# Ontology-driven generation of wiki content and interfaces

Angelo Di Iorio, Alberto Musetti, Silvio Peroni, Fabio Vitali

*Department of Computer Science, University of Bologna*

Mura Anteo Zamboni 7, 40127, Bologna, Italy

*(Received 6 January 2010; second version received 30 April 2010)*

The planetary success of Wikipedia has opened the road to using wikis as shared resources for communities to collect and organize facts, concepts and structures that constitute both the shared knowledge of the community and, more often than not, the very reason for the community to exist.
The ease of creating, editing and debating one's own and each other's contributions to the wiki knowledge-based are key aspects of the success and livelihood of the community itself. The need for semantic wiki data cannot be separated from the need of friendly authoring environments for those data.
This paper introduce a framework that allows users to easily create semantic wiki content by exploiting ontology-driven forms and templates. The system, called OWiki, is an instantiation of a more general model, named GAFFE, that exploits ontologies to generate metadata editors. Both GAFFE and OWiki are presented in this paper, with particular attention to the way they exploit ontologies to model the community shared knowledge, the interfaces used to create that knowledge and the way it evolves.

Keywords: ontologies, OWL, Model-View-Controller, OWiki, interfaces

## 1. Introduction

Every thriving wiki faces a teething problem some time after its tumultuous early periods: unconstrained development produces a large quantity of articles of very different quality, and, especially, with wild differences in style, organization, and detail level. Furthermore, key and undisputed facts about the topic of a wiki article are some times lost within a long (and possibly highly disputed) text flow so that it is difficult to immediately find them and subsequent edits may very well remove them even if inadvertently.

Refactoring a wiki to organize its content by means of more constrained development patterns is a long and painful process and care must be taken not to discourage the existing members in their contribution. One common approach is to separate undisputed facts from the actual articles, and to keep the structured data visually separated from the text content of the article. Templates, and particularly Mediawiki's infoboxes, are often used for this purpose.

But a key aspect in the refactoring of the wiki is the ability to impose clarity of meaning and homogeneity of organization in the structured data of all the articles dealing with topics of the same kind. A rich wiki with well-organized structured data associated to articles can be used for many purposes besides reading: sophisticated searches, classifications, and even automatic inferences become possible if the structures are correct, systematic and homogeneous.

A good approach to this end is to use ontologies to describe the type and number of structured annotations associated to individual wiki articles, and to have well-known Semantic Web

tools provide additional sophisticated functionalities on top of the standard wiki services. Semantic wikis thus exist to enrich wiki content with semantic data amenable to being exploited by reasoners, semantic query engines, etc. Usually wiki articles are associated to instances of the main classes of the underlying ontology, and structured data thus correspond to assertions whose subject is the topic of the page itself. Using categories for wiki pages is also a common way to to associate each wiki page to the class it is an instance of, thus making sure that the assertions the are placed in the page refer to the correct ontological classes.

Many existing approaches to Semantic Wikis, though, are mostly meant for scholars and experts in semantic technologies, and are often not appropriate for average contributors of community wikis with no background in OWL and related languages and concepts. Thus tools need to be devised that allow semantic assertions to be provided by all users regardless of their competence in semantic technologies. Furthermore, it is important that such tools do not discourage the contributor from spending time adding the requested information, and all efforts are made for the editing experience to be short, easy to perform and satisfying. Forms are often used to this end, providing an easy to implement interface that is also well known to most computer users. Unfortunately semantic forms as they exist currently do not guarantee the simplicity and ease of use that average users may require.

In order to define ease of use of an editing tool independently of the conceptual domain of the editable data we identify four basic qualities, namely *genericity* (based on arbitrary ontologies), *customizability* (editing forms adapt to the characteristics of the domain and the needs of the users, *proactivity* (voluntarily and systematically providing suggestions to users), and *validation* (verify immediately the plausibility of the values provided by the user). Furthermore, a good interface should employ a variety of different widgets specifically appropriate to the data to be inserted and the competencies of the users, it should provide intelligent heuristics for suggesting values so as to require the user to only insert few new values, change few others, but mostly approve the values as they were proposed, and it should provide a rich and complete set of rules to determine the appropriateness and correctness of the provided values.

In this paper we introduce a methodology and a tool to provide a good user experience to non-technical users that associate semantic assertions to wiki content, i.e., that add structured information to a wiki page in a manner that is amenable to semantic exploitation. Our approach, called GAFFE. (Generator of Advanced Forms and Friendly Editors), is an MVC-driven way to associate semantics to a textual document by having a domain ontology and a form ontology cooperate to create easy-to-use interfaces. OWiki, one of the tools that were implemented using the GAFFE approach, uses forms in the edit interface and infobox templates in the display interface of

MediaWiki. Forms and infobox templates are generated through the associated ontologies, and are completely customizable by editing it to change the nature, kind and mappings of the form widgets.

OWiki explicitly builds the domain ontology as the union of a page domain and a data domain. The page domain represents all the properties that have the wiki article as the subject (e.g., this article is a detailed overview, is meant for scholars, is written in English, talks about "War and Peace", is written by John Smith), while the data domain represents all the properties that have the *topic* of the wiki article as the subject (e.g., "War and Peace" is a novel, is meant for leisure reading, is written in Russian, talks about Napoleon's invasion of Russia, is written by Leo Tolstoy). While the data domain depends strictly on the topics of the wiki, the page domain can and has been described in terms of a single and shared ontology, the IFLA FRBR model (FRBR 1998), allowing for a large number of interesting assertions to be associated to the wiki page. OWiki has been used with success in a few end-user applications, with good users' feedback (Bolognini et al. 2009, Di Iorio et al. 2010).

This paper is structured as follows. Section 2 introduces the GAFFE model. In Section 3 the OWiki application is described in detail, in particular the domain ontology, the form ontology and the page ontology. In Section 4 we discuss a few use cases that demonstrate the viability and usefulness of the OWiki system and the GAFFE approach. Section 5 discusses the implementation details of the system. Section 6 compares OWiki with related works, while in the final section we draft some conclusions and indications of future work.

## 2. The GAFFE model

The process of associating metadata to a document is a complex one. The first problem is that several metadata schemas can be used to describe the same resource. Some of them are almost equivalent, others are characterized by individual features. For instance, all metadata schemas for describing bibliographic documents are expected to include information about the "author", "publisher" and "year". At the same time, a schema describing Ph.D. theses needs to include similar information (for instance, "author" and "year") and more domain-specific data such as "id number", "supervisor" and "discipline". The choice of the most suitable schema depends on two main factors: (i) the nature of the document and (ii) the applications (and users) that the document is meant to be processed by.

Still, choosing the appropriate metadata schema is not enough. It is just as important that the interface for authoring metadata is intuitive and usable. In fact, a good schema not supported by a good editing interface risks to be useless: authors would often decide not to insert metadata, considering it a pointless, time-consuming and postponable task.

In this paper we report the efforts to design and implement a flexible and user-friendly metadata editor. We identify the following four main features of such an editor:

- **Genericity**: the editor should support any metadata schema in a flexible way. The more the editor is flexible and adaptable, in fact, the more it is valuable for different contexts and users. Such a requirement has been proposed for several other projects too. For instance, the GEM framework (Gores et al 2009) is a format-independent editor for graph-based data: users are allowed to create abstract objects, and relations among them, that are eventually encoded in a specific data format. A generic editor reduces considerably the costs of supporting new schemas, following the evolution of existing standards and integrating heterogeneous resources.
- **Customizability**: instead of being a static form – strictly dependent on a given schema – the editor should be customizable for users' needs and preferences. That helps administrators to deliver to the final users more friendly and intuitive interfaces, well integrated with other editing environments. In particular, although often a requirement exists to adhere to a given metadata schema (say, Dublin Core), local rules may actually be requested so as to restrict the set of allowable values to a subset to the ones allowed by the metadata schema. In these situations, being able to provide a custom form that only allows the specification of the locally allowed values is important to maintain flexibility and usability.
- **Proactivity**: the editor should provide users with pre-filled form fields, suggestions, default values, access to environment variables, and so on. These facilities simplify the authoring process, as they reduce the number of actions users have to perform. Moreover, they reduce the possibility users miss some information or decide not to insert metadata at all. A proactive editor thus will heuristically propose values and learn by the users' modifications what are the preferred, most frequent and most likely values to propose next.
- **Validation**: the editor should apply validation mechanisms to check the correctness of the values. The ability to validate metadata has been defined as the "first tier of quality indicators" (Bruce and Hillmann 2004). In fact invalid metadata are misleading or useless at all. Validating metadata while they are being created improves the overall quality of the documents and does not require further consistency checks, which might be difficult or even impossible once the provider of metadata value has completed the job.

A solid approach to flexible interfaces consists of adopting the "Model-View-Controller" (MVC) pattern (Gamma et al. 1994), as developed in the software engineering community. This pattern implements a clear separation between the business logic of an application and the user interface for visualizing/editing data: it allows designers to generate applications whose interfaces can be easily modified without affecting the model and vice versa.

Discussing the benefits of MVC is out of the scope of this work, but what is important is exploring how this pattern can help designing a flexible and sophisticated metadata editor. In the context of metadata editors, the three components of MVC become:

- **Model**. The model corresponds to the actual metadata values as manipulated by the editor and associated to the document. Changing the metadata schema means changing the model of MVC, and this should at all times be possible, in order to obtain a schema-independent editor.
- **View**. The view is the way metadata values are shown to the users. We identify two main types of view: the edit interface and the visualization interface. The edit interface has to be a rich graphical interface, with a large number of graphical widgets to specify metadata values according to their type and expected values. The more widgets are sophisticated and well-structured, the more easily users can create metadata. The visualization interface shows the current document metadata values only, without changing the internal model. The visualization happens through deactivated form fields, but also with plain textual or tabular visualization. Since model and view are separated, we can assign multiple views to the same metadata schema, allowing for customized interfaces over the same metadata model, each tailored to roles, personal preferences and local policies of the intended users.
- **Controller**. The controller is the component in charge of managing the interaction between the users and the application. It has to store the values provided by users into the model, and ultimately embed them in the documents according to the local syntax. Moreover, it is expected to run a pre-processing phase to provide default values to relevant form fields and a post-processing one to validate metadata values as provided by the user. The controller thus handles all input events and notifies the model the users' actions that generate changes in the model itself.

Starting from the previous considerations about MVC, we propose to use ontologies to address both the issues of associating structured content to electronic documents, in order to help authors to create effective relationships between metadata instances and their schemas, and to define formal visualization for these data.

Our approach consists of two steps: first, creating ontological descriptions of the domain and the interfaces to manipulate metadata and subsequently, transforming those descriptions into actual interfaces shown to the users. We have developed the model *GAFFE* (Generator of Advanced Forms and Friendly Editors). GAFFE makes it possible to build customizable metadata editors that allow users to decorate a document with metadata, following any scheme as expressed through an OWL ontology. More precisely, GAFFE uses two different ontologies:

- the *domain* ontology represents the conceptual model of the metadata. Since this ontology is unconstrained, users can adopt any custom metadata schema without any restriction, as long as it is expressible in OWL;
- the *GUI* ontology specifies the classes and properties of widgets and form elements of a graphical user interfaces, as well as the mapping between interface widgets and properties of the documents;

The *instance* document is an instantiation of the GUI ontology describing an actual interface, as generated by associating domain elements to graphical widgets and by customizing the final appearance of each item.

## 3. OWiki: ontology-driven generation of templates and forms for semantic wikis

The general GAFFE model has been instantiated in OWiki, an extension of MediaWiki that supports users in creating and editing semantic data. The basic idea of OWiki is to exploit ontologies and MediaWiki editing/viewing facilities to simplify the process of authoring semantic wiki content.

In particular, OWiki exploits MediaWiki templates, infoboxes and forms. A *template* is set of pair *key-value,* edited as a record and usually formatted as a table in the final wiki page. Templates are particularly useful to store structured information: very easy to edit, disconnected from the final formatting of a page, very easy to search, and so on. Templates are defined in special pages that can be referenced from other pages. These pages include fragments with the same structure of the template but filled with instance data. The template-based component of a page is also called *infobox*.

OWiki exploits ontologies to represent the (semantic) knowledge-base of a wiki and templates to display and edit that ontology through the wiki itself. The integration and interaction between ontologies and templates can be summarized in two points:

- each class of the ontology is associated to a template-page. Each property is mapped into a key of the infobox.
- each instance of that class is represented by a page associated to that template. Each line in the infobox then contains the value of a property for that instance. Data properties are displayed as simple text while object properties are displayed as links to other pages (representing other instances of the ontology).
- each page can include extracts from other pages. These extracts are described by paragraphs and/or entire sections of the page in which they are included. Moreover, extract-related infoboxs are also added – i.e., the infoboxes made by using data concerning the respective ontological entities each extract refers to. So, through this inclusion, it is possible to have

more than one infobox for each page: one referring to the entire page itself, and others relating to any imported extract in it[1].

OWiki templates are actually transparent to users. In fact, each template is associated to a form that allows users to create and edit the relative instances. Users do not modify the templates *directly* but they only access specialized form fields.

The crucial point is that even forms are generated on-the-fly from ontological data. According to the GAFFE model, in fact, OWiki also includes a GUI ontology describing widgets and interface elements. The concepts and relations of the input ontology are mapped into form elements that are delivered to the final user.

During the installation phase OWiki creates a basic set of forms by merging the domain ontology with the GUI one. At the editing phase, the system shows a very basic form and saves it as a special page (template). This page can then be organized as a new form by adding dynamic behaviours, moving buttons, changing the field order and so on.

Before describing the internal architecture of the system, it is worth spending few more words about the way OWiki uses ontologies. In fact, the extensive usage of ontologies makes it possible (i) to make OWiki independent on the domain it is used for, (ii) to easily customize forms and templates, and (iii) to fully describe the evolution of a wiki page and its semantic content.

## *1.1 Using ontologies to model the domain*

In OWiki the entire *domain of discourse* – i.e., all the topics each page talks about – is handled by using an OWL ontology, called *domain ontology*. Two different kinds of classes exist in this ontology: those – *page-domain* classes – strictly related to articles and pages visualized by the wiki, and other ones – *data-domain* classes – that define additional data around the former ones.

[1] This is still an experimental feature and, consequently, it is not added yet to the OWiki demo.

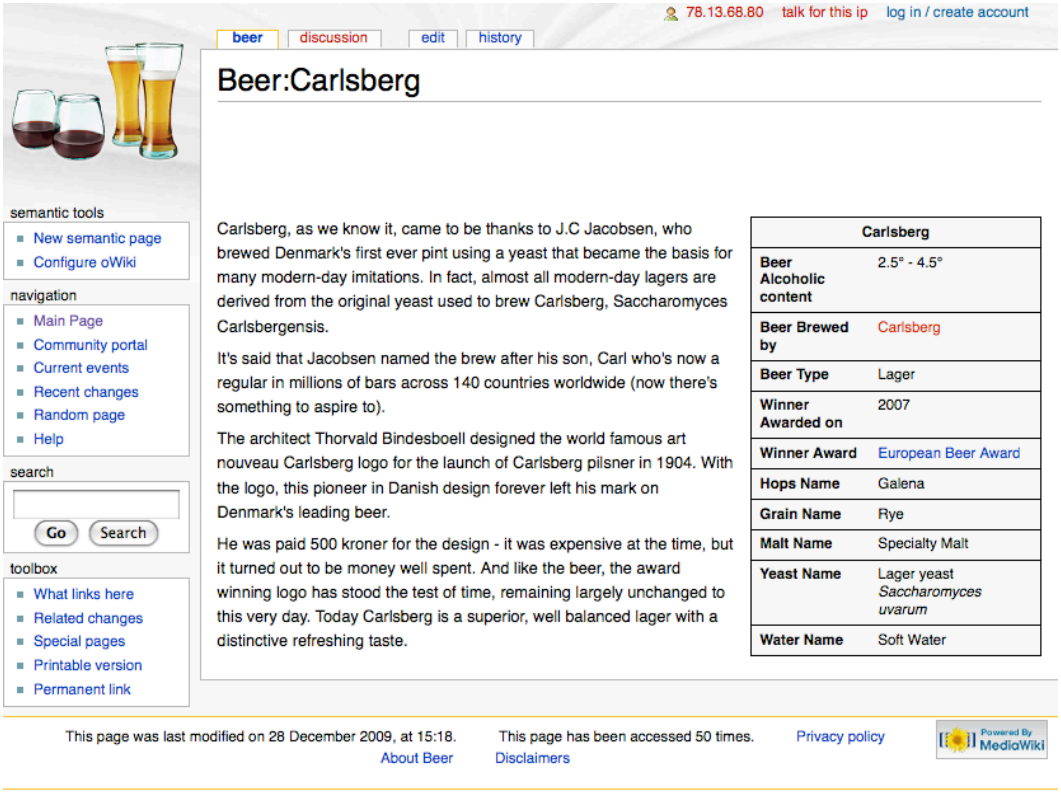


Figure 1. An OWiki page example about the Carlsberg beer.

As we have previously said, each *page-domain* individual results in a wiki page containing text content (the content is now stored in the MediaWiki internal database) and all semantic data directly related to that individual. Figure 1 shows a page about a particular beer[2] that contains a text description of it in the central page area, while in the right box there are all metadata about this particular beer.

While some metadata, such as "Beer Alcoholic content" or "Beer Brewed by", are proper to any beer directly, because they are defined by OWL data or object properties having the class *Beer* as domain, that is not true for others metadata, such as "Winner Award" and "Winner Awarded on". In fact, those properties are handled using the *data-domain* class *Awarding* that represents an event, concerning a particular prize, in which a beer participated to in a specific time. The model behind the value of such properties for the beer shown in Figure 1 is explained in the following excerpt (in Turtle syntax (Becket and Berners-Lee 2008)):

```
:carlsberg a :Beer ; :hasAwarding :awardingEuropean2007 .

:awardingEuropean2007 a :Awarding ;
      :hasAward :europeanBeerAward ; :hasYear "2007" .
```

2 The entire ontology is available at "http://owiki.web.cs.unibo.it/owl/carlsberg.owl".

The values shown in the Carlsberg page are not directly extracted from the Carlsberg ontological individual: they are taken from the awarding event the Carlsberg beer participated to.

Even if they are not directly represented as wiki pages, OWiki uses *data-domain* individuals to enrich even more the metadata of *page-domain* individuals. This enrichment needs to retrieve related data from non-page-domain-individuals making at least two steps on the RDF graph represented by the model. We call *property flattening* the visualization of those *data-domain* property values into a *page-domain* individual[3].

Let us to introduce the property flattening operation in a more formal way, starting by defining what we mean with *property path*. A property path $C_0$~$C_n$ is defined as follows:

$E_0\ p_0\ E_1\ p_1\ E_2\ p_2\ E_3\ \ldots\ E_i\ p_i\ E_{i+1}\ \ldots\ E_{n-1}\ p_{n-1}\ E_n$

where:

- Each $E_i$, different from $E_n$, is a class;
- Each $p_i$, different from $p_{n-1}$, is an object property;
- Whether $p_{n-1}$ is an object property $E_n$ is a class, otherwise $p_{n-1}$ is a data property and, consequently, $E_n$ is a literal;
- $E_i$ represents the domain of $p_i$ while $E_{i+1}$ is its range;
- $E_0$ is representable as OWiki page;
- All the instances belonging to any $E_i$, different from $E_0$ and $E_n$, are not representable as OWiki pages.

Then, we define the *flattening property operation* as the process that allows to visualize all the values of the properties in any existing property path $C_0$~$C_n$ by using the OWiki/MediaWiki infobox related to $C_0$.

## *1.2 Using ontologies to model the interface*

OWiki exploits ontologies to also model end-user interfaces, according to the basic GAFFE model. In particular, the system includes a GUI ontology identifying all the components of web forms. The system instantiates and merges that ontology with with the domain one in order to generate the final forms. The definitive version of the GUI ontology is still under development but the core concepts and relations are stable and already tested in the current prototype.

The GUI ontology defines two types of graphical elements: *controllers* and *panels*. Panels are containers for other elements (that can be panels, in turn) used to organize the overall interface, while controllers are single widgets allowing users to actually fill metadata.

The main class of the ontology is *OWikiForm.* Instances of this class will be used to generate each form associated to each wiki page. Each instance will in fact contain either graphical

3 A free demo version of this application is available at “http://owiki.web.cs.unibo.it”.

elements or property values from the domain ontology. *OWikiForms* can contain *simple* or *complex* types of controllers. Simple types will be associated to data properties in the domain ontology, while complex type will be associated to object properties.

Simple types model the basic form elements (*Textfield*, *ComboBox*, *CheckBox* e *RadioButton*) while complex types model constructs useful for mapping the GUI ontology to the domain one. In fact, there are two complex types: *ConnectField* and *ObjectContainer*. *ConnectField* model links to another wiki document. This will be finally used to provide users with auto-completion operations on corresponding form fields: when the user will fill this field, the system will suggest a set of linked documents she/he can choose from (or create a link to a completely new resource). These links are in fact derived from the relations in the domain input ontology. *ObjectContainers* are widgets that includes properties of a class non-directly linked to the one defining a particular page, including into documents data about other (related) subjects. This class implement what we illustrated previously as *property flattening*.

## *1.3 Using ontologies to model the evolution of a wiki page*

OWiki relies on a strong ontological model for capturing the evolution of wiki content as well. In particular, the system adopts FRBR (*Functional Requirements for Bibliographic Record*)(FRBR 1998), an entity/relationship model proposed by the *International Federation of Library Association* (IFLA).

FRBR is a general model for describing the evolution of *any* document. It works for both physical and digital resources and proved to be very flexible and powerful. One of the most important aspect of FRBR is the fact that it is not tied up with a particular metadata schema or implementation.

Details of FRBR are out of the scope of this paper. However it is interesting to give a brief description of its basic (ontological) concepts and the way they are applied in the OWiki framework.

FRBR describes all documents starting from a four different and correlated points of view. Let us take into consideration the article about the Carlsberg beer[4] ("Beer:Carlsberg") previously introduced. The four different FRBR levels for this document are described as follows:

- *Item*. Any **physical copy** of "Beer:Carlsberg" in any possible format (digital or not) – e.g., the HTML copy download by my browser or the PDF copy I printed;
- *Manifestation*. A particular **format** in which "Beer:Carlsberg" is stored, for instance HTML and PDF. A *Manifestation* is exemplified by one or more *Items* – the document as expressed in a particular format that strictly represents all the other copies downloaded/printed from it;

[4] http://owiki.web.cs.unibo.it/mediawiki/index.php?title=Beer:Carlsberg.

- *Expression*. Any text representing the actual **content** – generally, what the author wrote in terms of textual content and infobox metadata – of the article. In particular, each *Expression* is represented by a particular OWiki/MediaWiki article revision. An *Expression* is embodied in one or more *Manifestations*, as the same author's revision (e.g., the revision dated "14:46, 14 December 2009") can be expressed in different format (HTML, PDF, etc.);
- *Work*. The higher level **abstraction** of the article, i.e., the ideas in the author's head for the article. A *Work* is realized through one or more *Expressions* – for example, the revisions "14:46, 14 December 2009", "11:21, 29 April 2010", etc.

In Figure 2 we graphically summarize the four FRBR levels for "Beer:Carlsberg".

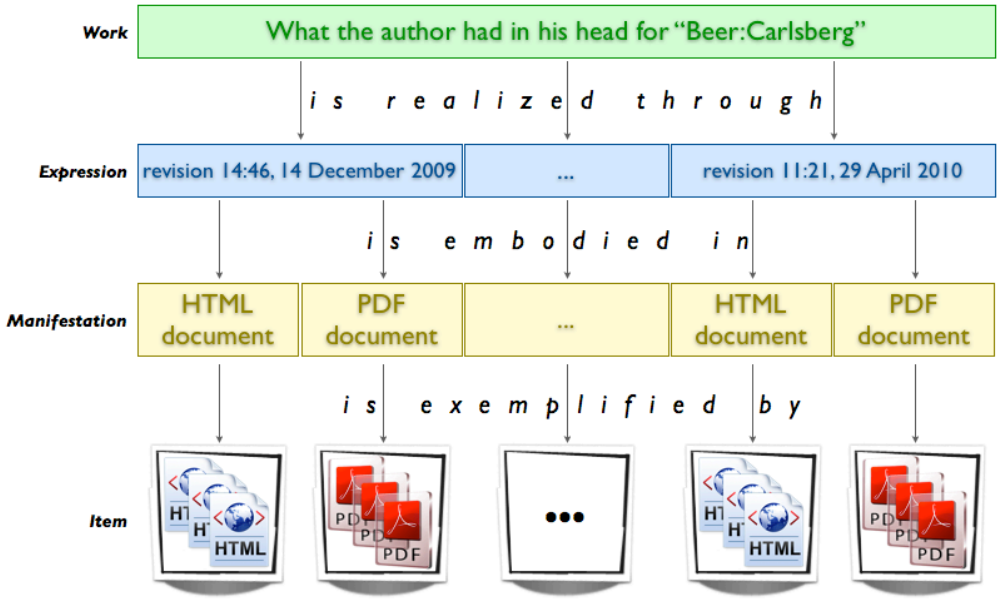


Figure 2. The FRBR levels for the work the "Beer:Carlsberg" article.

OWiki users such document classification to store metadata about any wiki page and to track its evolution. Each page, at any point of its life, represents not only a *page-domain* individual (in the ontology described in the previous section) but also a Manifestation of a certain document, i.e., the document generated starting from the *page-domain* individual itself.

Metadata are handled and shown within wiki pages – that is a Manifestation. The OWiki visualization and form editor allows users respectively to show and modify directly the Expression. These modifications are also propagated to the Work level. Such a process implements what we previously called *property flattening*. For example, the page about Carlsberg is enriched with all the FRBR properties relative to the Manifestation class, and also with those concerning the relative Expression and Work by *flattening* them into the Manifestation domain.

## 4. Studying OWiki through a use-case

The main goal of OWiki is to simplify the creation of semantic data *through and within* wikis. The complexity of such metadata authoring process, in fact, is hidden behind the application in order to not force users to learn new interfaces and tools. They can easily create semantic data by exploiting forms and templates that are automatically generated from ontological data.

In this section we explain with much details this generation process, clarifying how ontological data are converted into (customized) interfaces. Basically, the overall OWiki process consists of three steps:

- ontology import and forms generation;
- forms customization;
- templates and data generation.

### *1.4 From ontologies to forms*

The first step consists of importing the input domain ontology into the wiki. Let us consider a sample application we will discuss throughout the following sections: an OWiki installation describing beers, breweries, ingredients, etc. Figure 3 shows some classes of a domain ontology suitable for such an application.

Classes and properties are mapped into wiki pages following the schema briefly described in the previous section: each concept is mapped into a page and properties are expressed through templates. In particular, data properties become lines of templates infoboxes and object properties becomes typed links.

The OWiki conversion process also produce forms to edit the ontological content. Forms are dynamically built by analysing the class properties of the imported ontology and by mapping each property in the proper element of the GUI interface.

In the example, the class *Beer* defines three properties: *name*, *type* and *alcohol content*. According to the type of these properties OWiki generates text fields or radio buttons. The default element is a text field that allows any type of value. Since in the input ontology the only possible values of the property *beer type* are *Ale*, *Lager* and *Pilsner*, the system add to the form a RadioButton element specifying those values.

For object properties OWiki chooses between two types of widgets according to their range: whether the range class is a descendant of the *oWiki* class, the system adds a *ConnectField* to the form; otherwise it adds an *ObjectContainer*. Since the *Beer* class has the object property *brewed by* with the *Brewery* class specified as range, for example, the system add to the form a widget that allows to include a link to a corresponding brewery page. This widget will also provide auto-completion features built on top of the relations expressed in the input ontology.

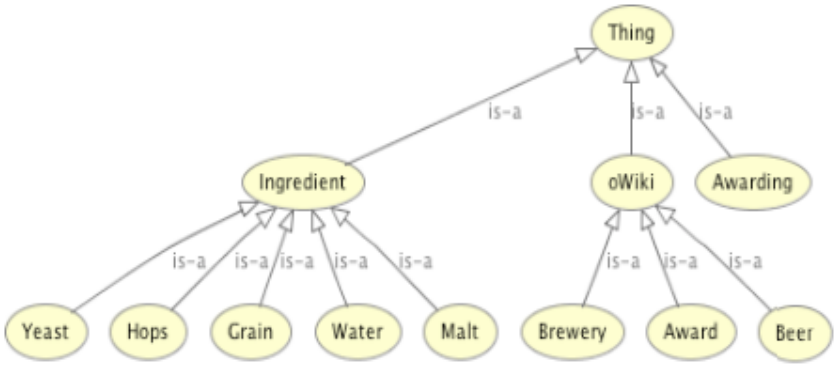

Figure 3. A graphical representation of the OWiki domain ontology about beers.

## 1.5 Forms customization and filling

OWiki also include a configuration interface that allows users to set the mapping between the input ontologies, and to configure the overall organization of the form and its formatting properties. Intuitive, easy to use and handy forms can be created without mastering the HTML source code directly.

The first time a user edits a page, OWiki shows a basic form. The author can then organize a new form adding dynamic behaviours, moving buttons, changing fields order and so on. The following picture shows a simple example of a customized forms: while the original form only listed a set of plain text-fields, this one is organized in panels and uses radio-buttons, images and dynamic widgets. An example of customized form is shown in Figure 4.

Customization can happen at different level. The user can change colour, font, background of the text to increase the appeal and impact of the form; she/he can change the position and the order of the elements to increase the importance of certain data; she/he can change the optionality of the elements, their default values, and so on.

The current implementation requires users to customize forms by editing an XML configuration file, through the wiki itself. Even if such an approach is not optimal, the internal architecture of the system relies on a strong distinction between the declarative description of the form (through the GUI ontology) and its actual delivery. That makes possible to implement a user-friendly and graphic environment to create and customize forms. One of the our future activities is the implementation of such an editor within the OWiki framework.

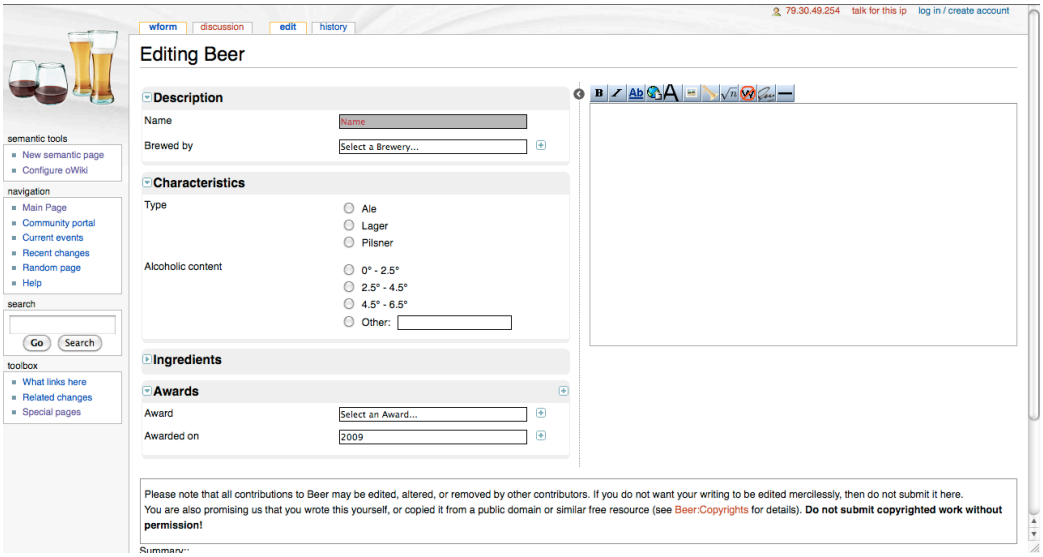


Figure 4. A customized form generated by OWiki.

## 1.6 From semantic data to templates and views

Automatically-generated forms are finally exploited by the wiki users to actually write the semantic data. As described in the previous section, data are stored as templates and templates are manipulated by forms *in a transparent manner*.

Let us consider again the *Beer* class of the example. OWiki generates a form to create instances of that classes showing three main components:

- a text field to insert the name of the beer
- a radio-button to select the type of the beer. Values in the radio button are directly extracted from the domain ontology.
- a text field to insert the brewery, that suggests breweries by exploiting information in the domain ontology.

These components can even be organized in multiple panels. Once the user fills the form OWiki saves a template with the proper information. Infobox templates, in fact, are used to display metadata and to cluster information about the same document.

Each infobox line correspond to a field in the form that, in turn, corresponds to a parameter and its value in the domain ontology. As expected, the data property of a class are displayed as simple text while the object property are displayed as links to other documents.

The page corresponding to the Carlberg beer in the example, that is an instance of the class *Beer* and has been edited via the corresponding form, will contain the following (partial) infobox:

```
{{Infobox Beer |
hasoWikiNamePage=Carlsberg |
Beer_brewedBy=[[Brewery:Carlsberg|Carlsberg]] |
Beer_beerType=Lager
```

```
|Beer_hasAlcoholicContent=2.5° - 4.5°
|Hops_hasName=Galena | … . … …. … }}
```

Notice that the property *Beer_brewedBy* contains a link to the page *Carlsberg* that is now an instance of the *Brewery* class. Relations in the input ontology are then mapped into the links between pages. And the *Carlsberg* instance follows the same approach, being it described by the infobox:

```
{{Infobox Brewery |
hasoWikiNamePage=Carlsberg |
Brewery_hasAddress=Valby 11 DK - 2500, Copenhagen |
Brewery_brews=[[Beer:Carlsberg|Carlsberg]] }}
```

## *1.7 Consistency and content evolution*

Some final considerations are worth about the consistency of OWiki. First of all, note that OWiki forms only work on the instances of the underlying ontology, without impacting classes and relations among them. The consequence is that, assuming that users do not corrupt infoboxes (that are anyway available in the source code of a wiki page), the overall ontology keeps being consistent. The OWiki instance is in fact consistent by construction with the domain and the GUI ontology and it is populated via forms in a controlled way.

Note also that OWiki is strongly connected with the MediaWiki content manager since both ontological data and forms are stored within wiki pages, in the MediaWiki database itself. That means that all facilities for navigating the page history, showing changes between pages, going back to previous versions are available. Not only the modifications to the content but also the modifications to the domain ontology (if reloaded) and the forms are recorded in the history.

The structure of each form, for instance, is described within a wiki page (in XML syntax). Changes to that structure generate new versions of the page, which users can surf, diff with the newest one and roll back. The effects to the wiki of changes to the domain ontology are also worth discussing. Four changes are handled by OWiki: adding/removing a class and adding/removing a property. The creation of a new class is reflected by the creation of a new category page, a new pair of form/template to edit/view instances. The deletion of a class only changes the properties of existing page instances: templates and forms will not be used anymore, as those instances are not relevant in the ontology anymore. The definition of a new property for a class generates a new field for the corresponding forms (that will be used only for new instances) and, symmetrically, the deletion of a property remove the corresponding element in the form (that, once again, will be used only for new instances).

## 5. Implementation details of OWiki

OWiki is an integrated framework composed by three modules, delivered with different technologies:

- a *MediaWiki extension*. It is a module integrated in MediaWiki written in PHP that adds OWiki facilities;
- an *Ontology manager*. It is a Java web service that processes OWL ontologies to produce forms for editing metadata. This manager uses internally both Jena API[5] and OWLAPI[6];
- an *Ajax-based interface*: a client-side module that allows users to actually insert data through the forms generated by the OWiki engine.

The PHP OWiki module follows the same architecture of any MediaWiki extension: some scripts and methods are override to provide new features. In particular, the module implements a revised editor that initializes OWiki environment variables, sets the communication with the client and sets data necessary to store forms in the MediaWiki database without interfering with existing data.

To manipulate ontologies, OWiki implements a web service that uses internally the Jena API. Jena is integrated with the Pellet reasoner (version 2.0.0-rc5[7]). This web-service actually generates templates from the ontological data, that are later sent to the PHP module and stored in the right place of the MediaWiki installation.

The connection between the PHP and Java modules, and the core of the overall framework is the OWiki client. The client is a javascript application, based on Mootools[8], is charge of actually generating and delivering forms. It is strongly based on the Model-View-Controller (MVC) pattern mentioned and its internal architecture can be divided in four layers, as shown in Figure 5.

The four layers interact among them and with the OWiki server to provide users with sophisticated functionalities:

- *The Connection Layer* manages the overall environment, the initialization phase and the communication between all other layers.
- *The Model Layer* (Model of MVC) manages the data to be displayed on the page. It is composed by a factory that creates wrappers for each type of data and instantiates data from the ontology.
- *The LookAndFeel* (View of MVC) manages the final representation of the form. It is a set of components of different types (basic elements, complex elements, manipulators, decorators,

5 "http://jena.sourceforge.net".
6 "http://owlapi.sourceforge.net".
7 "http://pellet.owldl.com".
8 "http://mootools.net/".

session managers) implementing well-defined interfaces that make the layer integrable with other Javascript applications.

- *The Interaction Layer* (Controller of MVC) implements the logic of the application, the communication with the web-service, the generation of semantic data and the end-user interaction.

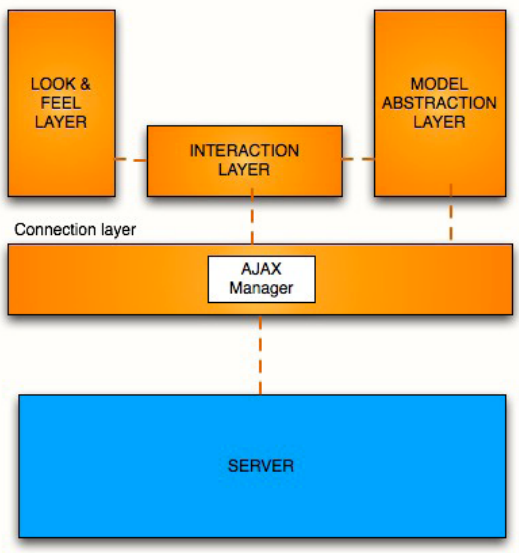


Figure 5. The OWiki client engine.

## 6. Related Works

The integration and interaction between ontologies and wikis is a hot research topic. Several approaches to semantic wikis have been developed to bring together the benefits of the free editing philosophy of wikis and ontological data. Semantic wikis can be organized into two main categories according to their connections with the ontologies: "wikis for ontologies" and "ontologies for wikis" (Buffa et al. 2008).

In the first case, the wiki is used as a serialization of the ontology: each concept is mapped into a page and typed links are used to represent object properties. Such a model – initially proposed in the ontology for MediaWiki articles, called WikiOnt (Harth et al. 2005) – has been adopted by most semantic wikis. SemanticMediaWiki (Volkel et al 2006) is undoubtedly the most relevant one. It provides users with an intuitive syntax to embed semantics, i.e. RDF statements, within the markup of a page. SemanticMediaWiki allows users to freely edit the content without any limitation. The more the information is correctly encoded the more semantic data are available, but no constraint is imposed over the authoring process. SemanticMediaWiki has been developed with the idea of creating a machine-readable version of Wikipedia, to better exploit that huge amount of information and the competencies and enthusiasm of its community. The original term Wikitology (Klein et al 2005) summarized very well the potentialities of such approach. The

DBPedia project (Auer et al 2007) is also worth mentioning, being the most recent effort in translating the Wikipedia content into RDF.

Although they are both extensions of MediaWiki, there is a very important difference between SemanticMediaWiki and OWiki in terms of ontological expressivity. While SWM and most other wikis only create assertions whose subject is represented by the subject of the wiki page containing that assertion, OWiki allows users to associate sub-forms to classes of the ontology and to retrieve and set properties of those classes in an indirect and transparent manner (see the "properties flattening" process in section 3.1 for more details).

One of the main obstacles to the realization of Wikitology, DBPedia and similar projects is certainly the difficulty in creating semantic content. Although the syntax is very simple, in fact, authors still have to learn some new markup and above all to manually write correct statements. SemanticForms (Koren 2008) is an extension of SemanticMediaWiki that addresses such issue by allowing users to create semantic content via pre-defined forms. SemanticForms generates forms from templates, whose fragments and data have been previously typed. The generation process exploits an embedded mapping between each datatype and each type of field (radio-buttons, checkboxes, textareas, etc.). Users do not need to manually write statements anymore as they are only required to fill HTML forms.

There are a lot of evident similarities between SemanticForms and OWiki, since they both are MediaWiki extensions that generate forms automatically and help users to annotate pages more easily. On the other hand, some relevant differences are worth emphasizing. First of all, both templates and forms generated by OWiki rely on a stronger ontological data-model since they are directly derived from classes and properties of the input (domain) ontology. Thus, OWiki forms can exploit information stored in the input ontology – such as the cardinality of a relation or the set possible values of a property for a specific class – that are neglected by SemanticForms. Note also that the process of "properties flattening" makes it possible to enrich OWiki forms with data that are not captured by the SemanticForms generation process. Furthermore the set of oWIKI widgets and, above all, the mapping rules between ontological data and widgets is more extended and customizable.

The difficulties in generating SemanticMediaWiki data have been mitigated by an ad-hoc importer that allows the creation of multiple articles from an input OWL ontology (Vrandecic and Krotzsch 2006). This tool uses a PHP API for managing OWL and automatically creates wiki content according to the basic MediaWiki model: each concept is mapped into a page and each property into a typed link. Some checks about the consistency of the ontology and the duplication of non-relevant data are also performed. The domain ontology import in OWiki is very similar to this process: while the SMW Importer maps all classes into pages, however, OWiki selects some

classes/properties to be “flattened” and expressed in other pages. Such a conversion is controlled in the input ontology itself and gives users more control and flexibility.

Other wikis provides users with mixed interfaces for creating semantic data. MaknaWiki (Dello et al. 2006) is a JSP wiki-clone that allows users to embed semantic statements or to fill HTML forms for querying and adding data to the ontology represented by the wiki. These forms provide general tools for aided navigation of the semantic data, do not depend on the domain of the wiki and their structure is hard-coded in the system. Rhizome (Souzis 2005) provides friendly interfaces and textareas where users can write statements directly. It relies on ZML (a textual syntax serializable into XML), a generic language to express semi-structured data, and an engine to apply rules for intermixing semantics and free texts. A novel solution is provided by AceWiki (Bao et al. 2009) and CNL-approach (De Coi et al. 2009). AceWiki is a semantic wiki that allows users to write ontological statements by simply writing sentences in the ACE (Attempo Controlled English) language. The system includes a predictive authoring tool that suggests options to the users and autocompletes fields consistently to the ontology represented by the wiki. The same editor can be used to extend the ontology by creating new classes, instances and relations. Bao et al. (2009) proposed a similar approach for SemanticMediaWiki, as they designed a CNL (Controlled Natural Language) interface able to convert sentences written in multiple languages into semantic data. Although very intuitive and expressive, such tools (as well as Makna and Rhizome) still require users to master directly semantic data while OWiki and form-based tools hide such complexity.

The extensive usage of OWL, templates and forms makes the integration between wikis and ontologies a concrete possibility. In fact, some researchers have also proposed SMW-mOWL, a “meta-model extension of SemanticMediaWiki”. SMW-mOWL is a way to encode an entire OWL ontology into a wiki by exploiting “semantic templates”. Basically it consists of storing ontology elements as template instances: classes, anonymous classes, properties, restrictions, individuals are all represented as templates, with a direct mapping between SMW-mOWL and OWL-AS. SMW-mOWL improves the ontological expressiveness of SemanticMediaWiki. Moreover, such an approach makes it easier to edit ontologies within the wiki itself as templates can be edited using the form-based interfaces of the SemanticForms extension (Koren 2008). Compared to OWiki, the SMW-mOWL approach is yet more expressive but conceptually more difficult and targeted to expert users. Users have great control over the ontology but they are still required to directly master properties, relations, restrictions and so on. OWiki average users, on the other hand, populate the ontology in a fully transparent manner without even knowing that the content is built on top of ontological data.

The second category of semantic wikis, based on the principle of “ontologies for wikis”, includes all those wikis that are actually built on top of ontological foundations. The idea is to

exploit ontologies to create and maintain consistent semantic data within a wiki so that sophisticated analysis, queries and classifications can be performed on its content.

IkeWiki (Schaffert 2006) was one of the first wikis to adopt this approach. Its deployment starts by loading an OWL ontology into the system that is automatically translated into a set of wiki pages and typed links. Multiple interfaces are provided to the users for editing the plain wiki content, adding new metadata or tagging pages. IkeWiki strongly relies on Semantic Web technologies: it even includes a Jena OWL repository and a SPARQL engine used for navigation, queries and display of the semantic content of the wiki. Similarly, OntOWiki (Auer et al. 2006) is a complete ontology editor. It relies on a strong distinction between the ontological back-end of the system and a user-friendly interface. Data are natively stored as OWL/RDF statements that are dynamically rendered into the final HTML wiki pages. OntOWiki provides users with multiple views of the same content: (1) ontological data can be navigated by listing classes, individuals, properties, etc., (2) domain-specific views can be added as plugins (for instance, a MapView of geographical data can be dynamically mashed-up form GoogleMaps) and (3) editing-views are natively available in the system. The "editing views" of OntOWiki are particularly interesting in the OWiki context. In fact, the system provides a library of interface components for data editing, called *widgets*, that allow users to manipulate the ontological data stored in the system. The OntOWiki widgets are fully configurable although the configuration process is still difficult and targeted to expert users. Both IkeWiki and OntOWiki, as well as many others wiki ontology editors, require *all* users to master Semantic Web technologies and to explicitly deal with classes, statements, URIs, namespaces, and so on. OWiki only requires such skills to domain experts that can even work outside the wiki itself, while average users populate the ontology via intuitive HTML forms.

SweetWiki (Buffa et al. 2008) implements a user-friendly ontology tool designed for both expert and non-expert users. Two aspects characterize the system: the strong connection with the ontologies and the provision of Ajax-based interfaces for editing content and metadata. SweetWiki defines a "Wiki Object Model", i.e. an ontology describing the wiki structure. Concepts like "document", "page", "link", "version", "attachment" are all codified in an OWL file that is accessed and manipulated through the wiki itself. These concepts are made explicit in SweetWiki, although they are usually hard-coded in most semantic wikis. SweetWiki also allows users to import external ontologies and to access and manipulate those ontologies through ad-hoc interfaces (similar to those provided by the above mentioned full ontology editors). Finally, the system provides "assisted social tagging" facilities: users can add metadata to any page and can put pages in relation. These metadata values form a folksonomy that, on the one hand, is freely editable by users and, on the other, is built on top of ontological data. The interface for tagging, in fact, suggests consistent

metadata by exploiting SPARQL queries and autocompletion features. Though such aspect is similar to OWiki, SweetWiki does not focus on *consistent* ontology population – as OWiki does – but on aiding users in annotating pages, organizing and sharing their tags.

Finally, UFOWiki (Passant and Laublet 2008) is another project that aims at integrating wikis, ontologies and forms. UFOWiki is a wiki farm, i.e. a server that allows users to setup and deploy semantic wikis. The overall content is stored in a centralized repository as RDF triples that express both the actual content of each page and its metadata. Multiple wikis are deployed by the same farm, so that they can share (ontological) data in a distributed environment. This is a very interesting feature, not supported by OWiki, that makes the system very flexible and powerful. It is connected to an architectural difference between UFOWiki and OWiki or similar systems. While OWiki is a wiki extension (that is actually designed to even work for *non-natively* semantic wikis) UFOWiki is a fully distributed framework to synchronize multiple semantic wikis. Such integration of multiple wikis, however, is an interesting research direction for OWiki too.

Multiple ontologies are used to model UFOWiki data. The SIOC(Semantically Interlinked Online Community) (Breslin et al. 2005) is used to describe wiki pages and users' actions, helped by a domain ontology that can be imported and mapped into the wiki. Users are also provided a combination of plain-text editors and forms to modify the ontology within the wiki. The UFOWiki forms are generated on-the-fly starting from the mapping between the classes and properties of the ontology and the fields and types of the form. The wiki administrators are in charge of setting this mapping through a graphical and intuitive interface based on Drupal and Flexinode (Chaffer 2004). This is another interesting difference with OWiki: although UFOWiki interfaces are very friendly, administrators are required to manually configure the mapping and the components of each form. The start-up phase of OWiki, on the other hand, is more sophisticated and just few clicks are enough to generate complex forms from ontological data.

The UFOWiki ontological expressiveness is a final aspect worth mentioning. While most other wikis only create assertions whose subject is represented by the subject of the wiki page containing that assertion, UFOWiki allows users to associate sub-forms to classes of the ontology and to handle these parts as separate resources. OWiki provides similar functionalities. But while UFOWiki requires users to explicitly manipulate the instances associated to sub-forms, OWiki uses “flattened properties” and does not even require those instances to be mapped into actual pages. On the other hand, UFOWiki gives (expert) users more control and exploits a larger set of ontological relations.

In conclusion, going back to the initial distinction between “wikis for ontologies” and “ontologies for wikis”, OWiki currently belongs to the second group. The wiki knowledge base is initially built on top of the domain ontology and then populated via forms. The wiki is actually used

to populate the ontology but not build and update it (adding/removing/updating classes and properties). In the future we also plan to investigate a further integration between the wiki and the ontology – and a further integration between the textual content of a wiki page and the relative infoboxes – in order to also use OWiki as a full-fledged simplified authoring environment for ontologies.

## 7. Conclusions

This work introduced either a general methodology - the GAFFE model - or a specific tool to simplify the authoring of semantic data for wikis. In fact, OWiki instantiates the GAFFE approach in the wiki context, meeting all the requirements discussed in section 2. The system proved to be (i) *generic* as arbitrary domain ontologies can be loaded and converted into wiki pages, forms and templates, (ii) *customizable* as users are allowed to configure forms and associate different widgets to the same ontology classes and properties, (iii) *proactive* as auto-completion features (supported by an OWL reasoner) and suggestions are provided when possible and (iv) *validating* as the system checks immediately the data consistency against the domain ontology (once again, by exploiting the OWL reasoner).

The strength of the OWiki approach relies on its connection with ontologies and its capability of providing users with simple but powerful interfaces. OWiki makes it possible, on the one hand, to create consistent ontological data and, on the other, to edit these data through intuitive forms and templates. Note that the framework is designed for creating ontological content within *not-natively-semantic* wikis too. It does not require *all* users to master Semantic Web technologies *explicitly* but hides such complexity exploiting a clear distinction between domain experts, interface designers and final users.

The development of OWiki is not complete yet. The current implementation is an integrated prototype that allows users to load and manipulate ontological data regardless of their application domain. A free demo – built around the example domain of beers and breweries – is available at http://owiki.web.cs.unibo.it/. Yet, it has been already able to show its usefulness by being the underlying core to two application that have been developed in the last year, and that have been described fully in (Bolognini et al., 2009) and in (Di Iorio et al., 2010): a community wiki to build custom tourist guides and as an expanding dictionary of multidisciplinary concepts for a fruitful comparison of the scientific and humanistic mindsets.

More functionalities will be added in future releases of the system: a sophisticated interface for customizing forms, a larger set of widgets described in the GUI interface and integrable in the OWiki pages, and a more flexible importer. Particular attention will also be given to the synchronization between templates, ontologies and wiki content. While the current prototype

handles the infoboxes as separate resources, in fact, we plan to study solutions for letting users to use either plain wiki textareas or templates to edit the same semantic data. The propagation of changes from wikis to underlying ontologies is a related research area we plan to investigate.

## 8. References


Auer S., Dietzold S., Riechert T., 2006: OntOWiki - A Tool for Social, Semantic Collaboration. *Proceedings of 5th International Semantic Web Conference 2006* pp 736-749.

Auer S., Bizer C., Lehmann J., Kobilarov G., Cyganiak R., Ives Z., 2007. DBpedia: A Nucleus for aWeb of Open Data. *Proceedings of the 6th International SemanticWeb Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007)*, Busan, South Korea pp. 715-728.

Bao, J., Smart, P. R., Shadbolt, N., Braines, D. and Jones, G., 2009. A Controlled Natural Language Interface for Semantic Media Wiki. *In: 3rd Annual Conference of the International Technology Alliance (ACITA'09)*, 23rd - 24th September 2009, Maryland, USA.

Becket, D., Berners-Lee, T., 2008. Turtle - Terse RDF Triple Language. W3C Team Submission. http://www.w3.org/TeamSubmission/turtle/.

Bolognini V., Di Iorio A., Duca S., Musetti A., Peroni S. and Vitali F., 2009. Exploiting Ontologies to Deploy User-Friendly and Customized Metadata Editors, *Proceedings of the IADIS International Conference WWW/INTERNET 2009*, Rome, Italy, 19 - 22 November 2009

Breslin J.G., Harth A., Bojars U., Decker S., 2005: Towards Semantically-Interlinked Online Communities. *Proceedings of the 2nd European Semantic Web Conference (ESWC '05)*, LNCS vol. 3532, pp. 500-514, Heraklion, Greece.

Bruce, T. R., Hillmann, D. I. 2004. The continuum of metadata quality: defining, expressing, exploiting. In *Metadata in Practice*, American Library Association, Chicago, IL. 238--256.

Buffa, M., Gandon, F., Ereteo, G., Sander, P., and Faron, C. 2008. *SweetWiki: A semantic wiki*. Web Semantics 6, 1 (Feb. 2008), 84-97.

Chaffer J., Drupal Flexinode, 2004. Available at: http://drupal.org/project/flexinode.

De Coi, J. L., Fuchs, N. E., Kaljurand, K. Kuhn, T. 2009. Controlled English for Reasoning on the Semantic Web. In Francois Bry and Jan Maluszynski, editors, *Semantic Techniques for the Web — The REWERSE Perspective*, number 5500 in *Lecture Notes in Computer Science*, pages 276–308. Springer, 2009.

Dello, K., Elena Bontas Simperl Paslaru, Tolksdorf, R, 2006. Creating and using semantic web information with makna. In Max Volkel and Sebastian Schaffert, editors, *Proceedings of the First Workshop on Semantic Wikis - From Wiki To Semantics*, volume 206 of Workshop on Semantic Wikis, pages S.43–57, Budva, Montenegro, June 2006. ESWC2006.

Di Iorio A., Musetti A., Peroni S., Vitali F., 2010. Crowdsourcing semantic content: a model and two applications, *Proceedings of the 3rd International Conference on Human System Interaction (HSI 2010)*, Rseszow (PL), 13-15 May 2010.

FRBR. Functional Requirements for Bibliographic Records, Final Report / IFLA Study Group on the Functional Requirements for Bibliographic Records. – Munchen : K.G. Saur, 1998. (UBCIM Publications, New Series ; v. 19)

Gamma, E., Helm, R., Johnson R., Vlissides, J. 1994. *Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Gores, J., Jorg, T., Stumm, B, Dessloch S. 2009. GEM: A generic visualization and editing facility for heterogeneous metadata, Computer Science Research and Development – Special Issue, Volum 4 Number 3, October 2009, Springer Berlin / Heidelberg.

Harth, C., Gassert, H., O'Murchu, I., Breslin, J. G., Decker S., 2005. WikiOnt: An Ontology for Describing and Exchanging Wikipedia Articles. In *Proceedings of Wikimania 2005 – The First International Wikimedia Conference*, 2005.

Klein, B.; Höcht, C.; Decker, B., 2005. Beyond Capturing and Maintaining Software Engineering Knowledge - "Wikitology" as Shared Semantics, Workshop on Knowledge Engineering and Software Engineering, at *Conference of Artificial Intelligence 2005*, Koblenz.

Koren, Y., 2008. Semantic forms. http://www.mediawiki.org/wiki/Extension:Semantic_Forms.

Passant, A., Laublet, P., 2008. Towards an Interlinked Semantic Wiki Farm, in '*3rd Semantic Wiki Workshop (SemWiki2008) co-located with ESWC2008*'.

Schaffert S., 2006. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management . In: *1st International Workshop on Semantic Technologies in Collaborative Applications STICA 06*, Manchester, UK, June 2006.

Souzis, A. 2005. Building a Semantic Wiki. *IEEE Intelligent Systems* 20, 5 (Sep. 2005), 87-91.

Vrandecic, D. & Krötzsch, M., 2006. Reusing Ontological Background Knowledge in Semantic Wikis, in Max Völkel & Sebastian Schaffert, ed., '*Proceedings of the First Workshop on Semantic Wikis -- From Wiki To Semantics*' , *ESWC2006*.

Všlkel, M., Krštzsch, M., Vrandecic, D., Haller, H., & Studer, R. (2006). Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, WWW 2006, Edinburgh, Scotland, May 23-26, 2006.