Even with Jetty, HYENA establishes connection through a servlet.
[6] More containers can be added programmatically at any time.

facilities. XSDoc has more in common with WIKKED than is apparent at first glance: HYENA partly specializes in using RDF for software engineering. One can, for example, reify Java source code locations as RDF nodes and refer to them in WIKKED. While we do not have XSDoc's flexible format support, we do think that RDF is a better data format when it comes to *linking* and by using Eclipse to track locations, our linking becomes even more robust.

In the Wiki-Templates paper [19], the authors prominently express the need to support structure in wikis. Thus Wiki-Templates provide sophisticated support for structured data entry. Lenses provide much of the same functionality for WIKKED, but projections are not (yet) embeddable inside a wiki page. Furthermore, storing data in RDF has the advantages of being clearly defined and standardized, as opposed to the custom approach taken by Wiki-Templates.

## 7 Future Research

One way we could go with HYENA is to put more emphasis on its currently underdeveloped web-serving abilities. It would have to be based on Ajax [20] (or Comet [21]) and fully support Fresnel lenses and SPARQL queries. Additionally, we do not currently provide any way of authentication or encryption. Finally, it would be nice to combine blog and wiki into something that has been called a *Bliki* [22]. Because WIKKED is RDF-based and thus highly structured, this would be a very natural extension, as opposed to some of the kludges that currently exist.

## 8 Conclusion

In this paper, we have shown how HYENA and WIKKED tackle the problem of integrating structured and semi-structured data. This problem is very pertinent to wikis and even more so to semantic wikis. Our answer is separation of concerns: HYENA edits (structured) RDF data, whereas the embedded wiki engine WIKKED is responsible for displaying (semi-structured) wiki content "inside" RDF nodes. We found that this separation of concerns has great usability advantages. It also turns the typical semantic wiki inside out: the wiki is embedded in an RDF editor and not the other way around. WIKKED has been adapted to RDF in one important way: links to data (including other pages) are reified as true RDF relations and not hidden in the wiki text. That means that WIKKED pages are relatively robust regarding changes in the RDF such as node renaming. Finally, WIKKED further enhances HYENA by letting the user invoke HYENA functionality from within a wiki page. As a result, one can make wiki pages more interactive.

*Acknowledgements.* The authors would like to thank Hubert Baumeister for introducing them to Wikis, when they were still an insider tip; and the SemWiki workshop reviewers for their comments and their contributions to the section on related work.

## References

1. Wikipedia: Semantic Wiki. http://en.wikipedia.org/wiki/Semantic_Wiki (2006) [Online; accessed 2006-05-02].
2. Rauschmayer, A.: A Short Introduction to RDF for Software Engineers. (2005)
3. Manola, F., Miller, E.: RDF Primer, W3C Recommendation. http://www.w3.org/TR/rdf-primer/ (2004)
4. Leuf, B., Cunningham, W.: The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley (2001)
5. Eclipsepedia: Eclipse Rich Client Platform. (http://wiki.eclipse.org/index.php/Rich_Client_Platform) [Online; accessed 2006-05-02].
6. Carroll, J., et al.: Named Graphs. (http://www.w3.org/2004/03/trix/) W3C Interest Group.
7. Broekstra, J., Kampman, A., Mika, P., et al.: Sesame Home Page. (http://www.openrdf.org/)
8. Semantic Web Interest Group: Fresnel - Display Vocabulary for RDF. http://www.w3.org/2005/04/fresnel-info/ (2005)
9. : The Friend of a Friend (FOAF) project. (http://www.foaf-project.org/) [Online; accessed 2006-05-02].
10. Berners-Lee, T.: Notation3 (N3) a Readable RDF Syntax. (http://www.w3.org/DesignIssues/Notation3.html)
11. Wikipedia: S-expression. http://en.wikipedia.org/wiki/S-expression (2006) [Online; accessed 2006-05-04].
12. Bloch, J., et al.: A Metadata Facility for the Java Programming Language (2002) Java Specification Request 175.
13. Siberski, W.: bibtex2rdf - A Configurable BibTeX to RDF Converter. (http://www.l3s.de/ siberski/bibtex2rdf/)
14. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
15. Prescod, P.: Second Generation Web Services. http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html (2002)
16. Walton, L., Walton, C.: Eclipse Wiki Editor Plugin. (http://eclipsewiki.sourceforge.net/)
17. Liminal Systems: ZML (Zippy Markup Language). (http://www.liminalzone.org/ZML)
18. Aguiar, A., David, G.: WikiWiki Weaving Heterogeneous Software Artifacts. In: Proc. Int. Symp. Wikis, ACM Press (2005)
19. Haake, A., Lukosch, S., Schümmer, T.: Wiki Templates—Adding Structure Support to Wikis on Demand. In: Proc. Int. Symp. Wikis, ACM Press (2005)
20. Garrett, J.J.: Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php (2005)
21. Russell, A.: Comet: Low Latency Data for the Browser. http://alex.dojotoolkit.org/?p=545 (2006)
22. Fowler, M.: What is a Bliki? http://www.martinfowler.com/bliki/WhatIsaBliki.html (2003)

# A Semantic Wiki for Mathematical Knowledge Management

Christoph Lange[1] and Michael Kohlhase[2]

[1] Informatik, Universität Trier, lange@castor.uni-trier.de
[2] Computer Science, International University Bremen, m.kohlhase@iu-bremen.de

**Abstract.** We propose the architecture of a semantic wiki for collaboratively building, editing and browsing a mathematical knowledge base. Its hyperlinked pages, containing mathematical theories, are stored as OMDoc, a markup format for mathematical knowledge representation. Our long-term objective is to develop a software that, on the one hand, facilitates the creation of a shared, public collection of mathematical knowledge (e.g. for education). On the other hand the software shall serve work groups of mathematicians as a tool for collaborative development of new theories.

## 1 A Semantic Web for Science and Technology via Mathematical Knowledge Management (MKM)

The Internet plays an ever-increasing role in our everyday life, and science is no exception. It is plausible to expect that the way we do (conceive, develop, communicate about, and publish) mathematics will change considerably in the next ten years. In particular, most of the mathematical activities will be supported by mathematical software systems (we will call them *mathematical services*) connected by a commonly accepted distribution architecture. It is a crucial but obvious insight that true cooperation of mathematical services is only feasible if they have access to a joint corpus of mathematical knowledge[3]. Therefore, a central prerequisite for this is the creation of a technology that is capable to create, maintain, and deploy *content-oriented* libraries of mathematics on the web. The world wide web is already now the largest single resource of mathematical knowledge, and its importance will be exponentiated by the emerging display technologies like MathML, which integrates LaTeX-quality presentation into the hypertext and multimedia capabilities of the WWW.

The **Semantic Web** is a *Web* of *data* for *applications*, just as the WWW is a web of documents for humans.

If we extend this vision of Tim Berners-Lee's to mathematics on the web, many services come into mind:

[3] Be it one central knowledge base or many of them glued together through an exchange mechanism

1. cut and paste on the level of computation (take the output from a web search engine and paste it into a computer algebra system).
2. automatically checking published proofs, if they are sufficiently detailed and structured.
3. math explanation (e.g. specializing a proof to an example that simplifies the proof in this special case).
4. semantic search for mathematical concepts (rather than keywords): "Are there any objects with the group property out there?"
5. data mining for representation theorems: "Are there undiscovered groups out there?"
6. classification: given a concrete mathematical structure, is there a general theory for it?

All of these services can currently only be performed by humans, limiting the accessibility and thus the potential value of the information. On the other hand, the content-oriented mathematical libraries can only be generated by humans, as it has been proved by the successful *PlanetMath* project[4], which features free, collaboratively created entries on more than 8,000 mathematical concepts. *PlanetMath*, however, is not completely machine-understandable. There is a fixed set of metadata associated with each article, including its type (definition, theorem, etc.), parent topic, Mathematics Subject Classification, synonyms and keywords, but the content itself is written in LaTeX and can only be searched in full-text mode.

## 2 Semantic MK Markup with OMDoc

We will make use of the structural/semantic markup approaches using formats such as OpenMath [BCC+04], MathML [ABC+03], and OMDoc (Open Mathematical Documents [Koh06]), the latter of which embeds and extends the former ones. These formats, constituting the state of the art for representing mathematical knowledge, are now used in a large set of projects in automated theorem proving, eLearning, ePublishing, and in formal digital libraries. OMDoc builds on a semantic representation format for mathematical formulae (OpenMath objects or Content MathML representations) and extend this by an infrastructure for context and domain models from "formal methods". In contrast to those, these structural/semantic approaches do not require the full formalization of mathematical knowledge, but only the explicit markup of important structural properties. For instance, a statement will already be considered as "true" if there is a proof object that has certain structural properties, not only if there is a formally verifiable proof for it. Since the structural properties are logic-independent, a commitment to a particular logical system can be avoided without losing the automatic knowledge management which is missing for semantically unannotated documents. Work on the OMDoc format shows that most added-value services in knowledge management do not need tedious

[4] http://www.planetmath.org, see also [Kro03]

formalization, but can be based on the structural/semantic level. OMDoc assumes a three-layered structure model for semantic representation formalisms:

**Object level:** represents objects such as complex numbers, derivatives, equations etc. Semantic representation formats typically use functional characterizations that represent objects in terms of their logical structure, rather than specifying their presentation. This avoids ambiguities which would otherwise arise from domain specific representations.

**Statement Level:** (natural/social/technological) sciences are concerned with modeling our environment, more precisely with statements about the objects in it. We can distinguish different types of statements: model assumptions, their consequences, hypotheses, and measurement results. All of them have in common that they state relationships between scientific objects and have to be verified or falsified in theories or experiments. Moreover, all these statements have a conventionalized structure, such as Exercise, Definition, Theorem, Proof, and a standardized set of relations among each other. For instance, a model is fully determined by its assumptions (also called *axioms*); all consequences are deductively derived from them (via *theorems* and *proofs*), and therefore their experimental falsification uncovers false assumptions of the model.

**Theory/Context Level:** Representations always depend on the ontological context; even the meaning of a single symbol[5] is determined by its context, and depending on the current assumptions, a statement can be true or false. Therefore the sciences (with mathematics leading the way) have formed the habit to fix and describe the situation of a statement. Unfortunately, the structure of these situation descriptions remains totally implicit, and can therefore not be used for computer-supported management. Semantic representation formats make this structure explicit. In mathematical logic, a theory is the deductive closure of a set of axioms, i.e. the (in general infinite) set of logical consequences of the model assumptions. Even though this fully explains the phenomenon context in theory, important aspects like the re-use of theories, knowledge inheritance, and the management of theory changes are disregarded completely. Therefore, formalisms with context level use elaborate inheritance structures for theories, e.g. in form of ontologies in the Semantic Web or in form of "algebraic specifications" in program verification.

An important trait of the three-layer language architecture is the inherent dependency loop between the object and theory levels mediated by the statement level: the objects obtain their meaning from the theories their functional components are at home in, and the theories are constituted by special statements, and in particular the objects that are contained in them. Making these structures explicit enables the mechanization and automation of knowledge management and the unambiguous, flexible communication of mathematical objects

[5] e.g. the glyph $h$ as the height of a triangle or Planck's quantum of action.

and knowledge that is needed for meaningful interoperability of software systems in science.

## 3 Cross-Fertilization of MKM and Wiki

Even though the work reported here was initially motivated by solving the MKM author's dilemma (see below), we contend that the new application area MKM can also contribute to the development of semantic wikis.

### 3.1 Benefits of a Wiki for MKM

As any semantic or traditional wiki, a wiki environment for MKM encourages users to collaborate: Non-mathematicians can collaborate in creating a "Wikipedia of mathematics" by compiling the knowledge available so far, while scientists can collaboratively develop new theories. However, to encourage users to contribute, wiki-like openness to anybody probably won't suffice. Unlike the text formats used by common semantic wikis, the OMDoc format makes the fine-grained semantic structure implicit in the text explicit in the markup, making it tedious to author by hand. Moreover, only after a substantial initial investment (writing, annotating, and linking) on the author's part, the community can benefit from the added-value services supported by the format — e.g. the creation of customized textbooks [MS04]. If author and beneficiary of such services were different persons, though, only few persons would be willing to contribute to a knowledge base. This "MKM author's dilemma" [KK04] can be overcome when the authors themselves are rewarded for their contributions by being offered added-value services, which improve immediately the more annotations and cross-references the users contribute, — for example a facility for navigation through the knowledge base along paths of semantic relations between the theories, which are computed from the OMDoc document collection.

Furthermore, mathematicians developing theories will be assisted to retain an overview of theory dependencies in order not to break them. Social software services will further utilize the semantic information available from the theories and from tracking the user interaction log ("Who did what on which page when?").

### 3.2 An Alternative 'Semantic Web'

Most semantic wikis are based on ideas and techniques from Berners-Lee's Semantic Web. In accordance with the general definition in the introduction, the Semantic Web uses RDF triples [LS99] to describe resources such as XML fragments in documents and the background knowledge in ontologies to draw inferences about their content. Note that the Semantic Web makes a conceptual *division between data* (arbitrary objects — called "resources" — that can be identified by URI references; usually XML fragments) and *context* (encoded in topic maps, or an ontology language like OWL [W3C04] or KIF [Gea92]). In contrast to this, content/context markup systems like OMDoc consider scientific *knowledge as the primary data* and take the *context* to be made up of *reified knowledge* (see the discussion in section 2). This makes collections of OMDoc documents into *referentially closed systems* (all the knowledge referred to can be expressed in the system itself), which in turn allows *ontological bootstrapping* (the ontologies needed to draw inferences can be built up as we build up the data). Note that only part of the mathematical knowledge embedded in mathematical documents can be exploited for ontological reasoning[6], as it cannot faithfully be expressed in first-order logic (much less so in description logics). Consider for instance the following fragment from a math book:

**Definition**: $f \in \boxed{\mathcal{C}^0(\mathbb{R},\mathbb{R})}$, iff for all $x, y \in \mathbb{R}$ and $\epsilon > 0$, there is a $\delta > 0$, such that $|f(x) - f(y)| < \epsilon$ if $|x - y| < \delta$.
**Definition**: $f \in \boxed{\mathcal{C}^0(\mathbb{R},\mathbb{R})}$, iff for all $x \in \mathbb{R}$ and $\epsilon > 0$, there are $f'(x)$ and $\delta > 0$, such $\left|\frac{|f(x)-f(x+h)|}{h} - f'(x)\right| < \epsilon$ for $h < \delta$.
**Examples**: If $f(x) := |x|$ and $g(x) := 3x^2 + 2x - \pi$, then $\boxed{f \in \mathcal{C}^0(\mathbb{R},\mathbb{R})}$ and $\boxed{g \in \mathcal{C}^1(\mathbb{R},\mathbb{R})}$, but $\boxed{f \notin \mathcal{C}^1(\mathbb{R},\mathbb{R})}$.
**Theorem**: $\boxed{\mathcal{C}^1(\mathbb{R},\mathbb{R}) \subseteq \mathcal{C}^0(\mathbb{R},\mathbb{R})}$
**Proof**: Let $f \in \mathcal{C}^0(\mathbb{R},\mathbb{R})$, $x \in \mathbb{R}$ and $\delta = \epsilon > 0$, then $|f(x) - f(y)| \leq h \cdot |f(x)| \ldots$

Here, only the boxed fragments contain taxonomic information. Its justifications via $\epsilon/\delta$ arguments cannot be (simultaneously) be expressed in description logics. Thus any web ontology that deals with objects such as the ones above will necessarily have to approximate the underlying mathematical knowledge.

Generally in science, knowledge comes in documents and constitutes the context, whereas description logic ontologies only reference and approximate the knowledge in a document. Therefore, with OMDoc we propose an *alternative vision for a 'semantic web for science and technology'* where the ontologies necessary for drawing inferences are views derived from normative documents. Where ontological fragments cannot be derived automatically (an interesting research problem in itself), they can be embedded into OMDoc-encoded documents as OWL, and later simply extracted. Thus OMDoc — as an document format with embedded taxonomic information — serves as its own ontology language.

## 3.3 Opportunities of MKM for Semantic Wikis

The enhancements of the data model semantic wikis bring along — compared to traditional wikis — are already present in the OMDoc format, so that an OMDoc-based wiki only needs to operationalize their underlying meaning. For

[6] For the sake of this argument we will use the term web ontology language synonymously with "description logic", as in OWL-DL; if we pass to more expressive logics like KIF, we lose decidability and thus the raison d'être for web ontologies.

example, typed links, which are implemented via an extension to the wiki syntax in *Semantic MediaWiki* [VKVH06] or editable through a separate editor in *IkeWiki* [Sch06], are implemented by means of the `for` attribute to OMDoc's elements (e.g. `<example for="#id-of-assertion">`). It remains left to the wiki to make them editable easily and to visualize them adequately.

More than a general semantic wiki, one targeted at mathematics must ensure that dependencies between concepts are preserved (see section 4.3). Results in this area will be interesting for non-mathematical semantic wikis as well, especially when they support higher levels of formalization such as ontologies.

## 4 Design of the OMDoc Wiki

Before we can go into the design of the OMDoc wiki system and the user interaction — including Web-2.0-like added-value services, we will concern ourselves with its information model: what a wiki page should comprise, what semantic information can be inferred from the OMDoc documents and the user interaction logs, and finally how this can be utilized.

### 4.1 What Should a Page Contain?

The smallest unit in a wiki that can be displayed, edited, linked to, or archived is a page. While in a non-semantic wiki, one page can have arbitrary contents, in a semantic wiki it usually describes one *concept*, including its properties and its relations to other concepts.

OMDoc groups closely related concepts into 'theories' and advises to follow a 'little theories approach' [FGT92], where theories introduce as few new concepts as possible[7]. We follow this intuition and restrict the pages of the OMDoc wiki to single (little) theories to keep them manageable. Moreover, OMDoc distinguishes the knowledge elements in theories into constitutive ones like symbols, definitions, and axioms (these are indispensable for the meaning of the theory) and non-constitutive ones, such as assertions, their proofs, alternative definitions of concepts already defined, and examples. We insist that the latter are rolled out into separate theories (and therefore wiki pages). Small pages also improve the effectivity of wiki usage, as they facilitate re-use by linking and allow for a better overview through lists of recent changes and other automatically generated index pages.

Each theory page has an associated discussion page, which provides an adequate space for questions, answers, and discussions about this theory. OMDoc will be used for discussion pages as well, with some proposed extensions for discussion posts: New elements for questions, explanations, opinions, etc. will be added.

[7] A theory may introduce more than one concept, if they are interdependent, e.g. to introduce the natural numbers via the Peano Axioms, we need to introduce the set of natural numbers, the number zero and the successor function at the same time.

### 4.2 Utilizable Semantic Information

From the OMDoc wiki we can gain several kinds of semantic information, formally expressed as relations between concepts: First, there are *basic* relations provided by the individual theories. Then, there are basic relations given by the user interaction logs. Further, *inferable* relations can be defined as closures of the former and as unions of theory relations and interaction relations. Finally, there are other useful relations that the authors have to provide by manual annotations. All these definitions of relations are part of the system ontology[8] of the wiki, which will not be editable by the user.

**Relations Provided by the OMDoc Theories** Theories are related by theory imports (see section 2, "Theory Level") and by relations between their statements ("Statement Level"). For example, if theory $t$ states an assertion using symbols defined in the theories $t'$ and $t''$ or proves an assertion made in $t'$ using a theorem from $t''$ as a premise, $t$ is related to $t'$ and $t''$, but the individual statements are also related to each other in a more fine-grained view.

Semantic information will only be extracted from the theory and statement levels of OMDoc — directly or through reasoning in the case of transitive closures —, not from the object level[9]. The most important relation our application utilizes is the dependency relation between theories, defined by theory import declarations, and the acyclic graph formed by this relation.

**Relations Given by User Interaction** The basic relation given by user interaction is, "Who *edited* which theory page when?". This information is available for free in a wiki; it can be logged when a page is saved. Accordingly, a relation could be defined which states that a user *read* a theory. This is, however, hard to determine because of HTTP caching. Further relations are defined by user *feedback* to navigation choices proposed by the wiki (see section 4.3).

**Inferable Relations** Further relations can be inferred from those introduced so far, for example a metric estimating the *degree of difficulty* of a page, calculated by counting the questions on the discussion page. From the user interaction log, sets of related pages can be identified, which are not already related through dependency. For this purpose, a notion of *transaction* must be introduced, i.e. edits carried out by the same user in short succession. Similarly to products bought in online shops, two theories are considered "related" when many users edited them in the same transactions.

Even more sophisticated relations can be inferred from both OMDoc and interaction relations. The software could, for example, track how many examples

[8] not to be confused with the *domain ontology* (for mathematics) embedded in the OMDoc theories.

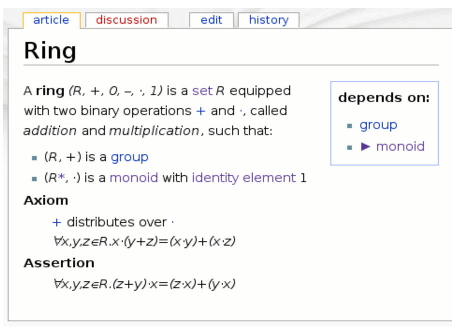
[9] The latter would be suitable for a future integration of computer algebra systems or automated theorem provers.

to a theory users read and improve the difficulty estimation by including those statistics.

### 4.3 User Interface and Interaction Model

**Rendering** Theory pages are presented to the user in a human-readable form (XHTML plus presentation MathML) generated by a style sheet. The XHTML contains inline hyperlinks to other theories where appropriate, for instance, from an example to the concept or assertion it explains. As OMDoc documents, however, need not contain any human-readable sections or comments — after all, the knowledge base might be used to support a theorem prover, not to create a textbook! — there is also a source code view with with lines indented, keywords highlighted and URIs displayed as hyperlinks. An intermediate view mode renders mathematical objects in the source code as formulae using MathML or TeX-generated images.

**Dynamic Navigation Links** Navigation bars with fixed links, such as links to global special pages like the recent changes list, as well as dynamic links to theories depending on the theory $t$ being displayed or related otherwise are provided. Links anchored to particular statements are rendered inline, but links anchored to whole theories — as, for example, imports — must be displayed on a navigation bar. If morphisms from the imported theory to the importing theory are used, as is the case with the import from *monoid* to *ring*, which is used to define that a ring is a monoid w.r.t. multiplication[10], they are also displayed on request. The triangle next to the link to *monoid* in the figure points out that a morphism has been specified.



Ring

A **ring** *(R, +, 0, −, ·, 1)* is a set *R* equipped with two binary operations + and ·, called *addition* and *multiplication*, such that:

- (*R*, +) is a group
- (*R**, ·) is a monoid with identity element 1

**Axiom**

+ distributes over ·
$\forall x,y,z \in R. x\cdot(y+z)=(x\cdot y)+(x\cdot z)$

**Assertion**

$\forall x,y,z \in R. (z+y)\cdot x=(z\cdot x)+(y\cdot x)$

**depends on:**

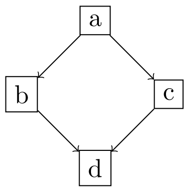
- group
- ▶ monoid

Dynamic navigation links improve usability by answering the questions “Where am I?” and “Where can I go?” [Nie99]. If dynamic linking directly depends on the page contents editable by the user, as is the case with theory dependency, users are instantaneously gratified for contributing to the structure of dependency by creating connections between theories [Aum05, sec. 3.2].

**Navigating the Dependency Graph** Not only will the user be able to navigate along the dependency graph, she will also be able to *interact* with the system: she will be asked whether she wants to explore the theories required as dependencies in further detail.

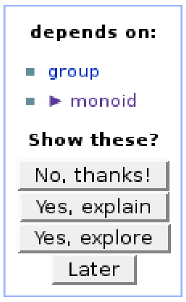
[10] This morphism basically maps the monoid’s $\circ$ operator to the ring’s multiplication operator $\cdot$ and renames the identity element from $e$ to 1.

Suppose that the user is currently reading theory $a$, which depends on $b$ and $c$, which in turn depend on theory $d$[11]. In this case the wiki will not only display navigation links to the direct dependencies $b$ and $c$, but it will also provide unobtrusive buttons that allow the user to give one of the following acknowledgments:



$$\begin{tikzcd}
 & \boxed{a} \arrow[dl] \arrow[dr] & \\
\boxed{b} \arrow[dr] & & \boxed{c} \arrow[dl] \\
 & \boxed{d} &
\end{tikzcd}$$

**No, thanks!** "I already know $b$ and $c$, please let me just read about $a$."
**Explain** "Please show me $b$ and $c$ so that I can learn about $a$'s prerequisites." — $b$ and $c$ will be displayed.
**Explore** "Please show me *all* prerequisites for $a$." — In our example, these are $b$, $c$, and $d$, which could be opened in separate windows or serialized into one page.
**Suspend** "I want to know about $b$ and $c$, but only later." — The system keeps a notice in the user's profile that she wants to read $b$ and $c$ sometime. Reminder links to suspended theories are shown on a separate navigation bar.



Not only the last case will be recorded — the others are interesting as well for the purpose of *social bookmarking*. For example, if many users requested a theory $t$ to be explained, the system could default to display not only the direct dependencies but also the level-two dependencies, for it seems that $t$ is too difficult for only being explained shallowly.

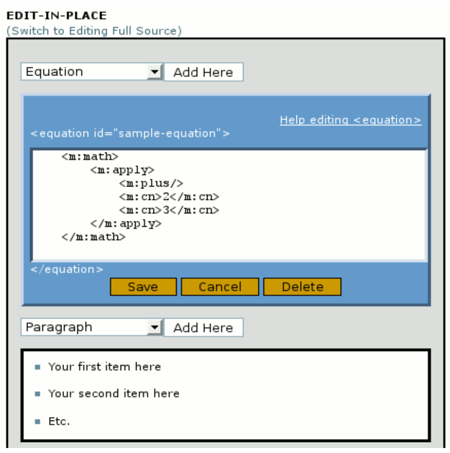
Furthermore, the system does not only keep track of which theories the user wants to be explained, but also which theories the user has already learned. For each theory, a button will be offered for telling the system "I have learned this". Links to theories learned can then be disabled or displayed in a more unobtrusive color than links to theories that are new to the user.

**Preserving Dependencies on Editing** So far, there has not been any approach to preserving dependencies between pages in a semantic wiki. Tracking dependencies and reasoning about them is an integral part of mathematical practice. Known results are often generalized to find the "real" dependencies, mathematical theories and books are rewritten to make dependencies minimal. Therefore this problem cannot be neglected in a mathematical wiki. In the special case of OMDoc, where dependencies need not be formally verifiable when they have sufficient structural properties (see section 2), a dependency could formally be broken but seem intact to the system anyway. Therefore, we propose a first, simple approach to this problem; a more sophisticated "management of change" process could be integrated later on the basis of work in formal methods [Hut04,AHMS02].

If a theory $t$ depends on a theory $t'$, which can be edited independently from $t$, modifying $t'$ could break $t$ because some definition in $t'$ required by $t$ might

[11] See [Koh06, fig. 6.1] for a real-world example of such a diamond graph.

have been changed fundamentally[12]. The OMDoc wiki keeps the knowledge base consistent by making hyperlinks not to theories in general, but to certain versions of them. When an author enters a link to `group-theory`, for example, this reference will be stored as `group-theory/latest`. On the other hand, the author of $t$ depending on $t'$ should be notified about updates to $t'$ so that he can benefit from improvements made there. Such notifications can appear statically on the author's watch list[13], but also dynamically in an area near the editing box, while $t$ is being edited. The author then can decide whether to adjust his references from `t'/old` to `t'/improved` — depending on whether $t'$ has really been improved (e.g. with corrections or additional documentation) rather than changed in a dependency-breaking way. The other way round, a user editing $t'$ will be notified that there is a theory $t$ depending on the one he is editing and can decide whether to upgrade $t$'s references to $t'$ or to leave it.



```
<equation id="sample-equation">
    <m:math>
        <m:apply>
            <m:plus/>
            <m:cn>2</m:cn>
            <m:cn>3</m:cn>
        </m:apply>
    </m:math>
</equation>
```

- Your first item here
- Your second item here
- Etc.

**User-friendly Editing** The simplest user interface for editing a wiki page is a text area showing the whole contents of the page. As editing OMDoc theories this way is tedious, our wiki will provide alternatives.

The Ajax-based Edit-in-place interface from *Rhaptos* (the software run by *Connexions* [CNX06,The06b], a community-driven collection of open educational content) will be tailored to editing OMDoc. Edit-in-place [The06a] can insert or edit several types of page sections: paragraphs, equations, lists, and more. All sections are displayed in a near-WYSIWYG view, but clicking one of them replaces its view by a text area containing its XML source. Three buttons are displayed below the text area: "Cancel", "Save", and "Delete", the latter two of which commit the editing transaction by sending an asynchronous request to the server.

While Edit-in-place facilitates editing OMDoc on theory and statement level, it is not helpful on the object level because...

1. Mathematical formulae are deeply nested in most cases, while "Edit in place" has been designed with flat XML structures in mind.
2. There are shorter and more intuitive notations for formulae than OpenMath or Content MathML.

Therefore the OMDoc wiki allows for entering mathematical objects in the simpler syntax of *QMath* [Pal06], a batch processor that transforms plain text

[12] It would be good style to copy $t'$ to a new theory with a different name, anyway.
[13] Watch lists are, for example, known from *MediaWiki*

to OMDoc. *QMath* uses tables mapping text symbols to their OMDoc or OpenMath representation; these tables are also made editable in the wiki. The wiki will keep mathematical objects entered in *QMath* in this format for usability reasons, only converting them to OMDoc when pages are exported to another application.

The same way as *QMath* facilitates the creation of mathematical formulae, wiki text syntax will be offered as a simple way to enter OMDoc's rich text [Koh06, sec. 14.5].

## 5 Implementation Notes

The OMDoc wiki presented in this paper is currently in a prototype stage under active development. Once completed, it will be released under an open source license; for earlier versions, please contact the authors. We have based our system on *IkeWiki*[14] system as a development platform because of its

**Modular Design of Backend and GUI:** There are separate stores for page contents and the knowledge base. After the XML-encoded contents of a page have been read from the database, small modules — so-called "wiklets" — perform tasks like enriching the DOM tree of the page with navigation side bars created from semantic annotations, and then the enriched page is rendered for presentation using customizable XSLT style sheets.

**Rich Semantic Web Infrastructure:** IkeWiki supports many standards of the Semantic Web. The knowledge base is stored as RDF; OWL-RDFS reasoning and SPARQL queries are supported.

**User Assistance for Annotation:** Editors for page metadata and link types, which can likely be utilized for editing OMDoc, are available.

**Orientation Towards Learning:** One objective in IkeWiki's ongoing development is its expansion towards a learning environment [SBB$^{+}$06]. Upcoming versions will likely qualify as a base for an OMDoc wiki with learning features (see section 4.3).

Some parts of the OMDoc wiki will, however, be very different from *IkeWiki*'s operating principles and hence require substantial amounts of refactoring and rewriting — for example:

- The presentation view of a page, for example, cannot be generated by a single-pass XSL transformation from OMDoc to XHTML+MathML; instead, the multi-level OMDoc presentation workflow [Koh06, sec. 25] has to be adopted.
- The semantic relations between OMDoc theories are not exclusively stored in the RDF knowledge base, as is the case with semantic relations between *IkeWiki* pages; instead, the OMDoc wiki has to keep the annotations in OMDoc synchronized with the knowledge base, which will still be used for reasoning.

[14] http://ikewiki.salzburgresearch.at, see also [Sch06]

## 6 Conclusion and Outlook

*Mission ...* The upcoming release of the OMDoc wiki presented in this paper will offer a user friendly editor for OMDoc's XML source code (section 4.3). Pages are viewable as XHTML+MathML as well as hyperlinked source code (4.3). Semantic relations, to be displayed on a navigation bar, will be inferred from the dependency relation between theories (4.3). Learning will be supported through interactive navigation along the dependency graph (4.3). There will be a simple assistance helping users preserve dependencies (4.3).

Later we will improve display of semantic relations, also taking into account the more fine-grained relations inferable from OMDoc's statement level (4.2) and from user interaction alone (4.2). Once techniques for management of changes to OMDoc documents have been developed, they will be integrated into the wiki to offer a more sophisticated dependency preservation.

*... and Vision* With the OMDoc wiki we pursue an alternative vision of a 'Semantic Web'. Like Tim Berners-Lee's vision we aim to make the web (here mathematical knowledge) machine-understandable instead of merely machine-readable. However, instead of a top-down metadata-driven approach, which tries to approximate the content of documents by linking them to web ontologies (expressed in terminological logics), we explore a bottom-up approach and focus on making explicit the intrinsic structure of the underlying scientific knowledge. A connection of documents to web ontologies is still possible, but a secondary effect: In OMDoc we can have explicit taxonomic relations as in "all rings are groups" — where the taxonomy is given by definition — or even implicit ones as in "all differentiable functions are continuous" — where the taxonomy is expressed by a theorem. If these theorems and definitions are of a suitable form, or explicitly indicated to be taxonomic by the author, we can harvest this information and transform it into a web ontology format such as OWL [W3C04] and make it available to the Semantic Web.

## References

[ABC$^+$03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at http://www.w3.org/TR/MathML2.

[AHMS02] Serge Autexier, Dieter Hutter, Till Mossakowski, and Axel Schairer. The development graph manager MAYA (system description). In Hélène Kirchner and Christophe Ringeissen, editors, *Algebraic Methodology and Software Technology — 9th International Conference AMAST 2002*, number 2422 in LNCS, pages 495–500. Springer Verlag, 2002.

[Aum05] David Aumüller. SHAWN: Structure Helps a Wiki Navigate. In W. Müller and R. Schenkel, editors, *Proceedings of the BTW-Workshop "WebDB Meets IR"*, March 2005.

[BCC$^+$04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. http://www.openmath.org/standard/om20.

[CNX06] CONNEXIONS. Project home page at http://cnx.org, seen March 2006.

[FGT92] William Farmer, Josuah Guttman, and Xavier Thayer. Little theories. In D. Kapur, editor, *Proceedings of the 11th Conference on Automated Deduction*, volume 607 of *LNCS*, pages 467–581, Saratoga Spings, NY, USA, 1992. Springer Verlag.

[Gea92] M. Genesereth and R. Fikes et al. Knowledge interchange format: Version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[Hut04] Dieter Hutter. Towards a generic management of change. In Christoph Benzmüller and Wolfgang Windsteiger, editors, *Computer-Supported Mathematical Theory Development*, number 04-14 in RISC Report Series, pages 7–18. RISC Institute, University of Linz, 2004. Proceedings of the first "Workshop on Computer-Supported Mathematical Theory Development" held in the frame of IJCAR'04 in Cork, Ireland, July 5, 2004. ISBN 3-902276-04-5. Available at http://www.risc.uni-linz.ac.at/about/conferences/IJCAR-WS7/.

[KK04] Andrea Kohlhase and Michael Kohlhase. CPoint: Dissolving the author's dilemma. In Andrea Asperti, editor, *Mathematical Knowledge Management, MKM'04*, number 3119 in LNAI, pages 175–189. Springer Verlag, 2004.

[Koh06] Michael Kohlhase. OMDOC *An open markup format for mathematical documents (Version 1.2)*. LNAI. Springer Verlag, 2006. to appear, manuscript at http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf.

[Kro03] Aaron Krowne. An architecture for collaborative math and science digital libraries. Master's thesis, Virginia Tech, 2003.

[LS99] Ora Lassila and Ralph R. Swick. Resource description framework (RDF) model and syntax specification. W3C recommendation, World Wide Web Consortium (W3C), 1999. http://www.w3.org/TR/1999/REC-rdf-syntax.

[MS04] E. Melis and J. Siekmann. Activemath: An intelligent tutoring system for mathematics. volume 3070, pages 91–101. Springer-Verlag, 2004.

[Nie99] Jakob Nielsen. *Designing Web Usability : The Practice of Simplicity.* New Riders Press, 1999.

[Pal06] Alberto González Palomo. QMath: A human-oriented language and batch formatter for OMDoc. In OMDOC *An open markup format for mathematical documents (Version 1.2)* [Koh06], chapter 27.2. to appear, manuscript at http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf.

[SBB$^+$06] Sebastian Schaffert, Diana Bischof, Tobias Bürger, Andreas Gruber, Wolf Hilzensauer, and Sandra Schaffert. Learning with semantic wikis. http://www.wastl.net/download/paper/Schaffert06_SemWikiLearning.pdf, 2006.

[Sch06] Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. Technical report, Salzburg Research Forschungsgesellschaft, 2006.

[The06a] The Connexions Team. Connexions: Help on editing modules. http://cnx.org/help/EditingModules, 2006.

[The06b] The Connexions Team. Connexions: Sharing knowledge and building communities. http://cnx.org/aboutus/publications/ConnexionsWhitePaper.pdf, 2006.

[VKVH06] Max Völkel, Markus Krötzsch, Denny Vrandečić, and Heiko Haller. Semantic Wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, May 2006.

[W3C04] OWL web ontology language overview. W3C Recommendation, World Wide Web Consortium, February 2004.

# Towards a Semantic Wiki-Based Japanese Biodictionary

Hendry Muljadi, Hideaki Takeda, Shoko Kawamoto, Satoshi Kobayashi, and Asao Fujiyama

National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
{hendry, takeda, skawamot, satoshi-k, afujiyam}@nii.ac.jp
http://www.nii.ac.jp

**Abstract.** This paper describes an on-going project to develop and maintain a web-based Japanese Biodictionary within a Semantic Wiki environment. For the development of the dictionary, MediaWiki is extended to enable the writing of labeled links that represent RDF triples. The extension enables Semantic Wiki to provide not only collaborative environment for experts in various biology fields to create and edit the dictionary, but also the navigation support to manage relations between terms. Using a simple wiki syntax, people can develop and maintain the dictionary visually and easily.

## 1 Introduction

Developing a web-based Japanese Biodictionary is obviously a hard and time-consuming task. The project for the development of a web-based Japanese Biodictionary has done a survey over 77 Japanese textbooks in various biology fields, which include textbooks for high-school, undergraduate and graduate students. There are more than 30,000 different terms extracted for the web-based dictionary. However, writing the description of each term is another thing to do. It will cost a lot of money to pay for the copyright if we use the description written on the textbooks or other dictionaries. On the other hand, as the web-based Japanese Biodictionary is also targeting high-school students as its viewers, it is necessary to provide an easy-to-understand description of the terms as well as the relations between terms [1].

Learning from the development of Wikipedia[1], we consider that collaboration on the web is perhaps the best solution for the development of the dictionary. By allowing experts in various fields of biology to collaborate, the dictionary may well be developed. However, as we implemented the MediaWiki[2] in its original form, we found that users cannot describe the relation between terms flexibly. Allowing the MediaWiki to write labeled links can solve this problem.

This paper presents the development of a Semantic Wiki-based Japanese Biodictionary. Section 2 presents the features of the proposed Semantic Wiki, which is

[1] http://en.wikipedia.org/wiki/Wikipedia
[2] http://www.mediawiki.org

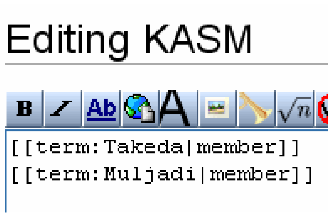
called MewKISS. Section 3 presents the current state of the Semantic Wiki-based Japanese Biodictionary. The conclusion of this paper is presented in section 4.

## 2 Semantic Wiki

### 2.1 Extending MediaWIki

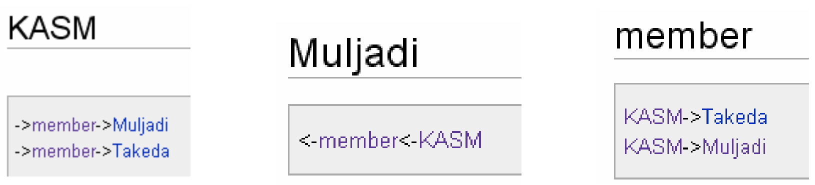
MediaWiki is a very useful tool for collaborative content management. MediaWiki also has a category management function which allows users to create class and sub-class relation between Wiki pages under the namespace ( “Category” ), and class and instance relation between Wiki pages under the namespace ( “Category” ) and common Wiki pages. However, MediaWiki in its original form does not allow users to create relation between pages flexibly. Our research has enabled MediaWiki to write labeled links [2]. Wiki syntax to write the labeled link is [[term:target_page|property]]. The labeled link relations will be displayed on the Wiki pages as follows.
1. On the source_page: -> property -> target_page
2. On the target_page: <- property <- source_page
3. On the property: source_page -> target_page

Fig.1 shows an example of the writing of labeled link on a Wiki page. Fig.2 shows how the labeled link relations will be displayed on the Wiki pages.


Editing KASM

```
[[term:Takeda|member]]
[[term:Muljadi|member]]
```

**Fig. 1.** Writing the labeled links on a Wiki page


KASM

->member->Muljadi
->member->Takeda

Muljadi

<-member<-KASM

member

KASM->Takeda
KASM->Muljadi

(a) on a source_page (b) on a target_page (c) on a property page

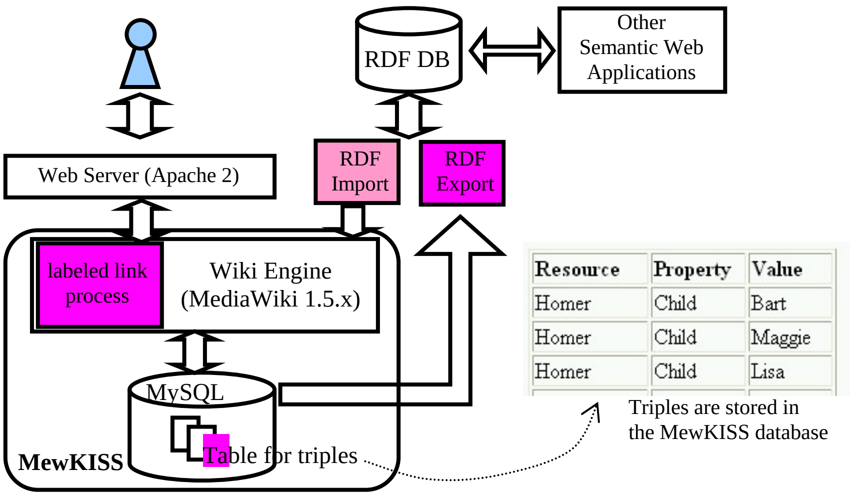
**Fig. 2.** Displaying the labeled link relations on Wiki pages

### 2.2 Features of the Proposed MewKISS

MewKISS is an abbreviation for MediaWiki with Simple Semantics. The word “KISS” is written with full capital letters to stress that the proposed Semantic Wiki is developed by considering the idea of the KISS principle[3]. MewKISS emphasizes the user-friendliness of the Semantic Wiki engine. It is developed to allow non-technical users to write and edit metadata according to RDF statements easily, and leaves the more technical aspects to external applications.

Fig. 3 shows the overall structure of the MewKISS[4]. The features of the MewKISS can be summarized as follows.

1. A collaborative lightweight metadata management. Enabling MediaWiki to write labeled links with simple syntax allows users to create and manage relations between Wiki pages easily and flexibly. The writing of labeled links allows users to write and edit RDF triples even though the users have no knowledge about it.
2. Navigation support. Displaying labeled links allows users to navigate the relation between Wiki pages visually.
3. Mapping to other Semantic Web application. MewKISS handles only simple RDF statements. By converting the RDF triples, which are stored in the new table of MewKISS, into XML-encoded RDF data format, the RDF triples can be exported to RDF database such as Sesame[5].
4. An integrated content and metadata management. By extending MediaWiki, the Semantic Wiki has the benefit of having all the functions available in MediaWiki as a collaborative content management system. Thus, the Semantic Wiki can be used as a collaborative and integrated content and metadata management system.


RDF DB
Other Semantic Web Applications
Web Server (Apache 2)
RDF Import
RDF Export
labeled link process
Wiki Engine (MediaWiki 1.5.x)
MySQL
Table for triples
MewKISS
Triples are stored in the MewKISS database

| Resource | Property | Value |
|---|---|---|
| Homer | Child | Bart |
| Homer | Child | Maggie |
| Homer | Child | Lisa |

**Fig. 3.** Overall structure of the proposed Semantic Wiki

[3] http://en.wikipedia.org/wiki/KISS_principle
[4] Our current prototype is accessible from http://semanticwiki.jp/
[5] http://openrdf.org/

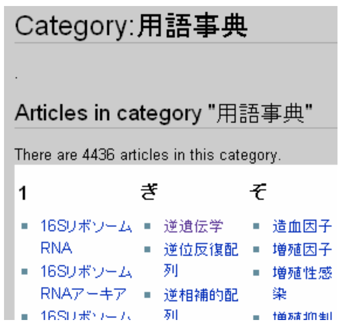
## 3 Semantic Wiki-Based Japanese Biodictionary

### 3.1 Current State of the Semantic Wiki-Based Japanese Biodictionary

Currently the prototype system contains more than 4,000 terms. There are 5 RDF properties used in the biodictionary: is-a, part-of, synonym, English and English synonym. The first two properties, is-a and part-of, are used to represent class and sub-class relations. Synonym is used to link a Japanese biology term with its synonym Japanese term. English is used to link a term with its direct translation's English term, while English synonym is used to link a term with its English term's synonym. Using our proposed Semantic Wiki, other properties, such as kind-of, may also be easily used.

Category management function is also used for categorization and also to list all the terms in the dictionary.

Fig. 4(a) shows a category page that is used to list all the terms in the biodictionary. Fig. 4(b) shows the editing page of a term. Fig.4(c) shows the Wiki page of a term.



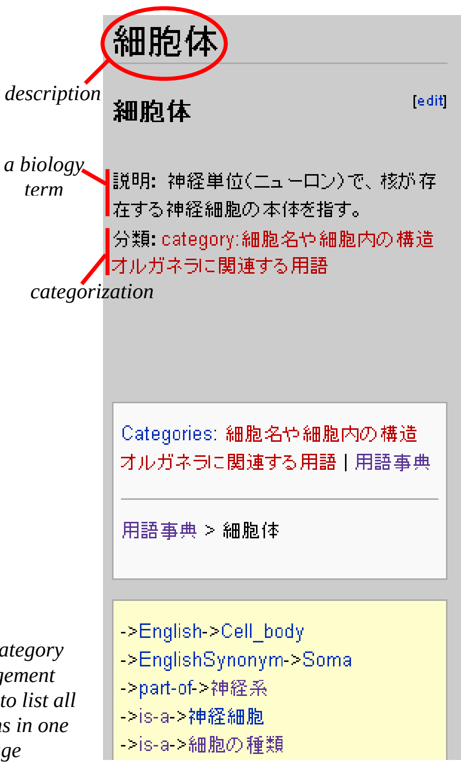
Category:用語事典

Articles in category "用語事典"

There are 4436 articles in this category.

| 1 | ぎ | そ |
|---|---|---|
| 16SリボソームRNA | 逆遺伝学 | 造血因子 |
| 16SリボソームRNAアーキア | 逆位反復配列 | 増殖因子 |
| 16Sリボソーム | 逆相補的配列 | 増殖性感染 |
| | | 増殖抑制 |

(a) A category page to list all the created biology terms



細胞体

[edit]

細胞体

説明: 神経単位(ニューロン)で、核が存在する神経細胞の本体を指す。

分類: category:細胞名や細胞内の構造オルガネラに関連する用語

Categories: 細胞名や細胞内の構造オルガネラに関連する用語 | 用語事典

用語事典 > 細胞体

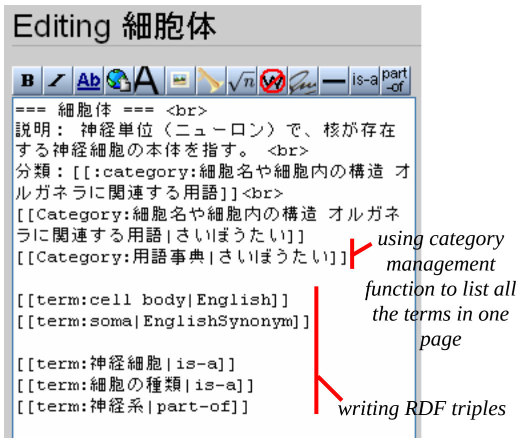
->English->Cell_body

->EnglishSynonym->Soma

->part-of->神経系

->is-a->神経細胞

->is-a->細胞の種類



Editing 細胞体

```
=== 細胞体 === <br>
説明： 神経単位（ニューロン）で、核が存在
する神経細胞の本体を指す。 <br>
分類：[[:category:細胞名や細胞内の構造 オ
ルガネラに関連する用語]]<br>
[[Category:細胞名や細胞内の構造 オルガネ
ラに関連する用語|さいぼうたい]]
[[Category:用語事典|さいぼうたい]]

[[term:cell body|English]]
[[term:soma|EnglishSynonym]]

[[term:神経細胞|is-a]]
[[term:細胞の種類|is-a]]
[[term:神経系|part-of]]
```

B
Ab
is-a
part-of
using category management function to list all the terms in one page
writing RDF triples

(b) Editing page of the Wiki page

**Fig. 4.** Wiki pages in the Japanese Biodictionary

### 3.2 Future Works

The current Semantic Wiki-based Japanese Biodictionary allows users to collaborately create and edit biology terms as well as their relations to other terms. By converting the stored RDF triples into XML-encoded RDF data format, RDF triples can be exported to Sesame. However, further work needs to be done to enable users to export the RDF triples to Sesame directly from the MewKISS environment. Further works also need to be done to allow simple and complex queries directly from the MewKISS environment, and on how the semantic mapping to other Semantic Web applications may benefit the biodictionary.

## 4 Conclusion

For the development of a Semantic Wiki-based Japanese Biodictionary, MediaWiki is extended to enable the writing of labeled links that represent RDF triples. MewKISS provides a collaborative, easy-to-use and integrated content and metadata management system. In the MewKISS environment, users can write and edit RDF triples, even though they have no knowledge about it. RDF triples stored in the table of MewKISS can be converted into XML-encoded RDF data format and exported to RDF database such as Sesame. Thus, MewKISS may well serve as a bridge between non-technical users and Semantic Web technology.

Developing the Japanese Biodictionary within a Semantic Wiki environment does allow experts in various biology fields to create and manage content of the dictionary as well as the relation between terms easily and visually.

## References

1. Kobayashi, S., Kawamoto, S., Mizuta, Y., Demiya, M.S., Muljadi, H., Suzuki, S., Abe, T., Araki, J., Shirai, Y., Ito, T., Kondo, T., Kitamoto, A., Miyazaki, S., Gojobori, T., Sugawara, H., Takeda, H., Fujiyama, A.: The New Generation Bioportal: the Development of a Web Site for Biology Education. In Proc. of the 80th Domestic Conf. of the Society of Biological Sciences Education of Japan (2006) 27 (in Japanese)
2. Muljadi, H., Takeda, H.: Semantic Wiki as an Integrated Content and Metadata Management System. In Poster & Demonstration Proc. of the 4th Intl. Semantic Web Conf. (2005) PID 44

# Ylvi - Multimedia-izing the Semantic Wiki

Niko Popitsch[1], Bernhard Schandl[2], Arash Amiri[1],
Stefan Leitich[2], and Wolfgang Jochum[2]

[1] Research Studio Digital Memory Engineering, Vienna, Austria**
{niko.popitsch,arash.amiri}@researchstudio.at
[2] University of Vienna, Department of Distributed and Multimedia Systems
{bernhard.schandl,stefan.leitich,wolfgang.jochum}@univie.ac.at

**Abstract.** Semantic and semi-structured wiki implementations, which extend traditional, purely string-based wikis by adding machine-processable metadata, suffer from a lack of support for media management. Currently, it is difficult to maintain semantically rich metadata for both wiki pages and associated media assets; media management functionalities are cumbersome or missing. With Ylvi, a semantic wiki based on the METIS multimedia framework, we combine the advantages of structured, type-/attribute-based media management and the open, relatively unstructured wiki approach. By representing wiki pages as METIS objects, we can apply sophisticated media management features to the wiki domain and provide an extensible, multimedia-enabled semantic wiki.

## 1 Introduction

Wikis facilitate simple, efficient, collaborative document creation and evolution. Based on our experience, we believe that wikis are a promising approach and will spread out to new application fields, such as corporate intranets, collaborative knowledge management systems, and e-learning scenarios. However, while traditional wikis (for example MediaWiki[3] or MoinMoinWiki[4]) are unable to semantically structure information, current developments in the field of semantically-enabled wikis [5, 1, 3] suffer from unsatisfactory support for the management of multimedia data, like videos, audio, or complex media presentations.

In this paper, we present Ylvi, a wiki implementation based on the METIS media management framework[2]. Ylvi combines the collaborative properties of traditional wiki systems with strong semantic features that characterize a structured media management framework to implement typed articles, semantic annotations, typed links, and advanced query processing.

## 2 Semantic Features in Ylvi

Ylvi makes extensive use of the METIS framework as its underlying media object management layer. METIS is a middleware component for the rapid development

** This work was supported by the Austrian Federal Ministry of Economics and Labour.
[3] MediaWiki: http://www.mediawiki.org
[4] MoinMoinWiki: http://moinmoin.wikiwikiweb.de/

of multimedia applications with a focus on metadata processing. Its data model is comparable to RDF/RDF Schema and can be extended by complex data types (dynamically loaded Java classes) that are essential for the development of advanced multimedia applications. Media objects can be typed, thereby inheriting strongly typed attributes that can be used to describe the media. Semantic models (ontologies) can be defined using an available Protégé[4] interface. METIS provides a plug-in framework for extending its core functionalities and semantics, including plug-in media-locators, data types, functions and predicates as well as sub-data models (so-called *semantic packs*, e.g. for meta data standards like Dublin Core). A query engine can be used to search along multiple dimensions (metadata values, semantic types, media features), and the Apache Lucene[5] full-text engine was incorporated for indexing text-based content. METIS implements its own multi-channel publishing strategy—basically a complex pipeline of XSLT transformations—for media aggregation.

Ylvi treats both wiki articles and multimedia objects in a uniform way: Both are modeled as METIS media objects that can be typed and attributed and may participate in directed, typed links. An overview of the semantic features provided by Ylvi is depicted in Figure 1 and described in more detail below.

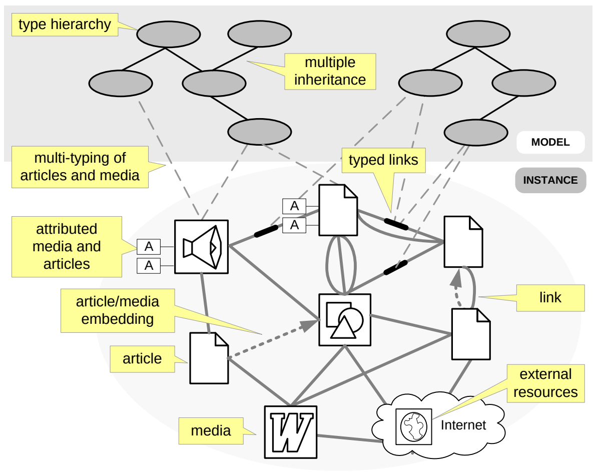


**Fig. 1.** Features of the Ylvi Semantic Wiki

*Type Hierarchy* – Ylvi is able to use ontologies (e.g. formulated in OWL) for the typing of articles and links. These can be imported by using the abovementioned Protégé interface.

*Multi-Typing* – In Ylvi, each article/media object can be an instance of an arbitrary number of types that are defined in the ontology.

[5] Apache Lucene: `http://lucene.apache.org`

*Attributed Media and Articles* – Article types in Ylvi are associated with a set of strongly typed attributes that can be defined using an expressive model (e.g. cardinality restrictions, default values, derived attributes). Each article/media object that is an instance of a type may define values for the respective types' attributes (e.g. picture dimensions, e-mail address, document author).

*Typed Links* – Traditional wikis support simple, purely navigational, unidirectional linking of wiki pages, and minimal inclusion of media objects (mostly images) into wiki pages. Ylvi allows the definition of multi-typed links between articles and media objects; consequently, Ylvi also supports the embedding of media objects or other articles in Ylvi articles.

In contrast to other semantic wikis, Ylvi does not only relate articles as a whole, but also retains the exact position of a link within the source code of an article. Multiple links between the same article pair are not collapsed, but are kept as multiple navigational and logical connections. This allows for example the enrichment of query results with excerpts from the articles in the result set, or ordering of result sets based on the links' textual context.

*Typed Links to External Resources* – As Ylvi does not distinguish between internal and external links, external resources (e.g. other wikis or web resources) can be integrated by using the same syntax, and these links can be typed and queried as well.

*Sophisticated Synonym Handling* – Traditional wikis use the page name as a unique identifier within the scope of one wiki instance. In this case, the wiki must rely on the manual definition of *disambiguation pages*[6], which are not machine-processable. Many semantic wikis, like IkeWiki[5], approach this problem by using an URI as an article identifier. However, this imposes two drawbacks: (*1*) in most cases, URIs are not intended for human consumption and are hard to read and to remember, and (*2*) the wiki has no mechanism to automatically detect ambiguous pages. In Ylvi, we use both a non-unique page name and an internal, auto-generated, unchanging page number as identifier. Users may link to a page using the page name (then, Ylvi automatically creates a disambiguation page), or may eliminate the ambiguity and link to a page using its internal page number.

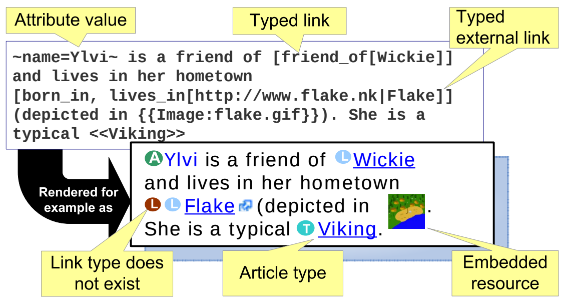
*Search* – Ylvi implements a hybrid semantic search, enabling queries for articles and media objects along multiple semantic dimensions (full-text, types, attributes, links).

## 3 Enriching a Wiki User Interface with Semantic Features

*Markup* – Wiki content is usually expressed in a simple markup language that is easily adopted by non-technical users. Unfortunately, so far no standardized wiki

[6] see e.g. `http://en.wikipedia.org/wiki/Wikipedia:Disambiguation`

markup language exists[7]. As it was our intention to develop an open and flexible system, Ylvi does not implement a particular markup language but rather provides the possibility to configure rendering pipelines consisting of plug-in components that convert markup elements into rendering directives for the chosen output channel (e.g. HTML or LaTeX). The single exception to this is the markup for link definition and semantic annotations. For the definition of (typed) links, Ylvi uses a MediaWiki-like syntax; for typing and attribute intstantiation, two new syntax elements are introduced (see Fig. 2).



```
~name=Ylvi~ is a friend of [friend_of[Wickie]]
and lives in her hometown
[born_in, lives_in[http://www.flake.nk|Flake]]
(depicted in {{Image:flake.gif}}). She is a
typical <<Viking>>
```

Ylvi is a friend of Wickie
and lives in her hometown
Flake (depicted in .
She is a typical Viking.

**Fig. 2.** Ylvi syntax and rendering example

Semantic annotations of Ylvi articles are expressed by using the shown markup elements (and some variations) – no special user interface is required for this. This has several advantages:

- A single, coherent input paradigm is used for content and annotations.
- Semantic annotations are part of an article's content and therefore benefit from all functionality applicable to text-based content in a wiki (versioning, diff, merging, quick copy/paste, . . . ).
- Semantic annotations remain in the article's source code even if the corresponding model elements (e.g. types, attributes) are removed. This makes sense as these annotations may still represent useful metadata of the article and may be automatically reused if the corresponding elements are added to the system again.

*Rendering* – To display an Ylvi article, its source code is passed through the rendering pipeline. The plug-ins interpret the markup (including the semantic annotations) of the article, transform it to a suitable output format (e.g. HTML) and enrich it with additional information that is relevant to the user.

[7] although there are already ongoing initiatives, see `http://tikiwiki.org/tiki-index.php?page=RFCWiki` and `http://www.usemod.com/cgi-bin/mb.pl?WikiMarkupStandard`

For instance, one may introduce a plugin that converts a designated markup of GPS coordinates into a list of articles annotated with nearby coordinates.

*Ontology Manipulation* – As described above, we define wiki markup only for semantic annotation, not for the definition and manipulation of semantic concepts (i.e. *ontology editing*). We do this because ontology development is a non-trivial problem and requires sophisticated user interfaces and tool support. Although Ylvi provides simple online ontology editing features (e.g. adding a new type to the ontology), we heavily rely on the available METIS extension for Protégé, which allows the user to transfer Protégé models into METIS, and makes the structures defined therein available for Ylvi.

## 4 Conclusion

In this paper, we have presented Ylvi, a semantic wiki that has extended functionality for media management. We described how we have merged the traditional wiki approach of collaborative content creation, the extended functionalities of semantic wikis (typed links, typed articles, attributes), and the power of a sophisticated media management framework. We demonstrated how we allow the user to integrate semantic markup directly into a wiki page and how we can use these context-conserving annotations to improve search results. With Ylvi, we have realized a wiki implementation that combines elaborate semantic features with an open, extensible architecture.

In the future, we intend to extend Ylvi's functionality by introducing user management and more accurate markup-based annotation and querying features, and we will extend Ylvi's semantic features by allowing links to be attributed, which supports the expression of more meaningful relations between articles. We consider Ylvi as a suitable framework for the creation of semantic, media-centric (intranet) applications and will continue to develop it into a solid knowledge management and exchange platform.

## References

1. D. Aumueller. Towards a semantic wiki experience - desktop integration and interactivity in WikSAR. In *Proceedings of the ISWC 2005 Workshop on The Semantic Desktop.* Galway, Ireland, November 6 2005.
2. R. King, N. Popitsch, and G.-U. Westermann. METIS - A Flexible Database Foundation for the Unified Management of Multimedia Content. In *Proceedings of the 10th International Workshop on Multimedia Information Systems (MIS 2004)*, 2004.
3. M. Krötzsch, D. Vrandecic, and M. Völkel. Wikipedia and the semantic web - the missing links. In *Proceedings of Wikimania 2005.* Wikimedia Foundation, July 2005.
4. N.F. Noy, M. Sintek, S. Decker, et al. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 16(2), 2001.
5. S. Schaffert, A. Gruber, and R. Westenthaler. A Semantic Wiki for Collaborative Knowledge Formation. *Semantics 2005, Vienna, Austria*, 2005.

# Automatic Deployment of Semantic Wikis: a Prototype

Angelo Di Iorio[1], Marco Fabbri[1], Valentina Presutti[1], Fabio Vitali[1]

[1] Department of Computer Science, University of Bologna, Italy
{diiorio, mfabbri, presutti,vitali}@cs.unibo.it

**Abstract.** Semantic wikis simplify the creation, searching and management of content in a specific domain of interest. Although very powerful solutions exist for adding semantics to wikis, the authoring process of domain-oriented content still remains a manual and quite consuming task. We propose a different approach to deploy semantic wikis: the automatic delivery of a customized wiki for a given domain, taking as input an ontological description of that domain. WikiFactory is an application that takes these ideas to implementation, based on a strong distinction between the ontology designer, the content author and the graphic designer. Moreover WikiFactory is designed to be independent for a specific wiki clone and commit an abstract description of pages onto a wide set of wiki platforms. In this paper we present the early implementation of WikiFactory that automatically generates pages for MediaWiki.

## 1. Introduction

Wikis[2] are increasingly gaining importance among the web authoring tools, either as personal web sites, or as well-featured information systems supporting schools, universities and firms[16]. Among wiki clones, a leading role is played by semantic wikis, which combine the success of the wiki model, with the power of Semantic Web technologies. A Semantic Wiki[17] is a wiki enhanced in order to encode more knowledge than just structured text and hyperlinks, and to make that knowledge readable by machines too. They make it easy to manage, search and retrieve information among the wiki pages and entities. Several examples of semantic wikis can be cited: RDFWiki[12] provides users with a simple text-based interface to edit content and metadata and stores all data as RDF statements; SemanticMediaWiki[10] is an extension of MediaWiki (the wiki platform used by the WikiPedia community[18]) that allows users to add metadata understandable by automatic processes too; Rhyzome[14] allows users to express RDF statements through a simplified syntax called ZML, and many other projects were proposed by researchers in order to merge wikis with semantic web technologies.

A different point of contact between these research efforts can be also figured out: using semantic information in order to generate wikis, apart from annotating them. In particular, we propose to generate content for a wiki, taking in input an ontological description of the domain where that wiki will be used. Each domain, in fact, suggests a natural structure of a related wiki, describing clusters of pages, navigation paths but also templates for each page, or dynamic behaviors and so on. For instance, a wiki for a university is supposed to have pages for courses, classes, professors, rooms, events, exams, and so on; each page is expected to express some information organized according to a given template. Moreover, a lot of repeated pages, repeated structures and repeated templates can be found. What usually happens is that a university employee writes the content of those pages one by one, filling them with the proper data, through an error-prone and time-consuming process.

In [5] we describe WikiFactory, a framework designed for the automatic generation of wikis from ontological descriptions. WikiFactory centres on an OWL description of a domain, written by different users with different skills and processed by an engine that translates such description into actual wiki pages. A first advantage of such approach is clear: it makes automatic, fast and easy the manual process described so far. But another aspect is equally important: WikiFactory does not produce only *wikis* but even a sort of *semantic wikis*. All pages are natively decorated with metadata, directly derivable from the input ontology: relations among entities in the domain can be easily mapped in relations among the wiki pages, as well as objects' properties can be transformed in metadata about those pages. The current implementation of the system does not store metadata yet, but it will be simple to include them into the final wiki, moving off the pool of semantic data behind the generation process.

More details about WikiFactory can be found in [5], where we discussed the rationale behind the system, the overall architecture and the goal of our model. In this paper we present a very early prototype of WikiFactory and discuss how the abstract model has been instantiated and implemented in a running application. The rest of the paper is structured as follows: section 2 gives a brief overview of the WikiFactory publishing model; section 3 illustrates how the current prototype works through a simple case study; section 4 discusses the internal architecture of the system and the conclusions depict possible evolutions of the overall project.

## 2. WikiFactory: a prototype for semantic wikis generation

WikiFactory prototype is a Java application aiming at demonstrating the feasibility and the potential of the WikiFactory's model. Although WikiFactory is designed to generate content for different wiki clones, the current prototype works on MediaWiki[11] and generates pages for that specific clone only. It is a very first implementation of a more complex architecture, which cover many issues about domain-oriented and reliable wikis.

The publishing model behind WikiFactory changes the classic wiki publishing approach since the creation of pages becomes an automatic effect of describing a domain, rather than a direct authoring process. The lifecycle of a common wiki is quite simple and straightforward: an administrator sets up a wiki software and its dependencies (as a web server or an external database), and later many users add content manually by creating and editing topics. Semantic wikis add a new dimension to this workflow, since users can also add metadata to the pages, during the editing phase.

The simplicity and speeding of such approach gave a great contribution to the intensive and widespread diffusion of wikis among Internet communities, companies and organizations. On the other hand, it is still limited in terms of automation, since most of the authoring work still remains completely manual. Fig. 1 shows such a simple scenario:

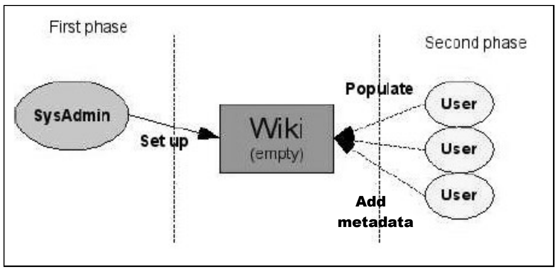


**Fig. 1.** The wiki publishing model

WikiFactory adds an intermediate phase to this process in order to automate the production of repeated pages and structures, by exploiting ontologies. Such improved version of the wiki publishing model is depicted in fig. 2.

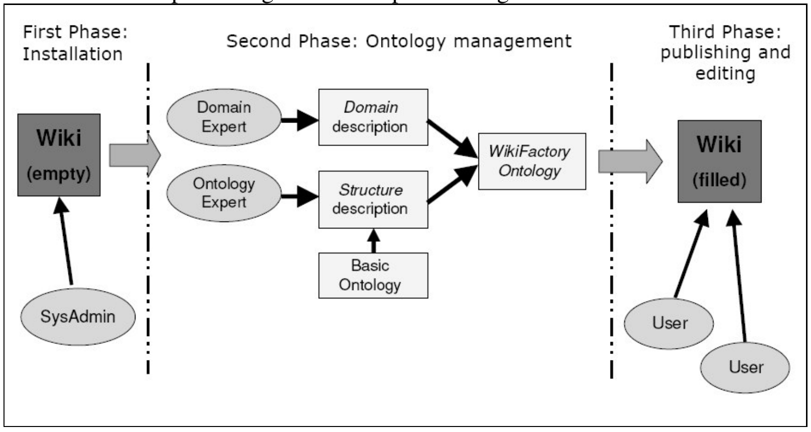


**Fig. 2.** The WikiFactory publishing model

The installation and editing steps do not change (although the automatic installation of the wiki environment is a requirement – with no high priority - of WikiFactory), while a new intermediary step shows users writing (or importing) an ontology and WikiFactory populating the final wiki with content extracted from this ontology. The core of the application is just the ontological description we name *WikiFactory (based) Ontology*, that describes everything needed by the WikiFactory engine in order to populate the final wiki.

Three main sub-components can be identified within such ontology:

- *WikiFactory Basic*: a core ontology supplying rules, constructs and objects used by the designers to define the rest of the ontology. Note that it is different from the whole assembled ontology, whose name is quite similar.
- *Domain Description*: a representation of the domain describing the entities and relations of the domain, and the data to be filled in the wiki.
- *Structure Description*: the actual description of the domain-oriented wiki, a model of topics' structures and templates.

The assembled ontology indicates how to populate the final wiki, by inserting the data provided by a domain-expert, and modelled by an ontology-expert. Two more actors, in fact, exist in that scenario:

- a *domain expert*, we name Bianca, i.e., an inexperienced user who adds content and uses the final wiki every day for carrying out her tasks;
- an *ontology expert*, we name Andrea, who adapts the requirements of the domain experts and actually produce the final ontology.

In order to explain the role played by these two experts and to discuss the structure of the ontology with more details, as well as the internal functioning of the system, we present a case study in the following section.

## 3. WikiFactory workflow: a simple case study

Consider a wiki used by a Computer Science Department of a University (CSD), say the University of Bologna. Such a wiki (CSD wiki) is supposed to have pages about professors, courses, classrooms, staff and so on. WikiFactory prototype is an early application, not yet mature to produce the whole CSD wiki, but it already provides the most relevant constructs in a flexible and extensible framework.

In the rest of the section we describe how the WikiFactory prototype can be used to produce a CSD wiki simply composed by a department home page linked to the list of the affiliated professors and, for each of them, a home page with some information and a list of courses he/she teaches. This example allows us to discuss two relevant goal of the application: (i) describing and committing a set of pages linked each other and (ii) describing and producing a single wiki page. As discussed in [5], the whole design is completed by an automatic deployment of scripts that support dynamic behavior, but that feature is not still supported in the current prototype.

### 3.1. Setting up a wiki target platform

According to the above mentioned schema, the first step consists of a common installation of a wiki platform: a system administrator from the University's technical staff receives a request for a wiki, say MediaWiki, to be installed on a department machine. He installs all required components and a new installation of MediaWiki is now available, with no content yet (apart from content already provided in the installer). It has to be noticed that among the requirements of the WikiFactory platform there is also the automatic configuration and installation of the desire environment, i.e. the wiki clone. Nevertheless, such requirement  now has less priority than others.

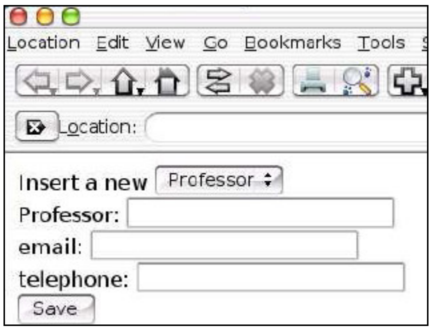
## 3.2. Describing a domain-oriented wiki

The second step of the process consists of producing the *WikiFactory (based) Ontology* collecting all data required to populate the final wiki. As discussed before different actors are involved in creating different parts of such ontology (actually the basic ontology is a built-in feature of WikiFactory).

### 3.2.1. What a domain expert does

Bianca is an employee of the Computer Science Department in charge of supplying information about professors and courses of the department. Such information has to be encoded in RDF statements saying that "a course named 'Web Technologies' exists and it's taught by Fabio Vitali" or "Fabio Vitali is a professor", or "You can contact Fabio Vitali by email at fabio@cs.unibo.it or by phone at 0512094872".

Obviously we cannot expect that Bianca fills manually such RDF document, but we need an automatic process that produces such document. Currently WikiFactory provides her a very simple interface shown in fig. 3. The study of the Graphical User Interface (GUI) is a sensitive aspect, an important requirement of WikiFactory. Nevertheless, its definition has not been approached deeply and comprehensively yet because it depends on the definition of all functionalities WikiFactory is intended to support.



Insert a new Professor
Professor:
email:
telephone:

**Fig. 3.** The WikiFactory Interface for Bianca

The relevant aspect is that Bianca perceives such task as a raw insertion of data, but she is actually populating the ontology (i.e, completing the *domain description*). A clarification is needed at this point: even such description could be further divided in two sub-components, the *generic description* of a domain, and *specific description* of an instance of that domain. Consider the CSD case study: a generic description says that a computer science department has a set of professors and each of them can be described by a set of personal information, including a list of courses he teaches;  a specific description says that the computer science of the University of Bologna has a specific list of professors, including Fabio Vitali, that teaches “Web Technologies” and can be reached by a specific email address or phone number.

One of main important activities we are planning for WikiFactory is just studying the automatic production of the above mentioned interface, from a generic domain description: the interface shown in fig. 3 has been created manually, but it could have been automatically created from a pre-existing ontology about the university domain.

#### 3.2.2. What an ontology expert does

Andrea is an ontology expert in charge of producing the *Structure Description* of the CSD wiki. He works with specific tools such as Protégé[13] and actually writes RDF statements listing the pages of the wiki, describing their connections with the domain entities and their internal structure. Another requirement of WikiFactory is interoperability with Protégé.

The main element provided by the WikiFactory Basic Ontology is called *TreeOf-Topic*. A TreeOfTopic represents a set of wiki pages (or a single one) and, as expected, will be instantiated into corresponding page(s) in the final wiki. In turn, a TreeOfTopic is composed by one or more optional *Iterator(s)* and a *TopicTemplate*.

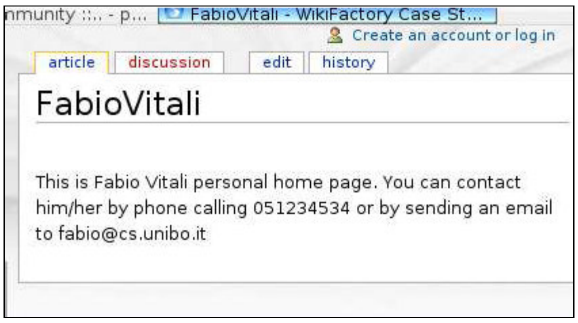
The element Iterator is a very general purpose structure indicating a class of individuals within the domain description: it can be used to tell WikiFactory to generate a page for every instance of that class, and to link that page to the current one. Andrea creates an instance of TreeOfTopic named Professors, containing a Professor Iterator; this Iterator has a specific property pointing to the instances of professors within the domain description. At the end of the process, for each professor included in the list filled by Bianca, a new link will be added to the Professors page.

The element *TopicTemplate* is used to declare the fragments composing a page. The basic assumption is that a small set of components can be identified, able to capture the internal structure of any wiki page, regardless of the wiki platform or the subject of the page. In [4] few patterns are identified able to express the content of any web page too: (classified) paragraphs, headings, tables, records and few other things. What Andrea does is simply describing which components are included in a given page and which text each component is made of. At the end of the process, these elements will be re-flowed and formatted according to the layout of the final wiki. Actually, the current prototype of WikiFactory allows Andrea to specify only the title and the whole body of a page, but we are working on more sophisticated templating languages and solutions.

A very simple rule is used within the WikiFactory prototype: each instance of a word preceded by the character ‘$’ will be automatically replaced by the value of the matching property defined in the Domain Description. Then, the following text fragment “This is *$resource_title* personal home page. You can contact him/her by phone calling *$telephone_number* or by sending an email to *$email*” will be filled with the right data, calculated by the Iterator, and previously inserted by Bianca. Obviously, such a simple templating language cannot be enough for all the real-world scenarios (consider for instance multiple phone numbers or emails), so that we are investigating more complex solutions: particularly interesting are some Java templating engines that could be easily integrated in WikiFactory, such as FreeMarker[6] or Velocity [1].

### 3.3. Instantiating a wiki

The final step of the process shows the WikiFactory engine translating the ontology created by Bianca and Andrea into an actual wiki. According to the semantics of the Iterators and Templating operators, the engine collects all data inserted by Bianca and put them into wiki pages. Fig. 4 shows how the information about a professor are presented in MediaWiki.


nmunity ::.. - p...
FabioVitali - WikiFactory Case St...
Create an account or log in
article
discussion
edit
history

FabioVitali

This is Fabio Vitali personal home page. You can contact him/her by phone calling 051234534 or by sending an email to fabio@cs.unibo.it

**Fig. 4.** A simple page on MediaWiki created by WikiFactory

## 4. A modular java application

The WikiFactory prototype is a java application composed by different modules that work together in order to deploy content on MediaWiki, taking in input the Wiki-Factory Ontology described so far. Fig 5 summarizes the architecture of the system:

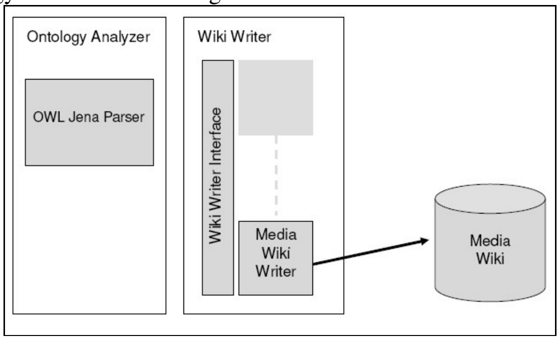


**Fig. 5.** The architecture of WikiFactory Application

Two main components can be identified:

- the *Ontology Analyzer*: that parses the input ontology and collect data about what and how is being published.

- the *Wiki Writer*: that commits the changes reported by the Ontology Analyzer on the target MediaWiki installation.

The Ontology Analyzer includes external libraries to handle OWL documents, in particular the Jena Parser[8], a Java library developed by HP for the access and management of Semantic Web documents. While retrieving information, the analyzer notifies them to the WikiWriter in order to actually produce pages. As expected, the Wiki Writer exports method for: (i) creating wiki-pages according to a given structure (that is, writing the body text indicated in the ontology) and (ii) iterating such creation for all the elements pointed by the Iterator construct. At the end of the process, a commit operation stores information on the wiki clone.

Actually the Wiki Writer is an abstract component, that is a Java interface implemented by every specific wiki clone Writer. The WikiFactory MediaWiki Writer exploits a MediaWiki extension that allows a batch creation of pages, by submitting a text file to a PHP script by Jonathan Cutrer[3]. According to the output of the Analyzer such a file (which even collects many pages together) is written and sent to the MediaWiki target installation through an HTTP connection.

Different solutions can be implemented by different writers, such as writing directly on files, or posting data as a common page editing, or writing a DB or anything else: users can deploy the same wiki from the same ontology on different platforms, since the complexity is hidden within the system. Particularly interesting in this context are the standard Wiki XML-RPC [9] APIs, a set of interfaces based on XML-RPC technologies that allow any client to interact with a wiki, regardless of its internal implementation. We plan to further investigate these libraries in order to standardize the communication and to make WikiFactory completely independent from the target wiki platform.

## 5. Conclusions

The WikiFactory prototype has shown how a simple wiki can be deployed, taking in input an ontological description of its domain of interest. The system relies on a strong distinction between the roles of the users: a domain-expert has to simply insert data, without dealing with their actual formatting into a wiki page, while an ontology-expert has to describe structures, without dealing with the data filling. Most of the routine work is performed by the engine behind the scenes, so that the whole process is simple and automatic.

The prototype presented in this paper is still very much in its initial phases, but we plan many activities towards a complete maturation. First of all, we are working to improve the WikiFactory Ontology and, subsequently, the OntologyAnalyzer: apart from issues about parameterization, configuration and performances, we are working on a more complex set of objects (including tables, records and other fragments) useful to create complex pages, to support more wiki features and to handle dynamic behaviour. As expected, we also plan to code new Wiki Writers, producing content for many other wiki platforms e.g., TWiki[15] or many semantic wiki.

Another important issue raised while working on the prototype is supporting modular deployment. We think that it is a prior and important requirement of WikiFactory to allow users to deploy wiki modules. In fact, it is desirable to add new elements (e.g., structures, templates, topics) to a domain-oriented wiki already deployed without affecting the existing content.

Issues about the interface for non-expert users are being investigated too: ontologies could be also exploited to dynamically create user interfaces (such as web forms, stand-alone applications or anything else), flexible and not hard-coded within the system.

Finally, we plan to focus on a tricky issue never discussed so far: what does it happen on a wiki deployed by WikiFactory after its installation? The current prototype does not face such problem yet, since the editing phase is completely disconnected from the wiki deployment but a lot of interesting issues can be raised by studying the consistency between the ontological view of a wiki and its actual pages, as well as the techniques used to update both of these views. What we want to do is either propagating changes from the ontology into the wiki or, the opposite, updating the ontology according to the modifications on wiki content. This is a very difficult task and we haven't yet found a good solution but we consider it the most important development track of our research.

However, WikiFactory is a lively project, whose preliminary implementation has been described in this paper. More detailed and up-to-date information can be found in its wiki site, at the address http://swe.web.cs.unibo.it/WikiFactory/.

## 6. References

1. Apache Jakarta Project, "The Velocity Template Engine", http://jakarta.apache.org/velocity/.
2. Cunningham, W. & Leuf B.*The Wiki way*. New York: Addison-Wesley, 2001.
3. Cutrer J. "Mediawiki bulkpage Page Creator", http://meta.wikimedia.org/wiki/MediaWiki_Bulk_Page_Creator.
4. Di Iorio A., Gubellini D., Vitali F. "Design patterns for document substructures". In the *Proceedings of Extreme Markup Conference 2005*, August 1-5, 2005, Montreal, Canada.
5. Di Iorio A., Presutti V., Vitali F. "WikiFactory: an ontology-based application to deploy domain-oriented wikis". To appear in the *Proceedings of the European Semantic Web Conference 2006*, June, 2006, Budva, Montenegro.
6. FreeMarker, "FreeMarker templating engine", http://freemarker.sourceforge.net/index.html
7. Guzdial, M. Rick, J. and Kehoe, C.: "Beyond Adoption to Invention: Teacher-Created Collaborative Activities in Higher Education", *Journal of the Learning Sciences*, 2001, Vol. 10, No. 3, 265-279.
8. HP Labs, "Jena – A Semantic Web Framework for Java", http://jena.sourceforge.net/.
9. Jspwiki.org, WikiRPCInterface, http://www.jspwiki.org/Wiki.jsp?page=WikiRPCInterface
10. Krotzch Markus, Denny Vrandecic, and Max Volkel. "Wikipedia and the Semantic Web The Missing Links". In *Proceedings of Wikimania 2005*, Frankfurt, Germany, August 2005.
11. MediaWiki.org, "MediaWiki", http://www.mediawiki.org/wiki/MediaWiki.
12. Palmer Sean B. "RDFwiki". http://infomesh.net/2001/rdfwiki/.

13. Protégé, "The Protégé Ontology Editor and Knowledge Acquisition System". http://protege.stanford.edu.
14. Souzis A. "Rhizome position paper". In Proceedings of the *1st Workshop on Friend of a Friend, Social Networking and the Semantic Web*, September 2004.
15. Thoeny P. TWiki: Enterprise Collaboration Platform. http://twiki.org.
16. Thoeny P., "TWiki Success Stories", http://twiki.org/cgi-bin/view/Main/TWikiSuccessStories.
17. Wikipedia.org, "Semantic Wikis", http://en.wikipedia.org/wiki/Semantic_Wiki.
18. Wikipedia.org. Wikipedia Home Page. http://www.wikipedia.org.

# Bringing the “Wiki-Way” to the Semantic Web with Rhizome

Adam Souzis[1]

[1] Liminal Systems, 4104 24th Street Ste. 422,
San Francisco, CA, USA
asouzis@users.sourceforge.net
http://www.liminalzone.org

**Abstract.** The Wiki and the Semantic Web can be compared as two different approaches to capturing knowledge, where the former trades away precise, explicit, and internally consistent semantics for speed and simplicity. Any attempt to bridge these two approaches has to either somehow reconcile these trades-off or make compromises one way or the other. This paper describes how Rhizome, an open source application framework for developing “Semantic Wiki” applications, attempts to bridge these approaches. Rhizome includes a text formatting language called ZML whose syntax is similar to text formatting languages found in most Wikis but with enhancement to make it easy for users to express explicit and arbitrary semantics. Rhizome relies on “shredding”, a flexible framework for specifying rules for characterizing semi-structured content with RDF and providing an ontology that can precisely describe the relationship between the source content and the resulting statements.

## 1 Background

The Wiki and the Semantic Web can be compared as two different approaches to capturing knowledge, where the former trades away precise, explicit, and internally consistent semantics for speed and simplicity. Any attempt to bridge these two approaches has to either somehow reconcile these trades-off or make compromises one way or the other; for example, by adding complexity and constraints that undermines Wiki design principles or by limiting the scope where Semantic Web data can be applied (e.g., limiting it to meta-data associated with traditional wiki pages).
The Wiki has proven to be a remarkably successful tool capturing knowledge in a collaborative, open fashion. The inventor of the Wiki, Ward Cunningham, has identified several Wiki design principles, which he refers to as the “Wiki-way”[1]. A review of his descriptions of some of these principles is suggestive of how they can be challenging for applications that utilize and create Semantic Web data:
“**Mundane** – a small number of (irregular) text conventions will provide access to the most useful page markup”[2] But this approach doesn't easily lend itself to making precise and controlled statements; indeed Semantic Web scenarios generally assumes a specialized user interface for a particular application domain.
“**Unified** – Page names will be drawn from a flat space.”[2] This principle seems in accord with the use of universally unique URIs as the basis of names for the Semantic

web; however, the scope of this namespace is so huge it is pragmatically difficult to treat as a flat space.
"**Tolerant** – Interpretable (even if undesirable) behavior is preferred to errors."[2] But ontologies and ontologies languages generally require some degree of internal consistency to function properly.
"**Open** – any reader can edit [a page] as they see fit."[2] However, when the content being created is Semantic Web data which can be readily consumed by -- and alter the behavior of – applications, security concerns must be addressed.
This paper attempts to conform to the ABCDE format for Semantic Conference Proceedings[3]; the next section, "Contribution" describes how Rhizome[4], an open source application framework that makes it easy to develop "Semantic Wiki" applications, contributes to the challenges outlined above; this is followed by the Discussion section which describes Rhizome's architecture in more depth.

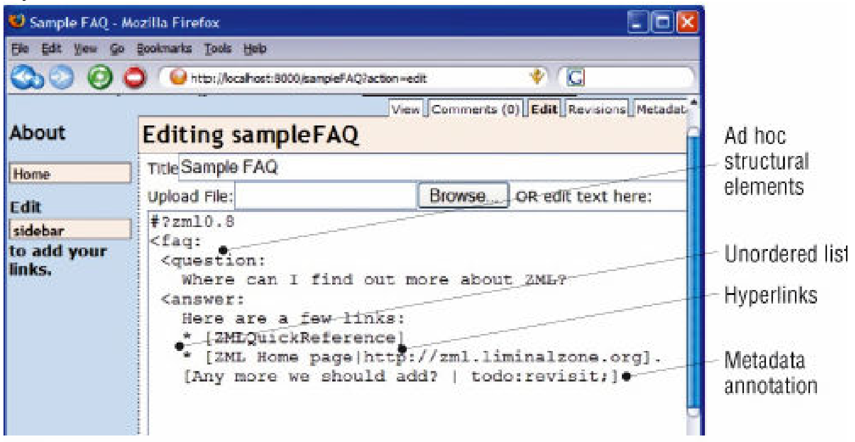
## 2. Contribution

Rhizome is an open source application framework that makes it easy to develop "Semantic Wiki" applications: applications that can create and utilize RDF data and Semantic Web ontologies while letting users interact with and modify that data in a Wiki-like fashion. In this section we describe how Rhizome attempts to fulfill the Wiki design principles discussed above.

### 2.1 Mundane

What sort of "(irregular) text conventions" should be used for authoring RDF triples?
The simplest approach would be a text format limited to providing a way to explicitly describe RDF triples. And arguably, existing plain text RDF formats such as N3 and Turtles already fit this criteria. However, this approach limits its audience to those with knowledge of RDF and domain-specific ontologies. And even for sufficiently trained users, writing precise and atomic RDF statements flies in the face of the Wiki's goal of being "quick".
A more ambitious approach would be to design a more traditional Wiki-like text format whose structure could be easily represented as RDF. However there are several challenges to creating a mapping to generic RDF or some general purpose ontology for content. First, current Semantic Web standards, such as OWL, are not yet powerful enough to inference equivalencies between a representation in a content ontology and its appropriate domain-specific ontology. Second, the most intuitive markup structure for a particular application doesn't always submit to a straightforward mapping to RDF. Finally, there's the practical issue that representing structural elements in free form text as RDF creates a tremendous volume of RDF statements, especially if order is preserved.
Because of these limitations, Rhizome's approach is to use a Wiki-like text format (dubbed ZML) that is flexible enough to express arbitrary structure but doesn't specify a particular translation to RDF. Instead, the system determines which translation rules to apply based on the content of the text.

Unlike other Wiki text formats, all structural elements in ZML can be arbitrarily nested (relying on whitespace much like the indentation rules found in the Python programming language) and annotated with attributes. The result of parsing ZML is an XML document and in fact ZML can used as a simple, concise alternative syntax for XML. This design enables the user to easily use microformats[5] or domain-specific XML vocabularies (for example, Rhizome supports vocabularies from the Apache Forrest and Docbook projects). Another advantage is that this lets arbitrary HTML or XML be converted to ZML, enabling round-trip conversions. For example, users can write content in ZML, edit it in a WYSIWYG (X)HTML editor, or process it with specialized tools that consume XML, and then view it as ZML again.
ZML also has syntactic constructions to make it easy to explicitly express semantic distinctions that are elided in other Wiki text formats. For example, we must distinguish between creating a reference to a WikiName (which, in our case, corresponds to a RDF resource name) and creating a hyperlink, which has explicit presentational intent and generally implies a relationship between the content and the link target. Similarly, we must distinguish between anchors and their common use as a way to name document sections.



```
#?zml0.8
<faq:
 <question:
   Where can I find out more about ZML?
 <answer:
   Here are a few links:
   * [ZMLQuickReference]
   * [ZML Home page|http://zml.liminalzone.org].
   [Any more we should add? | todo:revisit;]
```

**Fig. 1.** A screenshot of a page being edited in the Rhizome Wiki, with aspects of ZML syntax highlighted.

ZML doesn't directly translate into RDF; instead it relies on “shredding”, the process Rhizome uses to bridge implicit and explicit semantics. Shredding is a flexible framework for specifying rules for characterizing semi-structured content with RDF and providing an ontology that can precisely describe the relationship between the source content and the resulting statements.
Rhizome lets users create rules that trigger shredding on the basis of the content's type. For example, shredding an RDF/XML document would consist of parsing the RDF; shredding an (X)HTML document could invoke invoking a GRDDL (Gleaning Resource Descriptions from Dialects of Languages) [6] XSLT stylesheet; and shredding an MP3 file would consist of extracting the metadata out of the embedded ID3 tag. Using RxPath's support for RDF named graphs (see below), Rhizome can retain the relationships between an instance of content and statements extracted from

it, enabling it to know, for example, that the statements might be out of date when content has changed.
Rhizome also lets users directly view and edit raw RDF in ZML via RxML, an alternative syntax to RDF with the goal of enabling novices to read and edit RDF using a metaphor conceptually similar to and only incrementally more complicated than application properties file formats such as Microsoft Windows' .ini files. Although RxML can express any set of RDF statements, it presents the RDF in a constrained, simplified manner: as a list of resource URIs, each of which has a set of property name-value pairs.

### 2.2 Unified

Providing a unified namespace for users requires a strategy for mapping WikiNames to RDF resource URIs. One simple approach would be to treat the WikiNames themselves as a resource URI, e.g. by introducing a “wiki:” URL scheme. It is obvious that given the decentralized nature of the Semantic Web this approach could not scale without name conflicts arising. Alternatively, we could generate a unique URL from a WikiName; for example by using the actual URL to the web page that corresponds to the WikiName, or by pre-pending some application specific base URI. However, this contradicts the principle of a unified namespace by essentially creating separate namespaces -- users would not be able use to WikiNames to refer to resources outside the system without some way to refer to those namespaces.
Thus Rhizome assumes that in order to provide a single, flat namespace of WikiNames that is universally addressable we need to create a level of indirection between a RDF resource URI and its WikiName, and accept that the determination of this relationship is dependent on the context it appears in. WikiNames are treated as a property of a resource, with only slightly stronger semantics than RDF Schema’s “rdfs:label” property. When a WikiName is referenced in content, it is up to the shredding process to assert a relation between it and a RDF resource. This is appropriate because the question of how closely that name should be “bound” to an RDF resource is dependent on the needs of the specific application and what assumptions can be made about the context in which it appears.

### 2.3 Tolerant

The principle of tolerance is harder to achieve with Semantic Web data than the plain text found in traditional Wikis because Semantic Web data is precise and machine consumable and so very often requires some degree of validation. Rhizome allows an application to maximize the tolerance allowable by providing partial, incremental and ad-hoc of validation of RDF using Schematron. Thus validation can be accomplished without having to use complex ontology languages such as OWL, which can often break down in the face of inconsistency. Schematron[7] is a validation language that uses XPath expressions as assertions about the validitity of a XML document. Using RxPath (described below), Schematron can be used to validate a RDF model. The benefits of using Schematron to validate XML also apply to validating RDF:

Schematron allows complex, ad-hoc assertions to be expressed that can't easily be expressed in other schema languages. For example, because OWL is based open world model, it can't define constraints that apply against the entire model such as uniqueness or default values. And compared to languages like OWL, Schematron is easier to write and understand and requires much less specialized knowledge.

### 2.4 Open

Like tolerance, it is more difficult to achieve openness in Semantic Web applications than with traditional Wikis. Rhizome attempts to balance openness with security by providing an authorization scheme that is powerful yet unobtrusive (one that doesn't impose an addition work where it is not needed). Rhizome lets the application define authorization rules for the addition and removal of arbitrary RDF statements using the notion of access tokens that guard resources. This conceptually simple model can be used to build fairly complex authorization rules; for example, one that allows a guest account to create a new user account for herself, but not modify or create other accounts or objects. However, the RDF model can make it difficult to create these rules because of the very fine-grained nature of RDF resources (for example, even very simple types objects can require anonymous resource nodes). Rhizome deals with this by allowing the application to declare properties that are used to partition an RDF graph into coarser-grained objects to apply authorization to.[1] Rhizome also maintains a revision history of all changes to the system using named graphs to model transactions. This allows changes to be monitored and inappropriate modifications to be reverted when necessary.

## 3. Discussion

This section provides an overview of Rhizome's architecture.

### 3.1 Architecture

Figure 2 illustrates the overall architecture of the Rhizome framework. Components are arranged as a stack in which higher-level components depend on the lower-level components, but not vice versa. Consider each layer from bottom to top:

[1] Not discussed here is ways in which class inferences add complexity when rules based on class types are allowed.

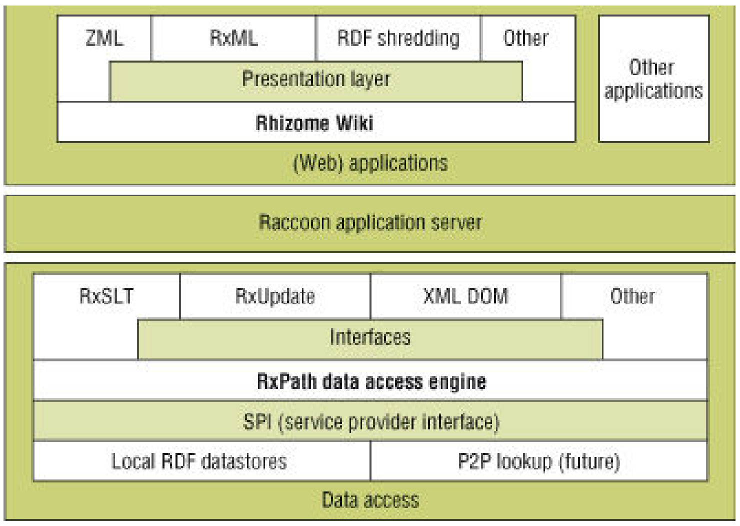


**Fig. 2.** Rhizome's architecture.

### 3.1.1 RxPath data access

RxPath is an RDF data access engine that provides a deterministic mapping between the RDF abstract syntax and the XPath data model. This lets users access RDF data stores as a (virtual) XML DOM (document object model) and query them using RxPath, a language syntactically identical to XPath 1.0. This approach allows the full range of XPath-based languages to be used to query and manipulate RDF models -- for example, XSLT for presentation and transformation, XUpdate[8] for modification, Schematron for validation, and XForms for presentation and modification -- without having to make any syntactical changes to those languages.

RxPath maps the set of (subject, predicate, object) triples in an RDF model into a virtual and possibly infinitely recursive tree in which:

- the root has a child node corresponding to each resource in the model,
- each resource node has child nodes for each statement that it is the subject of
- each statement node has a single child node corresponding to the statement's object.

If the statement's object is a resource, it might in turn have child nodes that correspond to the statements that the resource is subject of, and so on. Given such a tree, an XPath expression such as `/foaf:Document/dc:creator/*` will select a set containing all the authors of each document resource in the RDF model.
RxPath also supports "named graphs"[9] (also known as contexts), a common extension to the RDF model that is used to partition RDF statements into groups. RxPath uses a unique approach to contexts by treating them not as a one-to-one mapping with a subgraph of an RDF model, but as a collection of subgraphs composed through union and difference operators. This enables Rhizome to use

contexts simultaneously and efficiently to model many different concepts, such as metadata versioning, transactions, provenance, application partitioning, and personalization (user customizations). For example, Raccoon's transaction log of changes made to the RDF store is represented as a collection of contexts, each of which adds or subtracts from the previous context. Using contexts lets Rhizome capture when, where, how, and by whom a set of statements was made.

#### 3.1.2 Raccoon application server

Raccoon is a simple application server that uses an RDF model for its data store. Raccoon uses RxPath to translate arbitrary requests — such as HTTP requests or command line arguments — to RDF resources. Each of these can be associated with style sheets in RxSLT and RxUpdate languages, which can generate responses or update the RDF data store.

Raccoon's goal is to present a uniform and purely semantic environment for applications. This enables the creation of applications that are easily migrated and distributed and that are resistant to change. Raccoon is designed primarily for applications that look at the world as a universe of RDF statements, but it also works with XML-centric applications. Raccoon isn't designed to be a full-featured application server and in fact will often be embedded in another application server. Raccoon's job as an application server is a narrow one—to map a request to a response, possibly modifying the state of the application in the process:

Request $\rightarrow$ Application (Rules + Store) $\rightarrow$ Response

A request is a dictionary of simple values, and an application defines a pipeline of RxPath expressions that transform the request into the response. Raccoon presents both the request and the application's state using the RxPath data model. This approach enables the creation of applications that can be transparently distributed and aggressively cached. Application code is always executed within the context of a request. There are external requests, such as HTTP requests, and internal ones, such as the requests sent when an application starts or stops. Raccoon also provides basic transaction coordination for managing updates to the RDF store. Using contexts enables the application to choose an appropriate consistency model for its needs. If full global atomic consistency isn't needed, Raccoon can cache request responses even more aggressively and still provide the appropriate levels of cache coherency.

#### 3.1.3 Rhizome Wiki

Running on top of Raccoon is the actual Wiki application, which offers all the basic functionality found in Wikis, such as letting users create and edit pages on an ad hoc basis; along with some more advanced content management features such as roles and groups, release workflow, and basic facet navigation. Almost all of the Wiki's functionality is implemented in its dynamic pages, which are written in RxSLT, XSLT, and RxUpdate. Users can edit these like any other pages, making it easy to incrementally add and change functionality. They can also use RxUpdate to modify the underlying schema at run-time.This flexibility makes access control very important—to this end, Rhizome uses a flexible schema for authorizing both application-level actions and statement-level changes to the RDF store based on the authorization mechanism described in the previous section.

### 3.2 Conclusion

This paper has examined some of Rhizome's approaches to applying Wiki design principles to the Semantic Web. Despite the challenges of marrying two very different approaches to capturing knowledge, doing so can help reduce the barriers that often hinder the adoption of Semantic Web technologies, such as high learning curves for users, demands for precision and consistency, and the need to develop domain-specific user interfaces.

## References

1. Cunningham, W, Leuf, Bo.: The Wiki Way: Collaboration and Sharing on the Internet Addison-Wesley Professional (2001)
2. Cunningham, W, et. al. http://c2.com/cgi/wiki?WikiDesignPrinciples
3. http://www.dfki.de/~paulb/ABCDEF/ABCDEF.htm
4. Souzis, A: Building a Semantic Wiki. IEEE Intelligent Systems (Sep/Oct 2005) 87-91
5. http://www.microformats.org
6. Hazael-Massieux, D., Connolly D.: Gleaning Resource Descriptions from Dialects of Languages (GRDDL), World Wide Web Consortium (W3C) Note (2005)
7. ISO/IEC 19757-3 Document Schema Definition Languages: Part 3 — Rule-based validation — Schematron (2004)
8. Laux, A., Martin, L.: XUpdate—XML Update Language, XUpdate Working Group Specification (2000).
9. Carroll, J. et al.: Named Graphs, Provenance and Trust, In: Proc. 14th Int'l Conf. World Wide Web (WWW 05), ACM Press, (2005), 613–622.

# Towards a Wiki Interchange Format (WIF)
## Opening Semantic Wiki Content and Metadata

Max Völkel[1] and Eyal Oren[2]

[1] Forschungszentrum Informatik, Karlsruhe, Germany,
voelkel@fzi.de, http://xam.de
[2] DERI Galway, Ireland
eyal.oren@deri.org, http://eyaloren.org

**Abstract.** Wikis are increasingly being used in world-wide, intranet and even in personal settings. Unfortunately, current wikis are data islands: people can read and edit them, but machines can only send around text strings without structure. Wiki migration, publishing from one wiki to another one and free choice of syntax hold back broader wiki usage.
We define a wiki interchange format (WIF) that allows data exchange between wikis and between related tools. Different from other approaches, we also tackle page content and semantic annotations. The linking from formal annotations to parts of a structured text is analysed and described.

## 1 Introduction

Wikis are increasingly being used in world-wide, intranet and even in personal settings. There are over 130 wiki engines listed in the *C2 Wiki*[3]. Google statistics reveal that *MediaWiki*[4] and TWiki are the most popular engines, with rising tendency. MediaWiki has $314 \times 10^9$ hits, probably influenced by its usage on Wikipedia, one of the top 20 most used websites[5] in the world. *TWiki*[6] with $30 \times 10^9$ hits is mostly run in company intranets.

Apart from these two specialised wiki engines, the many other wiki engines have relatively similar popularity[7]: PukiWiki (8,7 Mio), TikiWiki (7,1 Mio), MoinMoin (6 Mio), Atlassian Confluence (4,2 Mio, commercial), PhpWiki (4 Mio), PmWiki (3,9 Mio), Xwiki (3,3 Mio), JspWiki (1,8 Mio), SnipSnap (1,8 Mio), JotSpot (1,5 Mio), UseMod (0,7 Mio), and SocialText (0,7 Mio, commercial).

Unfortunately, the increasing usage of wikis leads to new problems, as many people have to use multiple wikis, e. g. a personal (local) wiki, the company

[3] `http://c2.com/cgi/wiki?WikiEngines`.
[4] `http://www.mediawiki.org`.
[5] according to Alexa.com, April 2006
[6] `http://www.twiki.org`.
[7] Based on Google queries since 2004 up to 28.03.2006. Google hit count can only be a rough indicator of wiki engine popularity

intranet wiki, an external collaboration wiki and the Wikipedia for background knowledge. But there is no interoperability between these wikis: despite their open-ness, current wikis are data islands.

Ideally, one could export some or all pages from one wiki and import them into another wiki, which uses a different wiki syntax and offers a different set of features. But the diversity of different wiki syntaxes makes exchanging structured content quite expensive: a converter has to written for each wiki that is to be connected with other systems.[8]

Instead of writing conversion scripts between each pair of available wiki engines we propose the use of a wiki interchange format. This reduces the implementation costs roughly from $n^2$ to $n$ for $n$ different wiki engines. An interchange format would also allow the independent creation of multiple wiki syntaxes and user interfaces.

### 1.1 Structure of this paper

First we analyse requirements (see Sec. 2). In Sec. 3, we elaborate on what constitutes to the wiki data model and which is the right layer of abstraction for an interchange format. We also discuss interaction with wikis on the web. In Sec. 4, we make a proposal for a wiki interchange format (WIF) and a wiki archive format (WAF). We present the concept of a Wiki Mediation Server to handle runtime aspects of wiki migration. We report on implementation and experiences in Sec. 5 and review some related work in Sec. 6. Finally, in Sec. 7 we conclude and present future work.

## 2 Scenarios and Requirements

The following scenarios are currently not supported by existing wiki engines and would profit from a wiki interchange format (WIF):

**Exchanging wiki content.** Currently, the only way to copy content from one wiki to another one is through copy-and-paste and manual reformatting of the text. This is painful and unnecessary, since most wikis offer similar content formatting and structuring abilities, namely those found in HTML (as most wikis render their content as HTML). The rising popularity of personal wikis adds even more weight to the need of exchanging content between wikis.

**Wiki migration.** Sometimes, the complete content of one wiki should be migrated to another wiki, e. g. because another wiki engine should be used. This implies migrating all pages. Ideally, user accounts and page histories would also be migrated.

**Wiki syntax united.** Existing wikis use various different wiki syntaxes. Ideally, one could use the same favoured syntax on all wikis.

[8] Although some systems provide an HTML-import (e. g. JSPWiki [9] and TWiki), this does not fully solve the problem, as most other wikis can only export pages including navigational aids.

**Wiki archiving.** Creating a browse-able off-line version of a wikis content, e. g. to read on the train or as a backup.

**Wiki synchronising.** One of the big advantages of a wiki is the easy linking ability: users just have to remember the page title to create a link. External links require much more effort. Persons participating in multiple wikis have to remember many logins, navigation schemes and wiki syntaxes. Ideally, on could change a page in wiki *a* and have it automatically changed in wiki *b* as well.

From these scenarios and additional thinking, we can derive a number of requirements for a WIF:

**Round-tripping:** Full data round-tripping is the overall goal. Ideally, one could export all content of a wiki site into WIF and import it back again into another wiki installation without loss of data or structure. E. g.: If we have two wikis $a$ and $b$, where $a$ has a very rich and $b$ a very little page structure model, the transition $a \rightarrow b$ (i) would be useful for users of wiki $b$, even if a lot of structure within the pages would be lost due to the restricted page data model of wiki $b$. The mapping $b \rightarrow a$ (ii) would also in general be beneficial for the users of wiki $a$. The problems arise with mappings like $a \rightarrow b \rightarrow a$ (iii). Now the users of wiki $a$ would be faced with a loss of structure which they used to have before the export and re-import of data. This is a problem that *cannot* be solved in general, due to the lower expressivity of wiki $b$. As the mappings (i) and (ii) are useful in practice for wiki migration, we have to find a format that is expressive enough.

**Easy to implement:** as we need a way to obtain the WIF for each different wiki engine, it is important to keep development costs low.

**Renderable:** WIF files should be renderable in standard browsers.

**Completeness:** If a wiki lists e. g. all pages in the category "employee" using some kind of plugin, a WIF should export this information in an appropriate format. Wikis with a similar powerful plugin system would profit from an interchange format that keeps a reference to the plugin used. Less capable wikis, however, would need the content generated by the plugin, as they do not have the power to generate the content themselves. Users migrating from a wiki with a particular feature to another wiki engine would rather profit from having dynamic content materialised into static content than not having access to that content at all.

**Compactness:** The single-page-WIF needs to encode all structural information of a wiki, e. g. nested lists, headlines, tables, nested paragraphs, emphasised or strongly emphasised words. But it does not need to encode font size or font color.

**Ease of Use:** It should not only be easy to generate single-page-WIF, it should also be easy to work with it.

## 2.1 Experiences in Wiki Migration

Recently, in one of the authors' research groups, the intranet was migrated from SnipSnap wiki to MediaWiki – to use features of the Semantic MediaWiki extension [19]. As WIF was not ready at that time, one person manually converted the contents. As he migrated a wiki before, the basic task was not new to him. He worked on the stored text files and used a 213 lines long bash script, which in turn called 102 lines of *sed* and 29 and 34 lines of *awk* [4].

Reported problems where the unusual SnipSnap syntax - opening and closing tags are the same for plugins. Encoding was also an issue. Comments where migrated to MediaWiki discussion page sections. Finally, a generated XML file is imported using a MediaWiki XML import feature.

The approach took about 20 hours of work, would require significant refactoring before applicable to another target or source wiki engine, depends on having admin rights on the source and target wiki server and relies on specifics of the wiki engines, e. g. the XML import of MediaWiki.

## 3 Analysis and Discussion

In this section we arguet that an interchange format has to exchange data on the level of the wiki data model an not on the wiki syntax level. Additionally, we show which parts of the wiki data model are relevant for a WIF.

One could try to standardise the wiki syntax, as many people suggested on diverse wiki and web pages [1]. A standard markup would indeed be of great benefit for novice users and data migration. But in reality, existing wikis are unlikely to change the established syntax rules. For new wiki engines, the MediaWiki syntax should be considered, as it is likely to be the most widely known syntax. Some new Semantic Wikis such as IkeWiki adopt this approach. On the other hand, innovation in wiki engines and wiki syntaxes is still high: new wiki engines and new wiki features, such as semantic annotations and plugins, need syntax extensions as well.

Therefore, standardising wiki syntax is not a feasible solution; instead we propose to define a standard format for the data model.

As we want to exchange the data structures of a wiki page, we abstract away form the ways these structures were created. E. g. a bulleted list is a structural concept, worth exporting, while the syntax that was used to create this structure (star or minus sign at the beginning of a line) is of less interest for other wikis or tools.

Unfortunately, most wiki engines have no formal data model published. Their data models are defined only by their implementation of wiki syntax and rendering. In order to obtain a *common* wiki data format, one would have to

– formalise existing wiki data models including page graph, page metadata and the structured page content, and
– identify the union of all formalised data models.

We looked into the data models of SnipSnap (used as the intranet of one of the authors' institute), MediaWiki (popular, see Sec. 1) and JspWiki (used by one of the authors as a personal desktop wiki).

### 3.1 Wiki data model

What is in a wiki? Several definitions are possible, we go for a definition from the end-user point of view. The end-user has two ways of interacting with wiki content. She can either *browse* (read-only) the hypertext version or *create and change* content of pages using wiki syntax. It is *the content written by users*, that has to be migrated to other wiki engines. Some important wiki features such as "backlinks" are inferrable form the user-entered content. Arguably, usage data such as "10 most accessed pages" or the "recently changed pages" contribute considerably to the wiki usability. This data cannot be manipulated directly by users through the web interface. So we have to ask ourselves:

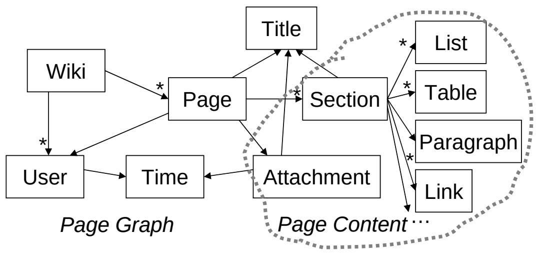


**Table 1.** A high-level view on the wiki data model

What constitutes the data model of a wiki? The content of a wiki is defined by the page graph including page metadata and the actual content of pages. Figure 1 shows some of the most common elements of a wiki data model. Existing exchange formats tackle the page graph part (as XML dialects), but often simply embed the page content part as wiki syntax (e. g. MediaWiki, SnipSnap). This is maybe due the fact that most wiki engines have an internal representation of the page graph but not necessarily a detailed representation of a page given in wiki text. In fact, pages are often rendered by wikis using a cascade of text search and replace commands using carefully crafted regular expressions. To sum up, we distinguish three levels of data for wiki content:

**A wiki page** consisting of:

- **Content structure:** e. g. headlines, nested lists, tables, or other elements that state a visual relation between content items.
- **Content style:** e. g. font size, font color, bold, italic or other visual rendering of content items.
- **Content semantics:** e. g. user authored content, backlinks, lists generated by a plugin or macro, embedding of other wiki pages, or a reference to a variable that display the number of pages in a wiki. The content semantics are invisible to e. g. an HTML processor.

**Metadata about a wiki page.** We have two kinds of metadata (at least in a Semantic Wiki context):

- **Explicit metadata** as stated by semantic annotations or as defined in special user interface elements (i. e. access rights).
- **Application metadata** such as last editor, creator of a page, previous version. This metadata cannot be changed directly by the user, only indirectly through application usage.

**Global wiki data** such as user accounts.

### 3.2 Semantic Wiki data model

Semantic Wikis such as SemperWiki [15, 14] add the notion of formal annotations to wikis, and we therefore extend the data model with annotations. We model an annotation consisting of three parts [16]:

1. A real world artefact that is annotated, such as a person, a book, or a web resource.
2. A formal identifier for the real-world artefact, such as an ISBN number or a URI [2].
   The link from identifier to real-world artefact is outside the expressiveness of formal systems. In RDF, we face an additional problem: URIs are used both as concept identifiers and as locators (URL) of resources on the WWW. Given the URI of a web resource, what is the URI of the concept described on that web page?
   In WIF, we solve the so-called "URI crisis" by using *mirror-URIs* [16]: for each web-locatable URI we construct a non-locatable URI by prefixing it with the URN-scheme 'concept' and URL-encoding characters as necessary. Such a mirror URI describes "the primary concept mentioned in the human-readable representation of the resource *a*". E. g. `http://w3.org` represents the web page of the W3 consortium, while `urn:concept:http://w3.org` could denote the consortium itself. The exact meaning of the URN is up to a social process outside the scope of this paper. Nevertheless, the distinction between locatable web resources and concepts is crucial.
3. Formal statements about the real-world artefact, using the formal identifier as a placeholder. We assume formal annotations to be represented as RDF [12]. Note that the formal statements can also include information about provenance or scope of the annotation.

## 4 Design

In this section we describe the design of the wiki interchange format (WIF) for a single page and the wiki archive format (WAF) for a set of wiki pages.

### 4.1 WIF – A single-page wiki interchange format

We map wiki pages to directories on a storage medium or in a zip file. Each folder contains:

**index.html** - the unchanged html file corresponding to the wiki. Includes all navigation buttons and forms. This view helps to identify content visually. CSS files should be included. This file can be obtained with simple HTTP-GET tools, such as `WGET`.

**wiki.txt** - the source code in wiki syntax. This is a simple to implement fall-back, if other steps in the conversion process produced wrong or suspicious output.

**wif.xhtml** - the wiki interchange format. In order to fulfill requirement "Renderable" (c. f. Sec. 2, WIF should re-use elements of HTML or XHTML. We decided to re-use XHTML elements, which are both open for machine processing, due to its XML nature as well as human viewable, using a standard browser. It is no problem to use elements not defined in XHTML: Browsers are requested to ignore unknown elements[9].

WIF should be valid XML. This allows to use XSLT stylesheets to convert WIF back into a wiki syntax of choice. XSLT stylesheets can be run on all platforms, many programming languages, and even in some browsers.

The number of WIF-elements should be small, in order to facilitate further processing (req. "ease of use"). A study by Google shows [10], that most HTML pages contain an average of only 19 different elements.

**index.rdf** - Annotations as RDF files. Meta-data, such as the distinction between wiki-link or external link is carried by using special XHTML attributes and values. This approach is inspired by Microformats[11]. More complex or user-given page meta-data should be stored in a linked RDF file.

In order to round-trip wiki specific features such as templates and macros, we represent them as annotations. The basic wiki page is exported as rendered. This fulfills requirement "Completeness" and ensures that another (even less capable) wiki, will show the page as the user knows it.

For macros and templates, annotations carry the additional hint that and how this part of the page was generated. Simple tools can only process the page as is, while smarter tools can exploit the knowledge of the annotations and e. g. update pages when a template is changed.

**Attachments** - Like emails, wiki pages can have arbitrary files attached to them. In order to store such files, we simply store them as files and link them from the WIF page. File creation date and other file related metadata can be encoded in the native file attributes.

Now we take a closer look at `wif.xhtml`. Which wiki structures are most important? We can distinguish three levels of formatting:

**Linking** is probably the most important element of a wiki. Wikis distinguish links to other wiki pages in the same wiki (wiki-links), links to pages in other wikis (interwiki-links) and links to any other web resource (external links).

[9] but process the content, for details see `http://www.w3.org/TR/xhtml1/#uaconf`
[10] `http://code.google.com/webstats/index.html`.
[11] `http://www.microformats.org`.

**Layout (inline-level)** is used to highlight parts of a text item. Basically, only bold and italic text can be used. For bold, HTML offers <strong> or <b>, for italic <em> or <i>. In single-page-WIF, we use only <strong> and <em>, to simplify its usage (req. Ease of Use). CSS layout is already separated from XHTML markup and is simply ignored.

**Structure (block-level)** is used to relate textual items on a page. Structural tags are paragraphs, headlines, lists and tables. Additionally, pages can contain horizontal lines and pre-formatted sections.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>...wiki page title ... </title>
  </head>
  <body>
     ... page content ...
  </body>
</html>
```

**Table 2.** The basic WIF page

A WIF-page thus has the structure shown in Fig. 2. WIF-Pages should always be UTF-8-encoded, to simplify further processing. The doctype should be XHTML 1.1. The `title`-element should be present and contain the wiki page title. The header may contain arbitrary additional elements.

To sum up, element tags used by WIF-processors in the WIF page body are: `a, dd, dl, dt, em, h1, h2, h3, h4, h5, h6, hr, li, ol, pre, strong, table, td, tr` and `ul`. Other elements in the body of a WIF page may be ignored. The WIF page has to be a valid XHTML document, i. e. the usual XHTML element nesting rules apply.

We reserve some attributes and values to encode special wiki semantics: In links (<a>), we use the attribute `class` to specify the nature of a link: `wiki`, `interwiki` or `external`. This approach is inspired by microformats ideas [12], and keeps WIF-pages valid XHTML and makes rendering in a browser easy. Note that an HTML-element may have multiple space-separated classes. As the title of a wiki page is often encoded, in order to make it a valid URL or filename, each link to an internal or external wiki page should also have a `title`-attribute with the wiki page title.

[12] `http://www.microformats.org`.

### 4.2 Linking pages with annotations

In order to keep the link between wiki pages (represented as XHTML) and their annotations (represented as RDF), we face a number of options. The problem is in principle the same as describing elements of one ore more XML documents with RDF. Keeping the link is important, e. g. to record from which parts of an XHTML page an RDF statement was derived from.

**Embed XHTML elements in RDF:** The idea is to mimic the XML tag relations with RDF predicates, similar to the ideas in [11, 5]. Thus we would end up having e. g. a relation `:hasHead` which corresponds to the <`head`> tag. The content of XHTML tags would be stored as *individual plain RDF literals*. Thus granularity of content would be exactly at the level of tags, which is not sufficient.

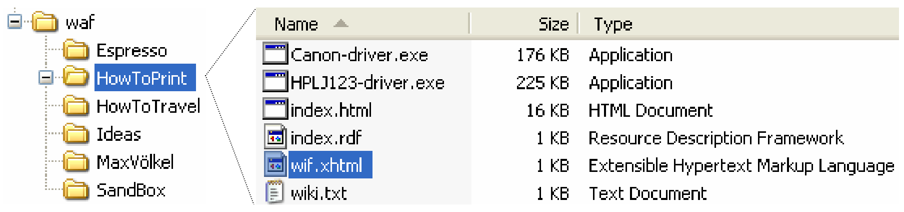
**Embed RDF in XHTML:** A W3C proposal dubbed "RDF/A" specifies how to encode (even arbitrary) RDF as XHTML *attributes* (hence the "A" in the name). GRDDL[13] can extract the RDF back out of the XHTML file. RDF/A files are not valid XHTML files, as they introduce some changes to the document structure. Additionally, current RDF infrastructure cannot operate on RDF/A files.

**Embed Page:** One could embed the whole XML file as *one big XML literal* into an RDF graph. Then we loose interoperability with current XML infrastructure, e. g. a browser cannot even show the embedded pages.

**Link RDF file:** We leave the XHTML content as a separate file, and point to an RDF file from the <head>-element. In this RDF file, we can reference the XHTML page using the local filename (e. g. HowToPrint.html).
Referencing parts of the XML file could be achieved by appending an XPointer [6] to the (relative) URI representing the XML document, e. g.
`wif.xhtml#xpointer(/html/head/title)` could denote the title of the WIF page stored as `wif.xhtml`. This approach is used by semantic web annotation tools such as CREAM [7] and Annotea [10]. Even before XPointer was specified, people used fragment identifiers to denote parts of documents [8].
The relation between the URI representing the XML document fragment and the XML document must be stated explicitly, as current RDF infrastructure does not handle XPointer expressions, e.,g. by a triple like adding `wif.xhtml#xpointer(/html/head/title)` `wif:partOf` `wif.xhtml`. With more formal defintions of the 'concept' namespace ID such statements could be inferred automatically. This approach is also used in CREAM [7] and Annotea [10].
However, there is a conceptual drawback: The web architecture and RDF speak about *resources*. An XML document is not a resource, but a *representation of a resource*. URIs are defined as *resource* identifiers. The representation returned by a server depends not only on the URI but also e. g. on the accept-header of the HTTP request. The semantics of the 'concept' namespace ID are thus defined explicitly only for XML content types. The XPointer specification uses the same approach.

[13] `http://www.w3.org/2004/01/rdxh/spec`.

For binary files, we propose a pragmatic approach: Each file called `filename.ext` can have an accompanying file `filename.rdf` which stores the metadata of that file. Both files are included in the wiki archive format, which is described in the next section.



- waf
  - Espresso
  - HowToPrint
  - HowToTravel
  - Ideas
  - MaxVölkel
  - SandBox

| Name | Size | Type |
|---|---|---|
| Canon-driver.exe | 176 KB | Application |
| HPLJ123-driver.exe | 225 KB | Application |
| index.html | 16 KB | HTML Document |
| index.rdf | 1 KB | Resource Description Framework |
| wif.xhtml | 1 KB | Extensible Hypertext Markup Language |
| wiki.txt | 1 KB | Text Document |

**Fig. 1.** WAF Example

### 4.3 WAF – The wiki archive format

Two use cases need a complete WAF, as opposed to a single-page-WIF:

**Wiki migration.** Here we have to handle the requirement to move a set of wiki pages at once. The easiest way to move multiple wiki pages across the network is probably to re-use an archive format such as the zip format. The same approach is taken in the Java world to handle a set of Java class files as a .jar file, which is a zip archive with a special structure. Another popular example of zip file usage is the open office format [14], which stores a set of XML files that together constitute one document.

**Wiki archiving.** Using a zip file with subdirectories for each wiki page has the additional benefit that, if done right, this archive can be extracted and viewed off-line, using a standard browser. Hierarchical namespaces can be modeled as subfolders.

We propose to use a zip archive consisting a set of WIF files. For Semantic Wikis, we also include RDF files, linked from XHTML files. For pragmatic reasons, we decided not to include different versions of a wiki page in the interchange format. WIF files should be legal XHTML files and the links between the pages should point to each other (e. g. a wiki page *a* linking to a page called "HowToPrint" would contain the snipped `...` `<a href="HowToPrint.html" title="HowToPrint" class="wiki">` `HowToPrint` `</a>` ... All WIF files should have a file extension of "`.html`". RDF files can have any extension (besides .html) as and should be linked from the XHTML files. Background knowledge, not related to a particular page should be stored as `index.rdf` in the root folder of the WAF zip file. An example of a WAF archive with six exported wiki pages is shown in 1.

[14] `http://www.oasis-open.org/committees/office/`.

### 4.4 Wiki Mediation Server

At runtime, we need a component that provides translation from one wiki into WIF and from WIF back into wiki syntax. For migration, the component should also interact with wikis through their web interfaces, simulating a human editor. This idea is similar to WikiGateway [17], but WikiGateway does not address the problem of page structure translation. In order to provide WIF-translation services also for other tools, we use a service oriented architecture, as shown in Figure 2. This architecture allows a user independent of the wiki engines *source* and *target* to migrate wiki content. The functions offered by the wiki mediation server are:

**GetPage(*t*)** retrieves a WIF representation of a wiki page *p*, given its title *t*.
**PutPage(*p*,*t*)** converts a page *p* given in WIF into a wiki syntax of choice and stores it under a given page title *t*.
**GetRecentChanges(*u*)** gives a list of pages (with URLs) that have been changed after a given point in time, including a time stamp indicating the last change.
**GetIndex()** returns a list of all pages stored in the wiki. The list should contain URLs.

We show now how this design solves the scenarios given in Sec. 2 and then describe the design of the specific functions.

**Migrating wiki content:** In order to migrate a page with title *t* from wiki *a* to wiki *b*, we call `migrate(`*t*`,`*a*`,`*b*`)` which calls in turn `x = a.getPage(p)` and then `b.putPage(x)`.
**Wiki synchronising.** In order to synchronise the wiki *a* with wiki *b*, the wiki mediation server has regularly to invoke a.GetRecentChanges, for all pages that have been changed after the last invocation of this function. Then, for each page *p* with title *t* that has been changed, we call migrate(*t*,*a*,*b*).
**Wiki syntax united.** In order to use an arbitrary wiki syntax *s* for a wiki *a*, we propose to use a proxy-wiki, which works as follows. For rendering a page with title *t*, we return a.getPage(t). When a user wants to edit a page, we call a.getPage(t), and convert the obtained WIF into syntax *c*. Upon page save, we convert the wiki text in syntax *c* back to WIF and put the result *r* using a.putPage(r) back into wiki *a*.

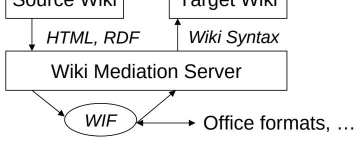


**Fig. 2.** Software Architecture for Remote Wiki Migration

## 5 Implementation and Experiences

A proof-of-concept implementation has been developed in Java made available as open source (LGPL) at `http://wif.ontoware.org`. Currently, it only exports a SnipSnap wiki – accessed via the web interface – into WAF. Support for JSPWiki import is being added. Output rendering for MediaWiki is available.

How do we obtain the data from a wiki? As each wiki has some kind of data persistence, the wiki administrator could access a wikis content on this layer. But each wiki has a different kind of persistence (e. g. text files, data bases, RDF), so writing adapters would require very different skills.

For open-source wiki engines, one could add a new export function to the code base. This would require to handle a large number of programming languages.

In order to migrate from abandoned wiki engines which are no longer maintained or where the user who wishes to migrate has not the necessary admin rights, we need another solution.

As almost every wiki has an HTML output, one could also try to get the content from there. Simulating a browser that clicks on "Edit this page" and other wiki core functions, one could get access to the same set of data as a user.

Current Wiki-APIs (XML-RPC based ones can be found e. g. in JspWiki[15], Atlassian Confluence[16], XWiki[17]) stop at the page graph level. A page can either be retrieved as raw text (wiki syntax) or rendered as HTML. The HTML version is often enriched by additionally inferred information, such as backlinks. Such information is harder to extract from HTML than from wiki syntax, e. g. the difference between wikilinks and external links can only be computed by comparing the base URL of a page with the link target. Nevertheless, Wiki engines use quite different syntaxes, but (almost) all wikis emit HTML in the user interface. Thus starting with the HTML syntax significantly eases development.

Semantic wikis offer the same ways of obtaining the data. Usually semantic data is made accessible via a link from the HTML version. Note that current semantic wikis do not retain the link which statement was generated from which part of the page.

As less then 1% of all web pages are valid (X)HTML [13], in the sense of passing the W3C validator[18], we cannot expect wikis to emit only valid HTML. Fortunately, there are a number of software components available, that mimic the parser behaviour in browsers to transform ill-formed HTML into well-formed (X)HTML. As analysed in [18], the best performing component is CyberNEKO[19]. CyberNEKO transforms any ill-formed HTML input into well-formed XML, add missing opening and closing tags as necessary.

Individual pages are read with the *Jakarta Commons HttpClient* and post-processed with custom XSL transformations into WIF. HTTP Basic Authenti-

[15] `http://www.jspwiki.org`.
[16] `http://confluence.atlassian.com`.
[17] http://www.xwiki.org
[18] `http://validator.w3.org/`.
[19] `http://people.apache.org/~andyc/neko/doc/html/`.

cation is used to get into protected pages, login data has to be supplied by the user as URL parameters.

Although we lose some information (such as slight syntax variations), we can obtain most information even from the HTML data. We simply compare the <base href> of the page with the link targets. If the query string or last path segment (for mod_rewrite) matches the target, it is a link to the wiki. As this matching is done in the stylesheet, more complicated transformations are possible. It is e. g. possible to detect Wikipedias Inter-language-Links and mark them up as such in WIF, e. g. by adding a CSS class for the target language to the link class. We ignore linked CSS stylesheets and some presentational tags, in order to get a more concise WIF, fulfilling requirement "Compactness".

The page index is read from the index page, which all wiki engines provde, and post-processed with XPath [3] expressions to get the actual page names.

Internally, the mediation server relies on Jetty[20], as an embedded web server, and JRest[21], as an automatic mapping from Java objects to RESTful servlets.

A demo server is currently set up, serving a WAF-archive for any SnipSnap wiki. A login is simulated in order to obtain the text-files in wiki syntax.

### 5.1 Experiences

No formal evaluation has been done. Instead, we review what we achieved.

Using WIF can dramatically lower the costs required to migrate wiki content. As in most integration problems using an intermediate format (WIF) reduces the number of translators needed from $n^2$ to $2n$. The process of writing the translators from HTML to WIF can partially be automated (see [18]. As WIF consists of less elements than full XHTML, XSL stylesheets to convert WIF back into wiki syntax are easier to write.

Future wikis can use the Wiki Mediation Server to offer real-time wiki syntax of choice. To do this, they have to act as a proxy. When a user edits a page, the user entered text in syntax $a$ is run through the parser of a wiki $a$, resulting in HTML. That HTML is converted first to WIF and then to wiki syntax $b$. This syntax $b$ is then stored in wiki $b$.

First tests show that the transformation from SnipSnap's HTML to clean XHTML and than via custom XSLT to WIF is indeed possible. Another XSLT was written to convert WIF to MediaWiki syntax.

Mapping wiki pages to folders leaves much freedom. The format is extensible, i. e. more files can be stored in the same directory, e. g. different versions. Complete static web sites can be generated from a WAF file, in fact, a WAF file *is* a static web site, in one zip file. This makes WAF also an ideal wiki backup format, with no need to keep the original server infrastructure alive. Alas some features (e. g. full-text search) get lost.

The problem with this approach is the reference management. To create a valid reference between static wiki pages, one has to link to `/PageName/index.html`, instead of just the page name.

[20] `http://www.mortbay.org`.
[21] `http://jrest.ontoware.org`.

## 6 Related Work

There are several proposals about wiki standardisation. The WikiModel[22], addresses an in-memory model for wikis in Java, using a particular semantic model (pages with sections, allowing page inclusion). WikiModel includes a clever wiki syntax and parser design.

WikiWyg[23] is an approach to offer in-browser WYSIWYG editing of wiki content. To do this, WikiWyg uses Javascript to convert from DOM to wiki syntax. Oddmuse[24] has an integration with Emacs [25] which allows users to read and update pages with Emacs, a desktop-based text editor.

Similar to WikiGateway, an extension to JSPWiki written by Janne Jalkanen [9], allows to get and put pages. DavWiki exposes the wiki pages as text files in a WebDAV directory. Note that the syntax conversion problem is left untackled.

SweetWiki[26] uses RDF/A to encode semantic annotations in HTML pages. Annotations can only be on the page level, not allowing annotating parts of a page. IkeWiki offers a custom XML-based export format. It is similar in spirit to WIF, as it exports only the core structures. But different from WIF, IkeWikis export is not suitable for viewing in a browser.

## 7 Conclusion and Outlook

We have shown how a wiki interchange format should be designed. Although our model is still in a prototype stage, we have carved the path for the development of a true Wiki Interchange Format. A reference implementation is available.

A standardisation effort needs consensus. We hope to continue the discussion that started at the WikiSym 2005 as *Wiki Standardisation Request 3* on the wiki page `http://www.wikisym.org/wiki/index.php/WSR_3`. The next steps are a more precise requirements gathering and and RFC-style document.

*Acknowledgments:* This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694. This research was partially supported by the European Commission under contract FP6-507482 (Knowledge Web) and FP6-027705 (Nepomuk). The expressed content is the view of the authors but not necessarily the view of the Knowledge Web Network of Excellence as a whole.

We thank Sebastian Gerke and Werner Thiemann for their help in creating the Wiki Exchange Format, and Dirk Riehle for initiating the BOF at the WikiSym 2005, where all this started. Thanks also to Mikhail Kotelnikov and Malte Kiesel for lengthy and fruitful discussions, and special thanks to Dirk Achenbach who went the painful way of manual wiki migration.

[22] `http://wikimodel.sourceforge.net/`.
[23] `http://www.wikiwyg.net`.
[24] `http://www.oddmuse.org`.
[25] `http://www.gnu.org/software/emacs/`.
[26] `http://wiki.ontoworld.org/wiki/SweetWiki`.

## References

1. M. Altheim. Inter wiki markup language (iwml), 03 2004.
2. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. Rfc 3986, The Internet Society, Jan 2005.
3. J. Clark and S. DeRose. Xml path language (xpath) version 1.0. Technical report, W3C, Nov 1999.
4. D. Dougherty and A. Robbins. *sed & awk (2nd Edition)*. O'Reilly Media, Inc., March 1997.
5. M. Erdmann and R. Studer. How to structure and access xml documents with ontologies. *Danta and Knowledge Engineering*, 2000.
6. P. Grosso, E. M. J. Marsh, and N. Walsh. Xpointer framework. W3c recommendation 25 march 2003, W3C, MAR 2003.
7. S. Handschuh and S. Staab. Cream - creating metadata for the semantic web. *Computer Networks*, 42:579–598, AUG 2003. Elsevier.
8. M. Hori, R. Mohan, H. Maruyama, and S. Singhal. Annotation of web content for transcoding. W3C Note, July 1999.
9. J. Jalkanen. Davwiki – the next step of wikirpcinterfaces? In *Proceedings of Wikimania 2005 - The First International Wikimedia Conference*. Wikimedia Foundation, JUL 2005.
10. J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared web annotations. In *Proceedings of the 10th International World Wide Web Conference*, pages 623–632, 2001.
11. M. Klein. *Interpreting XML via an RDF Schema*. IOS Press, Amsterdam, 2003.
12. G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`, February 2004.
13. M. L. Noga and M. Völkel. From web pages to web services with wal. In *NCWS 2003*, Växjö, Sweden, NOV 2003. Mathematical Modelling in Physics Engineering and Cognitive Science.
14. E. Oren. SemperWiki: a semantic personal wiki. In S. Decker, J. Park, D. Quan, and L. Sauermann, editors, *The Semantic Desktop – Next Generation Information Management & Collaboration Infrastructure*, Galway, Ireland, 2005.
15. E. Oren, J. G. Breslin, and S. Decker. How semantics make better wikis. In *WWW (poster)*, 2006.
16. E. Oren, R. Delbru, K. Möller, M. Völkel, and S. Handschuh. Annotation and navigation in semantic wikis. In M. Völkel and S. Schaffert, editors, *Proceedings of the First Workshop on Semantic Wikis - From Wiki to Semantics at the ESWC 2006*, May 2006.
17. B. Shanks. Wikigateway: a library for interoperability and accelerated wiki development. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pages 53–66, New York, NY, USA, 2005. ACM Press.
18. M. Völkel. Extraktion von XML aus HTML-seiten – das WYSIWYG-werkzeug d2c. Diplomarbeit, May 2003.
19. M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, May 2006.

# A Collaborative Programming Environment for Web Interoperability

Adam Cheyer and Joshua Levy

SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, USA
adam.cheyer@sri.com, levy@csl.sri.com

**Abstract.** We describe a new type of collaborative system that exhibits much of the simple, cooperative nature of a wiki, but allows dynamic sharing of functionality as well as of content. In contrast with traditional wikis, pages in this system are executable, and interoperate with each other by passing and returning data structures of known type, such as messages, URLs, or locations. This collaborative programming environment is well suited to retrieving and combining content available on the Web. Since code within pages can access any type of Web content, the environment provides a collaborative way to convert diverse, unstructured information into semantically annotated content that can be combined into new and useful services. We discuss how these ideas have been applied in WubHub, a prototype Web portal with a command-line interface.

## 1 Introduction

The rise of dynamic, massively multi-user Web applications has led to the rapid success of wikis, social bookmarking services, and similar tools, and begun to demonstrate the immense potential of large-scale collaborative software. Prominent recent successes include Wikipedia, Flickr, and del.icio.us [1–3], but current examples have become too numerous to list.

So far, most such systems have focused on the capture and retrieval of data that is mostly unstructured, like text (in wikis), or of a small number of predefined types, such as URLs (del.icio.us), images (Flickr), or tags (numerous sites). Increasingly, the key features of wiki-like tools – collaboration and ease of use – are being applied in new contexts, in systems that work with new types of data, and with data of increasingly rich variety.

This trend shows great promise in addressing the long-standing and difficult problem of semantic interoperability on the Web. A fundamental tenet of the Semantic Web effort is that increased semantic annotation increases opportunities for use of online services, including automated or semi-automated service discovery, composition, and invocation [4, 5]. At the same time, as the “Web 2.0” phenomenon has demonstrated, collaborative tools, even with very limited and informal semantic support, can dramatically increase the knowledge captured and accessible to users. The challenge – and the opportunity – is to find a way to accommodate greater semantic precision while simultaneously encouraging and reaping the benefits of large-scale community collaboration. Recent efforts ranging from microformats [6] to semantic wikis [7–9] are identifying a variety of approaches to reconciling these apparently conflicting goals.

In this paper, we outline a somewhat different angle on the same problem. Most existing collaborative sites, including newer systems like semantic wikis, are content oriented. We describe a collaborative framework that is *service oriented*, permitting sharing of services as well as of content. It is a collaborative programming environment that can access and integrate existing online content and services. In particular, it can extract structured information from existing unstructured Web content and combine it to produce new functionality. The collaborative process of defining new services simultaneously enables semantic annotation of existing unstructured data. Although it is still immature, we believe this framework could lead to practical and effective approaches to solving some current problems in Web interoperability.

In the next section we give a brief example of collaboratively defined services. Section 3 addresses design considerations for the underlying collaborative programming environment. Section 4 describes our proof-of-concept implementation. Sections 5 and 6 hold some additional discussion and related work. Future work is described in Sect. 7.

## 2 A Collaborative Service Portal

We motivate our discussion by sketching some examples of user interaction with WubHub, a proof-of-concept Web portal we have built that demonstrates collaboratively programmable services.

### 2.1 Calling and Combining Commands

The user visits the Web portal. Its interface consists of an input field – a Web-based command line – and a frame that displays content. The user can type in commands that perform many different services, such as retrieving information, performing an action or computation, or converting data. Commands can accept input as arguments, and provide a return value, which is displayed in the content frame. Commands can perform many useful everyday tasks. Figure 1 shows some typical commands. Figure 2 shows the portal in use.

Commands in WubHub can be composed to produce novel functionality, either within a new command definition or directly on the command line. For example, a user might employ three independently contributed functions to display a map with friends' locations, by typing

```
map: geocode: friends()
```

### 2.2 Finding and Editing Pages

What is novel about WubHub is not the commands themselves, but that these commands are built collaboratively by users. Just as wiki pages may be created

- `distance(94702, 94025)`: Use an online map service to calculate directions between two US zip codes, and return the driving distance between those points.
- `dict(forgetful)`: Look up the word "forgetful" in an online dictionary, extract the definition from the page, and return it to the user.
- `geocode("San Francisco, CA")`: Return the latitude and longitude of the specified address.
- `slashsearch(linux)`: Get a list of links to recent Slashdot articles mentioning Linux.
- `azn(dickens)`: Redirect the user's browser to Amazon's search results for "Dickens."

**Fig. 1.** Some example WubHub commands.

by one user and improved upon by others, WubHub pages containing content, presentation logic, data conversion, or computational functions can be woven together in an iterative way by a distributed community. Commands that perform services are written as pages containing a scripting language that allows access to remote Web sites and flexible manipulation of structured and unstructured data. Like conventional subroutines, pages may call other WubHub pages. The portal also provides simple development support, allowing the user to try writing new scripts and to debug them.

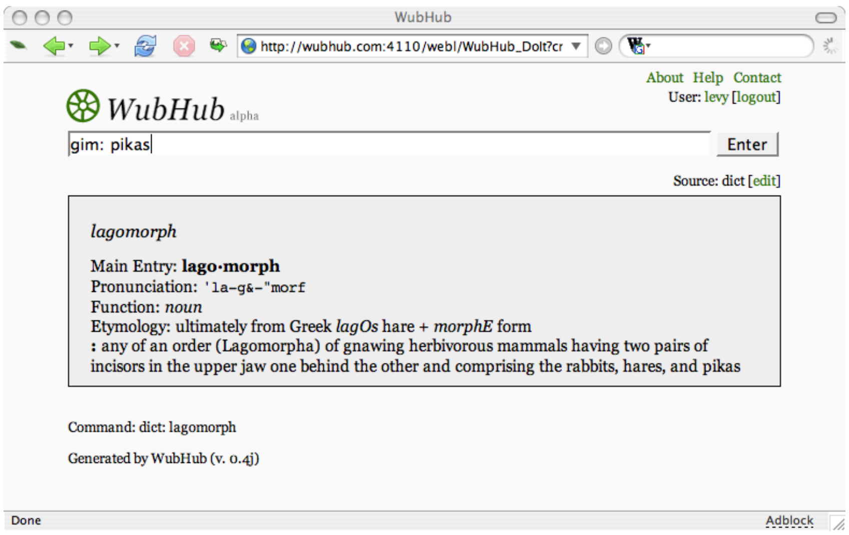
As users create new content and commands, they can organize and search for them using collaborative tagging. Additional meta-data makes it easy to discover pages added by the community using commands and subscriptions such as "whatsnew" and "popular" (Fig. 3). Finally, users can view the source code to any WubHub page, including system commands, and either modify the page directly (if they have sufficient privileges) or copy it to a local space where they can make their own variants. Just as a "view source" capability was essential to the proliferation of the HTML-based Web, we believe it can accelerate innovation and adoption in a WubHub-like environment.

### 2.3 Collaborative Programming and Semantic Annotation

In short, WubHub is a collaborative platform for service development, where users can add new services to the system, and data can be passed from service to service easily. In addition, since the scripting layer has the ability to extract data with known semantics from unstructured, ad hoc Web content, it can simultaneously provide a way to add semantic annotation to unstructured data on the Web.

## 3 Collaborative Programming

The collaborative service portal we have sketched is an example of a general type of collaborative system: A lightweight, highly collaborative programming



Source: dict [edit]

*lagomorph*

Main Entry: **lago·morph**
Pronunciation: `'la-g&-"morf`
Function: *noun*
Etymology: ultimately from Greek *lagOs* hare + *morphE* form
**:** any of an order (Lagomorpha) of gnawing herbivorous mammals having two pairs of incisors in the upper jaw one behind the other and comprising the rabbits, hares, and pikas

Command: dict: lagomorph

Generated by WubHub (v. 0.4j)

**Fig. 2.** A screen shot of WubHub, showing the result of executing a command to get a definition of a word. The user is now typing in a second command that will redirect to a Google image search for "pikas."

environment. This notion is quite new, and the proper design for such an environment, or even a complete set of requirements for one, is not yet fully apparent. However, our experience with WubHub has helped illuminate some basic requirements and an architecture, which we outline in this section.

## 3.1 Pages as Functions

The environment's main elements are

- Shared storage for *pages* holding pieces of content or code
- An execution environment, with a programming language and type system
- A user interface, with support for invoking commands and wiki-like editing of pages

A *page* in the environment is like a function in a programming language, with arguments and a return value, that is visible to, and modifiable by, many users. Essentially, the wiki-like editing capabilities are a front end for editing function definitions, and these functions may be executed, as if they were in a conventional (single-user) scripting environment. As in a traditional wiki, pages may actually contain only declarative data, such as text or HTML, which is really a special case where they accept no arguments, and provide their content as a return value. In effect, pages are variables within the programming environment, which may hold either functions or data.

– `ls`: List visible commands.
– `tags`: Browse pages by community tags.
– `whatsnew`: List recently created commands.
– `view(whatsnew)`: View the source code to the `whatsnew` command.
– `edit(whatsnew)`: Open a form for editing the source to the `whatsnew` command.
– `create`: Open a form for creating a new command.

**Fig. 3.** Selected commands for discovering, creating, and modifying pages.

### 3.2 Page Storage

Page storage is shared, so that all users have immediate access to pages from other users. Essentially, we desire a shared, transparent program and data persistence mechanism. Pages may be called from other pages directly, as if they were functions defined within the programming language.[1]

In addition to the page name and the body of a page, page records include signature information, a description, and other meta-data, such as tags added by users. For better organization, pages may be arranged into a hierarchy of modules. In general, the typical features of content repositories, including search, versioning, conflict detection, and access control, must be supported. A primary goal of the system is to encourage collaboration, so it is natural to make most content readable by all users.[2]

### 3.3 Programming Language

Pages must be written in a programming language. The requirements for such a "collaborative programming language" are a somewhat different than for conventional single-user programming languages. Some desired language features are

– *Interpreted (or dynamically compiled) execution*: We need the ability to compile or interpret new source code at all times.
– *Strong, dynamic typing*: Strong data types within the language are essential, so that it is possible for a routine to know the type of a value, and to ensure safe execution. Although static variable types could work in principle, in this environment the ease of use and flexibility of dynamic types may outweigh the benefits of compile-time type correctness.
– *A rich, extensible set of types*: The set of possible data types can be large, with a subtype relation and possibly other relations as well. It must be possible to add new types at runtime.

[1] The term "page" may be slightly misleading, since a page holds code or data, not a Web page. We use this term simply to distinguish pages from variables or functions in most programming languages, which are not necessarily persisted and do not hold the same meta-data.

[2] In principle, users could publish pages that are executable but not readable, but this "closed source" approach removes opportunities for collaborative bug fixing and individual customization.

– *Safe code execution*: The execution environment must provide security mechanisms that permit safe execution of untrusted code. It is necessary to run pages within a sandbox that minimizes the likelihood that scripts will compromise the host system's integrity (e.g., modifying system files), confidentiality (e.g., exposing sensitive local data), or availability (e.g., consuming excessive memory, processing, or bandwidth). The sandbox should also provide mechanisms that restrict abusive behavior that could adversely affect remote systems (such as posting comment spam to open websites).
– *Access control*: While not essential in all applications, access control support for data and code within the environment (such as user-based read, write, and execute permissions on pages) is highly useful.
– *Library support*: Support for rich data structures and parsing tools to convert ad hoc data formats to convenient internal representations, particularly for ubiquitous formats such as HTML and XML.
– *Ease of use*: For collaboration to work on a large scale, the language must be accessible to casual developers.

We know of no existing programming languages that fully meet all the above needs, but several popular languages, including Ruby, Python, JavaScript, and Lisp/Scheme, come close to meeting most of them. The least-supported language feature, and arguably the greatest technical obstacle to building the execution environment, is safe code execution. The Java platform is one of the few general-purpose programming platforms that provides a mature and full-featured sandbox model [10, 11], though this model still does not support resource restrictions, such as CPU or network bandwidth limits. Ruby provides some sandboxing support, but it is not as expressive or mature. Probably the most widely used execution sandbox is the JavaScript environment within most Web browsers.

### 3.4 Data Types

Types in a collaborative programming language form the glue that enables multiple functions to work together, extracting, aggregating, processing, and rendering information. Values may have numerous types, such as numbers, strings, lists, URLs, zip (US postal) codes, HTML Web pages, or street addresses. A type (e.g., "movie listing") has relationships with other data types, in particular subtype (e.g., "movie listing" is a kind of "event") and containment (e.g., a "movie listing" includes a time, duration, a movie, and a location of type "address").

As users develop new pages that utilize new data types, they are effectively adding to an ontology of possible data types. New data types can be created and used in response to needs for particular services. For instance, an HTML page with driving directions might contain a map, an address, an estimated distance, and other information. Various services might extract useful data from such a page. A service interested only in the map could create a data type for storing the map image, while a distance-computing service would use (or create) only a numeric type for distance.

### 3.5 Composing Services

Pages should be able to pass and return data values. Once content from the Web is brought into the system as a known data type, it can be passed as an argument to any other page that accepts that data type. New services can be built by composing the functionality offered by multiple pages. For example:

- One user writes a page that uses an online directions service to compute the distance and driving time between two zip codes. Another user writes a custom page that accepts a single address and determines the driving time from her house to that address.
- Several users write pages that query various online event-listing websites for upcoming events. Although the websites present the data in different formats, each page would extract information and return its result as an "event" data type holding a title, description, date, and URL. Another page is used to aggregate upcoming events at a particular location into a single array of objects, and a final page can be used to render them in a table.

### 3.6 Rendering and Data Conversion

The execution environment requires support for display of all kinds of data values. This is particularly helpful in keeping a clean separation between the underlying representation of data values, and the various presentations of them shown to the user. A simple technique is to build rendering functions that accept a value of a particular type and convert it to a presentable form (such as a fragment of HTML). These functions are simply pages themselves. For convenience, types have default rendering functions. That is, when a value is returned for display to the user, the system checks the value's type to determine an appropriate rendering page. The rendering page is called with that value as its argument, and the result of that call is now an immediately displayable value (for a Web interface, HTML).[3]

This type of rendering can actually be considered a special kind of data conversion: in effect, it is implicit type coercion in the programming environment. Other simple, automatic conversions can be convenient, such as converting an ISBN to a "book" object with author, title, and so on. Going further, it is possible to imagine an extensible registry of data conversions, specifying which types can be converted, the circumstances under which the conversion may occur, and what page implements the conversion.

### 3.7 User Interface

Users can interact with the system in two ways: To issue "commands" that dispatch existing services, and to build new services. The user interface for issuing

[3] If a user-defined type does not have a default rendering function defined for it, a generic object rendering function is chosen. Also, it can also be useful for a page explicitly to select the rendering method for its return value.

commands simply collects a page name and an optional set of arguments. A simple way to do this is to provide an actual command line, much like a shell prompt. More graphical user interface widgets can provide similar capabilities in a more polished form.

The programming interface is potentially more complex, and ideally is closer in spirit to an integrated development environment. The simplest possible editing environment is inspired by wiki editing, and consists of a set of pages that can be edited from within a Web browser. The benefits are simplicity and ease of use. However, additional features can facilitate development of new pages. One particularly useful feature is a read-eval-print loop for executing script expressions. For instance, when trying to extract content from a Web page, the Web page can be fetched and parsed, and then the user can interactively try evaluating various expressions to determine how to extract data of interest. Pages can also be invoked interactively during the development process. Of course, as with any development environment, more advanced graphical interfaces could offer shortcuts for composing existing functions or pages.

### 3.8 Web Protocol Support

The framework can interoperate with other Web services only if it has library support for relevant protocols. Structured formats such as RSS, SOAP, or Semantic Web service protocols are useful, but support for common, human-browsable (and in practice, non-standards compliant) HTML is also key, as this format is still the dominant form for free information on the Internet. In particular, there should be a parser and document representation (such as the W3C document object model [12]) that allows "Web scraping" and general manipulation of XML and HTML documents.

### 3.9 Architecture

We have mentioned all three key architectural elements of the framework: a storage mechanism, an execution environment, and a front end. The simplest deployment is to put all three parts on a single server. In this case, a Web-based user interface can be provided by a Web application server, which hosts the execution environment and the page storage. The thin clients are simply Web browsers.

Other architectural arrangements may offer greater benefits. The only element that must be shared, and hence is most easily kept on a centralized server, is the storage component. Page execution and the user interface could be deployed on a thicker client. This option may no longer offer a "zero install" client, but has the advantages of greater scalability. Another critical factor in this choice is risks of and incentives for abuse. Execution on the client removes incentives for abusing server resources, but increases opportunities for malicious client-side exploits.

```
// Fetch page, using localsearchmaps.com's free geocoder
var P = GetURL("http://www.localsearchmaps.com/geo/?loc=" + Url_Encode(addr));
// Parse results, extracting the numbers (lat, long)
var LatLong = Pat(P, `([-\d.]*)`);
if (Size(LatLong) > 2) then
   [. long=LatLong[1][1], lat=LatLong[2][1], label=addr .]
else
   nil
end;
```

**Fig. 4.** The source code for the `geocode` page, which accepts a parameter `addr`, passes it to a free geocoder service, and then extracts the coordinates from the results.

## 4 WubHub Implementation

### 4.1 Features

We have built a simple prototype of the environment, the WubHub portal, and, as shown in Sect. 2, it demonstrates most of the basic service editing capabilities we have described. Some of the more advanced features we have described, such as a rich type system or an extensible system for implicit data conversion, are not yet fully supported.

Our prototype is built with and uses the WebL programming language [13], a small scripting language for the Java platform that has built-in support for fetching, manipulating, and creating Web content. WubHub supports a variety of page types: WebL, HTML templates, plain text, URL redirects, and aliases. WebL pages are simply functions, written in the WebL, that accept parameters and return a value. (An example of a short WebL page is displayed in Fig. 4.) The HTML type provides a convenient way to create regular Web pages, like in a wiki, as well as a templating feature, to display results of WebL computations with HTML formatting.

When a user invokes a command, the corresponding page is executed, and the result is rendered and returned to the user. In the case of HTML templates, the page, with appropriate substitutions, is returned. In the case of URL redirects, an HTTP redirect, with appropriate substitutions, is sent to the browser. All commands have a corresponding invocation URL, so commands such as search results or data feeds can be bookmarked by users. For instance, if a command returns data in RSS format, a user could direct an RSS reader to subscribe to the URL of that command.

Pages are stored within a number of modules, including a global module, a system module, and individual modules for each user. All pages are completely public for viewing and execution, although a user may not modify pages in the system module or in the module of another user.

### 4.2 Implementation

The WubHub implementation is quite simple; in fact, WubHub is largely written using itself. Almost all basic features, such as page creation and page editing, are themselves handled by pages. (To prevent accidental or malicious damage to site functionality, these important pages may be edited only by site administrators.) One advantage of choosing WebL is that it provides all needed parsing tools for real-world, ill-formed HTML. It provides XML and HTML parsing, selection, search, and creation capabilities as native parts of the language. Data is passed between pages via native WebL types, which can represent HTML fragments, strings, numbers, arrays, and objects. Objects are used to create new types, and hold a set of named fields.

The execution sandbox consists of two layers. Because WebL is a very small language, it is relatively straightforward to check for any commands that might be abused, such as file-system operations. A second layer of protection is via a standard Unix chroot() environment. We have not yet implemented any resource constraints within the sandbox, so it is still possible to consume excessive CPU or network bandwidth.[4]

### 4.3 Deployment and Use

WubHub is remarkably easy to use as a portal that provides simple services rapidly. In many cases, it is faster and more efficient to use a command that performs a Web query than to browse the original Web site. As a special case, redirects to a URL do not provide any data extraction capabilities, but behave more like a bookmarking or search service.

We have deployed the system for use by a small community of pilot users, who have contributed a variety of simple commands. Examples range from assembling search results from well-known news or search engines, to sending SMS messages, to building statistical charts.

The current version of WubHub is a proof of concept only, and requires a number of improvements for broad use. A key difficulty is ease of programming. WebL is a somewhat arcane language, and is completely unfamiliar to most users. Debugging support via the Web is fairly limited. The type system should also be enhanced. Objects are essentially structures with named fields, without inheritance, so implementing more complex types and relationships between types is cumbersome. Finally, scalability and security, including resource throttling, are further areas that will require more substantial development effort.

## 5 Discussion

### 5.1 Practical Issues

The use of hand-coded Web scraping tools instead of well-defined Web service protocols raises an immediate concern: Because there is no defined service API,

[4] One approach to enforcing such constraints is to build a supervisor thread that monitors running threads and terminates those that exceed certain limits. In the meantime, to discourage abuse, we have employed an invitation system, where members can join only via an invitation from an existing member.

the code for extracting information from an unstructured Web site could break whenever the Web site changes the way it presents its information. On the other hand, the community of users has an interest in correcting such problems, and we should take care not to underestimate the strength of numbers.

Another issue is that some content providers legally restrict non-interactive access to their content (for example, because they depend upon click-through advertising revenue). We believe, however, that future Web applications will trend toward "software-as-a-service" publishing of functionality, and that business models are likely to arise that will encourage providers to participate in a collaborative, Web-scale marketplace of services and content.

### 5.2 Type Systems and Ontologies

Exactly what sort of type system is best suited to this environment is not yet clear. At one end of a spectrum, types and type checking could be quite simple; for instance, the data types could consist only of primitive types, arrays, and named records, and type checking would involve raising runtime exceptions when invalid operations, such as accessing invalid fields, occur. At the other end, we can imagine rich types with a great deal of semantics, and possibly reasoning support as part of the type checking process. For instance, some services apply only to data values that satisfy certain conditions – say, geographical locations within North America – and these subtype conditions could be checked automatically, either at development time or at runtime.

On a recent collaborative research and development project,[5] the authors approached this issue from the semantic side of the spectrum, employing OWL, Jena, and a "semantic object" framework to model system data and services [14]. However, for the applications and community targeted by WubHub, we have found it is an advantage to follow the philosophy of collaborative systems like wikis, where there is strong emphasis on ease of use and a low barrier to users correcting problems. A more relaxed approach to data typing can lead to semantic imprecision and bugs, but this risk is counterbalanced by rapid, iterative modeling and greater numbers of users identifying and correcting problems. Loose typing practices such as "duck typing," which has become popular among Python and Ruby programmers [15, 16], are also useful, as they encourage aggressive reuse of code.

### 5.3 Three Approaches to Semantic Annotation

One way to put the collaborative programming approach in context is to consider some of the other approaches to semantic annotation of unstructured content. In the traditional Semantic Web vision, information from all sources is transformed from unstructured formats to standardized semantic representations. Unfortunately, this vision has been relatively slow to take shape. A key impediment is

[5] CALO (http://www.caloproject.sri.com) is a DARPA-funded project where SRI is leading 25 subcontractors to construct a personal assistant that learns.

that interoperability typically depends on data providers converting or adapting their data to a new standard before it can be consumed by others – an expensive process, and one for which there is often no strong incentive to be an early adopter. Moreover, a standardization process is needed before potential users gain enough confidence to adopt a new format. In particular, with this approach, benefits are not immediate or incremental.

A second approach is to leverage the power of a community to collaboratively annotate content. Collaborative systems can solicit relatively small contributions from many early adopters, and do incrementally increase in value as more contributions are added. Note, however, that users must still do the work of migrating content into a new format and repository before it is useful (such as with del.icio.us).

But can we go one step further, supporting collaboration and also allowing existing data to remain in its existing formats and locations? This may be possible by shifting our focus from collaboratively developed content to collaboratively developed services. If collaborative infrastructure becomes powerful enough that users can share annotation *mechanisms*, rather than just annotated data itself, we can benefit from community collaboration to extract semantic information from many existing, unmodified data sources. Because the construction of new services is possible immediately, the framework provides an incentive to contribute, which accelerates adoption.

## 6 Related Work

### 6.1 Programmable Websites

Executable code within a wiki is not a new idea; some existing wikis already support dynamic variables and other execution directives (e.g. ZWiki [17]).[6] More ambitious "programmable wikis" have been discussed [18], but none have become mature or popular, mainly because of the difficulties of sandboxing (particularly for legacy wiki systems, such as those based on PHP or Perl), and perhaps also because of a lack of compelling example applications.

The inspiration for WubHub's command-line interface comes from the collaborative website YubNub [19], which allows users to define and share commands that redirect to other Web sites, particularly Web searches. However, it has no general programming mechanism.

A few websites have recently begun to allow users to develop new Web applications through the Web. These include Ning, YouOS, and EyeOS [20–22]. However, most of these sites aim to replicate a traditional development style, with a single developer (or perhaps a single team) producing a standard Web or desktop-type application. As far as we know, our system may be the first to encourage users to compose services from each other's contributions, so that a single service is actually the result of a community effort.

[6] Of course, Web application tools such as JSP, PHP, or executable templates provide another example of dynamic pages, but these pages are themselves rarely edited collaboratively.

### 6.2 Widgets and Browser Tools

Several systems, such as Yahoo's Konfabulator [23] and Google Widgets [24], encourage a community of developers to create distributable functionality that can be inserted into the user's workspace. In contrast with WubHub pages, each widget provides a standalone block of functionality tightly connected to graphical output and cannot be composed to produce additional functionality.

Active browsing, such as implemented by the Greasemonkey [25] browser plug-in, allows users to program their Web browsers to manipulate Web content. The purpose of scripts is usually to make specific adjustments to pages as they are browsed, such as adding links to related content. These "user scripts" can be shared via centralized websites [26], but currently there is little support for fully extracting content from websites or for composing scripts to get new functionality.

### 6.3 Semantic Web Services

Semantic Web services [27, 5] are the standard approach to providing services on the Semantic Web, and some subsequent research has investigated ways to involve users in semi-automated composition of services [28, 29]. There have been a number of tool-building efforts related to the major Semantic Web services initiatives. For example, the Protege-based OWL-S Editor [30] has facilities for creating and composing OWL-S service descriptions, and provides a graphical editor that supports service composition and allows discovery of relevant services from local or remote service repositories. Maximilien et al. [31, 32] have discussed orienting Semantic Web services around human activities, and argue that keeping humans in the process of Semantic Web service automation is an important way to encourage widespread adoption. Aspects of this argument are very similar to the case we have made for involving users in the creation of new services.

### 6.4 Other Collaborative Environments

The Croquet system [33] is an effort at building a graphical collaboration system architecture. It shares some key features with the environment we have described, including the blending of user and development environments and a shared space where users can interact with content and scripts. It differs in that it is a larger, more ambitious system, with an emphasis on 3D visualizations, and it focuses more on users and user interactions within the system, rather than interoperable software and services and interaction with the Web.

While the setting is quite different, some similar collaborative programming ideas have previously arisen in the context of multi-user online "MUD" games. For instance, the MOO programming language [34] allows users to program the MOO game server in an object-oriented way, adding rooms or objects with scripted functionality. User-scripted functionality is also possible in some recent online games, including Second Life [35].

## 7 Conclusions and Future Work

We have described a collaborative programming environment where users compose and create new services that access nearly any type of existing online services. A key strength of the environment is that members of the user community can build upon each other's work, assembling new services that provide new or more customized features. As a part of this process, legacy content and services are reshaped into a community-extensible set of known data types, effectively adding semantic annotation to existing Web content.

The WubHub prototype has demonstrated that users can collaboratively write and compose services by using such a framework. It also presents numerous opportunities for integration with other applications, such as allowing WubHub-supplied data to be integrated into desktop widgets via remote procedure call. However, as we have discussed (Sect. 4.3), a variety of implementation issues must be addressed before the system is ready for broad use.

The requirements and design we have outlined for the collaborative programming environment leave much room for future work. As already mentioned (Sect. 3.3), popular programming languages do not provide all the language features desired for such environments, so there are opportunities for designing language enhancements that make collaborative programming easier, more powerful, and more practical. Key areas include enhanced security and permissions management, and greater flexibility and semantic precision for data types. The programming environment also needs more modular software development support, user interface enhancements, such as graphically specified compositions, and more powerful debugging tools, such as automated testing and perhaps type inference or static analysis. Such enhancements present both technical and social challenges: In addition to solving knowledge representation and software engineering issues, they must also encourage productive community effort. If both aspects are addressed, the result may be a significant step toward functional and ubiquitous online services.

## 8 Acknowledgments

We wish to thank Jonathan Cheyer for his help with deployment, David Martin and Natarajan Shankar for their comments, and the WubHub users who have contributed ideas and their own new services, including John Bear and Andreas von Hessling.

## References

1. (Wikipedia) http://www.wikipedia.org.
2. (Flickr) http://www.flickr.com.
3. (del.icio.us) http://del.icio.us.
4. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (5) (2001) 34–43

5. Martin, D., McIlraith, S.: Bringing semantics to web services. IEEE Intelligent Systems (2003) 90–93
6. (Microformats) http://www.microformats.org.
7. (Semantic MediaWiki) http://meta.wikimedia.org/wiki/Semantic_MediaWiki.
8. (Platypus Wiki) http://platypuswiki.sourceforge.net.
9. (IkeWiki) http://ikewiki.salzburgresearch.at.
10. Gong, L., Mueller, M., Prafullchandra, H., Schemers, R.: Going beyond the sandbox. In: USENIX Symposium on Internet Technologies and Systems, Monterey, CA (1997) 103–112
11. Gong, L., Schemers, R.: Implementing protection domains in the Java Development Kit 1.2. In: Internet Society Symposium on Network and Distributed System Security, San Diego, CA (1998) 125–134
12. W3C: Document Object Model (DOM) level 1 specification (1998) http://www.w3.org/TR/REC-DOM-Level-1/.
13. Marais, H.: Compaq's Web Language: A programming language for the Web (1999) http://www.hpl.hp.com/downloads/crl/webl/library.html.
14. Cheyer, A., Park, J., Giuli, R.: IRIS: Integrate. Relate. Infer. Share. In Decker, S., Park, J., Quan, D., Sauermann, L., eds.: Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland. Volume 175. (2005)
15. van Rossum, G.: Python tutorial. (2005) http://docs.python.org/tut/tut.html.
16. Thomas, D., Fowler, C., Hunt, A.: Programming Ruby: The Pragmatic Programmer's Guide. Pragmatic Programmers (2004)
17. (ZWiki) http://www.zwiki.org.
18. (Meatball wiki's discussion of programmable wikis) http://usemod.com/cgi-bin/mb.pl?CommunityProgrammableWiki.
19. (YubNub) http://www.yubnub.org.
20. (Ning) http://www.ning.com.
21. (YouOS) http://www.youos.com.
22. (EyeOS) http://www.eyeos.com.
23. (Konfabulator) http://widgets.yahoo.com.
24. (Google Widgets) http://www.google.com/ig/directory.
25. (Greasemonkey) http://greasemonkey.mozdev.org.
26. (UserScripts) http://www.userscripts.org.
27. McIlraith, S., Son, T., Zeng, H.: Semantic web services. IEEE Intelligent Systems (Special Issue on the Semantic Web, March/April) (2001)
28. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Workshop on Web Services: Modeling, Architecture and Infrastructure, in conjunction with ICEIS2003. (2002)
29. Sirin, E., Parsia, B., Hendler, J.: Composition-driven filtering and selection of semantic web services (2004)
30. Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., Senanyake, R.: The OWL-S editor: A development tool for Semantic Web services. (2005) 78–92
31. Maximilien, E.M., Cozzi, A., Moran, T.P.: Semantic web services for activity-based computing. In: Proceedings of 3rd International Conference on Service-Oriented Computing (ICSOC 2005), Amsterdam, The Netherlands (2005)
32. Maximilien, E.M.: Semantic web services for human activities (2005) To appear.
33. Smith, D.A., Kay, A., Raab, A., Reed, D.P.: Croquet: A collaboration system architecture (2003) http://www.opencroquet.org/About_Croquet/whitepapers.html.
34. Curtis, P.: LambdaMOO programmer's manual (1997) ftp://ftp.research.att.com/dist/eostrom/MOO/html/ProgrammersManual_toc.html.
35. (Second Life) http://secondlife.com.