OWiki: enabling an ontology-led creation of semantic data

Angelo Di Iorio[1], Alberto Musetti[1], Silvio Peroni[1], Fabio Vitali[1]

[1]Department of Computer Science, University of Bologna, Italy,
{diiorio, musetti, speroni, fabio}@cs.unibo.it

Abstract. While the original design of wikis was mainly focused on a completely open free-form text model, semantic wikis have since moved towards a more structured model for editing: users are driven to create ontological data in addition to text by using ad-hoc editing interfaces. This paper introduces OWiki, a framework for creating ontological content within *not-natively-semantic* wikis. Ontology-driven forms and templates are the key concepts of the system, that allows even inexpert users to create consistent semantic data with little effort. Multiple and very different instances of OWiki are presented here. The expressive power and flexibility of OWiki proved to be the right trade-off to deploy the authoring environments for such very different domains, ensuring at the same time editing freedom and semantic data consistency.

1. Introduction

The explosion of social software tools has changed the way most users access the World Wide Web. Even inexpert users can now publish their content with a few clicks and do not need to master complex technologies or to use professional tools. This content is primarily targeted to being consumed by human users, as in YouTube videos, Twitter messages, FaceBook posts and so on. The creation of *semantic* web content – available for automatic search, classification and reasoning – is much more difficult and time-consuming. Technical competencies and domain-specific knowledge, in fact, are still required in authors. The shift of the Web from a *human-understandable* to a *machine-understandable* platform, as envisioned by the Semantic Web community [Berners-Lee et al. 2001], is far from being complete.

The term "lowercase semantic web" [Munat 2004] has been coined to indicate research efforts aiming at bridging the gap between simplified authoring and semantic web data. Such lowercase semantic web is not an alternative of the uppercase "Semantic Web", but rather an intermediate step towards the same goal. While the

Semantic Web aims at bringing full-fledged reasoning capabilities to intelligent software, the lowercase "semantic web" aims at encoding semantic data that can be accessed by everyday software and, above all, can be created by unsophisticated users.
Semantic wikis play a leading role in such a scenario. Semantic wikis are enhanced wikis that allow users to decorate pages with semantic data by using simplified interfaces and/or specialized wiki syntaxes. They provide users with sophisticated searching and analysis facilities, and maintain in full the original open editing philosophy of wikis in everything but the form in which the content is created.
Many semantic wikis, though, are essentially designed for scholars and experts in semantic technologies, and are still difficult to be used by average wiki contributors with no technical expertise. Forms are often used to mitigate this issue, providing an intuitive interface that is 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 expect. In most cases, in fact, these forms use generic text fields that are not fit to the domain the wiki is used for, or that only include a limited set of interface widgets.
In this paper we introduce a methodology and a tool to simplify the process of authoring semantic web data through wikis, and we present two very different environments where that tool was delivered. The overall approach relies on ontologies and allows even inexpert users to create semantic data with little effort. The tool, called OWiki, uses Semantic Web technologies to handle the wiki knowledgebase and MediaWiki forms and templates to deliver intuitive interfaces to the final users. Forms and infobox templates are automatically generated from ontological data, and are completely customizable to change the nature, structure and constraints of the form widgets.
The paper is structured as follows: Section 2 gives an overview of the main approaches to semantic wikis; the following one, Section 3, presents the OWiki approach, focusing on its ontologies and their application in this context, while Section 4 goes into the details of a use-case and the internals of the system are discussed in Section 5. Two different instances of OWiki are presented in the last part of the paper, before drawing some conclusions.

2. Related works: Semantic Wikis and Ontologies

The integration and interaction between ontologies and wikis is a hot research topic. 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. 2006].
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 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.

Although the syntax is very simple, SemanticMediaWiki 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 – i.e. predefined structures dynamically filled by content and rendered as tables within MediaWiki articles - 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. On the other hand, these forms have a fixed structure that cannot be customized and are still difficult to be mastered by the administrators of the wiki.

The idea of the second category – "ontologies for wikis" – 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. IkeWiki strongly relies on Semantic Web technologies: it even includes a Jena OWL repository and a SPARQL engine used for navigation, search and display of the semantic content of the wiki.

UFOWiki [Passant and Laublet 2008] aims at integrating wikis, ontologies and forms too. UFOWiki is a wiki farm, i.e. a server that allows users to setup and deploy multiple 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. Users are also provided plain-text editors and forms to modify the ontology within the wiki. These forms are generated on-the-fly starting from the mapping between *classes and properties* of the ontology and *types and fields* of the form. Administrators set this mapping through a graphical and intuitive interface.

The ontological expressiveness of UFOWiki is another aspect worth remarking. 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. The result is a more fine-grained control over the ontology and its populating process.

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

OWiki is Gaffe-based [Bolognini et al. 2009] 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 creates templates to display that ontology through the wiki itself. This integration and interaction 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.

The actual mapping process is more complex and allows users to select concepts and properties to be displayed, to infer properties and their values. Further details will be provided in Section 3.1.

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. OWiki also includes a *GUI ontology* describing widgets and interface elements. The concepts and relations of the *domain 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.

3.1. Using ontologies to model the domain

The *domain of discourse* – i.e., all the topics each page talks about – is handled by an OWL ontology, called *domain ontology*, whose role is to express data that will be later transformed into the *view/edit* interfaces shown to the final users. The classes of the ontology are divided in two groups: those strictly related to articles and pages visualized by the wiki – called *page-domain* classes – and the others that define additional data around the former ones – called *data-domain* classes.
Each *page-domain* individual will be transformed into 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 (available at http://owiki.web.cs.unibo.it) 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. OWiki automatically builds this page from the ontological data of the *domain ontology*.

Fig. 1 An example page about a Beer in OWiki

This mapping process is not 'blind' (one class into one page, without any check) but performs some intermediate controls and processing in order to provide users more flexibility. In particular, three strategies adopted by OWiki are worth discussing more in detail:

1. properties flattening
2. properties inheritance by class subsumption
3. classes and properties filtering

3.1.1. Properties flattening

The process of *property flattening* consists of adding properties to *page-domain* individuals (that will be directly mapped into wiki pages) that are not specified in the class that individual belongs, but are derived from other (not page-domain)

individuals related to the current one. Let us go back to the beer example in order to explain the need and advantages of this process.
While some metadata, such as "Beer Alcoholic content" or "Beer Brewed by", are proper to any beer directly (and 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". These properties are *indirectly* related to the beer and they are (correctly) defined on other classes of the ontology. In fact, there exists a class *Awarding* that represents an event, concerning a particular prize, in which a beer participated to in a specific time. The final users, on the other hand, are probably not interested in viewing/editing a wiki page about an Award and in manipulating directly that connection. It is more immediate and intuitive to just find a "Winner Award" property in the page (or better in the template) of a given beer and to update that property through the corresponding form.
The following excerpt (in Turtle syntax) shows the information stored in the domain ontology to handle the above-mentioned example:

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

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

In fact, 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.
OWiki allows users to mark properties to be "flattened" in order to simplify the final interfaces for the users, by reducing the overall number of wiki pages, and to automatically enrich selected *page-domain* individuals. Such enrichment needs to retrieve related data from non-page-domain individuals, by navigating the RDF graph represented by the model.

3.1.2. Properties inheritance by class subsumption

When a user asks for adding or editing an individual of a particular class, OWiki shows a series of form widgets derived from all the properties *explicitly* defined for that class, i.e. those properties having it as domain. Requiring ontology designers to define explicitly all properties for all classes is time-consuming and error-prone. OWiki is also able to infer, as part of a class C, all those properties that are not directly linked by any domain assertion to C, but that can be inferred by following the super-class hierarchy starting from C itself. Those properties are derived and eventually shown in the forms.

In fact, ontology designers define classes organizing them in hierarchies through *sub-class* relationships. Of course, these subsumptions, such as Animal subsumes Person, are usually defined when a class shares some common principles or characterizations with another class – for example, the fact of having a genre, that concerns animals as well as persons. Moreover, they represent the main

mechanism to infer whether individuals of a particular class (e.g., Person) belong implicitly to a much wide and less-specific class (e.g., Animal).

By means of subsumptions, an ontology engineer is free to define shared properties in a particular high-level class without specifying them again in each relative sub-class, simply assuming that those properties are implicitly inherited from the high-level class itself. Let us to introduce an example:

```
:Animal a owl:Class .

:hasGenre a owl:DatatypeProperty
       ; rdfs:domain :Animal
       ; rdfs:range xsd:string .

:Person a owl:Class
       ; rdfs:subClassOf :Animal .
:Cat a owl:Class
       ; rdfs:subClassOf :Animal .
```

Here we are saying that either animals, cats and persons may have a genre specified. The model defines explicitly that for the former kind of individuals (i.e., Animal), while it is inferable implicitly for the latter kinds of individuals (e.g., Cat and Person) because of the subsumptions that exist among those classes.

OWiki is able to understand all those inherit properties, therefore visualizing them in the template and form of each page-domain class they are related to, and without requiring the ontology designers to explicitly defined all of them.

3.1.3. Classes and properties filtering

A complete mapping of *all* properties and *all* classes of the domain ontology into form fields (and their corresponding infoboxes) would not be very helpful for the users. The resulting interfaces, in fact, would use *in exactly the same way* either data that could be processed transparently to the users, or that are worth viewing but not editing, or that are actually editable. Consider, for instance, a wiki page about a Person: the *creation-date* is an information relevant to the system that should not be shown to the users; the *URI* of the same page, on the other hand, should be displayed in order to identify that page but should not be editable by the user; the name of the person (as well as many other properties) should be both included in the Person infobox and editable in the corresponding form.

OWiki handles all these cases by allowing users to decide (i) which classes of the domain ontology are to be converted in wiki pages, (ii) which properties of those classes are to be stored in the system but never displayed to the users, or only viewed but never edited, or fully editable. Such process is called *property filtering* and exploits OWL annotation properties - metadata about the ontology properties themselves. In fact, users are allowed to classify both object properties and data properties of the domain ontology in: *#infoboxHidden* (never displayed),

#fieldHidden (viewed in the template but not in the forms), *#fieldLocked* (viewed in the form too but not editable), *#editable* (that is the default value). The OWiki engine parses such annotations and decides how to map ontological data into actual wiki pages.

The use of these annotations contributes to speed up and simplify the view and edit of ontological data: final users, in fact, are required to only fill the required data, while the system handles all others. The distinction between viewable and editable data makes the final users' interfaces even more intuitive and clear.

3.2. Using ontologies to model the interface

OWiki exploits ontologies to also model end-user interfaces. In particular, the system includes a GUI ontology identifying all the components of web forms. The system instantiates and merges that ontology 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.

Separating the GUI ontology from the domain one has a two-fold goal: (1) generating a declarative description of the interface widgets that can be reused across multiple domains not being bounded to specific data and (2) allowing users to customize final interfaces by only changing the association between content and interface widgets. Note that the GUI ontology can be designed once for all, while the domain one requires different expertise for different application scenarios.

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. Controllers can be grouped in two classes that, as expected, correspond to the *data-properties* and *object-properties* defined in the domain ontology.

3.2.1. Simple controllers: filling data properties

Simple controllers model the basic form elements, provided to the users for inserting data-properties. Some of them are: *Textfield,* a field of text for those properties whose value is a string or a number, *Drop-down list*, a menu for selecting a value among a finite set (whose values are defined in the domain ontology), a *ComboBox*, that combines the previous two widgets, C*heckBox* and *RadioButton*, as used in HTML, and so on.

Notice that this set is very extensible, so that newer and more specific needs can be covered, and the most recent developments of client-side programming make such extension easier.

3.2.2. Complex controllers: connections among ontology entities and among wiki pages

Complex controllers reflect connections among ontology entities – *object properties* – and allow users to create links between the final wiki pages and to decide which content to be shown.

We identified three complex controllers – notice that this set is extensible too – that correspond to three different needs of the users:

1. *ConnectField* model links to another wiki page. These controllers are usually associated to the majority of the object-properties in the domain ontology (see Section 3.1 for more details about selecting classes from that ontology). When the user will edit a wiki page corresponding to a given class, *ConnectField*s will provide auto-completion features: the system will suggest a set of linked pages 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.
2. *ObjectContainers* are used for *properties flattening* (as described in Section 3.1.1). This widget allows users to edit the properties of an individual, whose corresponding page actually does not exist in the wiki, by setting that property in the form of another class including it. The overall process is transparent to the users, which actually use the same interface to modify the properties of two (or more) individuals.
3. *InclusionLists* are used to combine the information of multiple individuals into the same wiki page. If in the domain ontology there exist two individuals connected by a specific object-property (temporarily called "*includes_content_from*"), the wiki page corresponding to the first one will include the content of the wiki page corresponding to second one. When editing that page, the user won't see the whole content of the second page but just an *InclusionList* controller allowing her/him to select the document to include. Notice that such inclusion is very different from a common link, derived from a different class of object-properties, which is displayed as a link when either viewing or editing a page. These controllers proved to be very useful for creating and authoring compounded documents (even from external content) as they just require users to select the material to be included without pasting and editing it manually.

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:

1. ontology import and forms generation;
2. forms customization;
3. templates and data generation.

4.1. 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 demo installation describing beers, breweries, ingredients, etc. Figure 2 shows some classes of a domain ontology suitable for such an application.

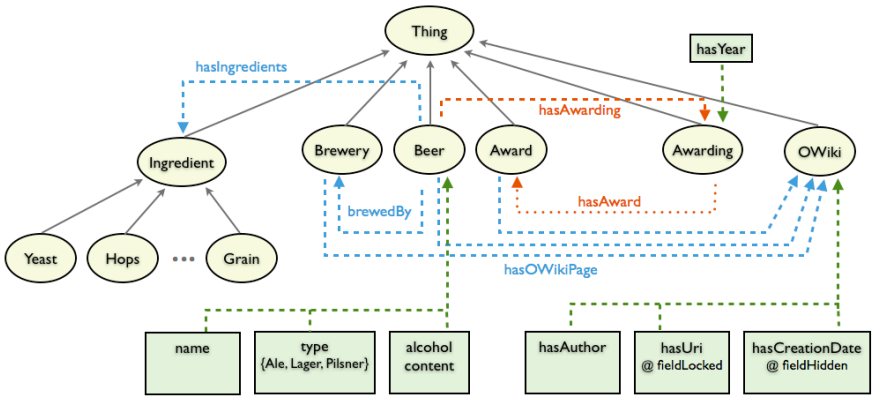

Fig. 2 A graphical representation of the OWiki domain ontology about beers.

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* become *typed links*.
Notice that such ontology uses the facilities discussed in the previous section to define which classes need to be mapped into wiki pages (through the "hasOWikiPage" property) or which properties of each class need to be viewable or editable (through the annotation properties shown in the right-bottom corner of the picture). Notice also that the class *Awarding* does not have the "hasOWikiPage" property set: that class, in fact, in not meant to be mapped into a wiki page, but will be only used to "flat" properties according to the strategies described in Section 3.1.1.

The OWiki conversion process also produces forms to edit the ontological content. Forms are dynamically built by analyzing the class properties of the imported ontology and by mapping each property in the proper element of the GUI interface.
Notice also that the overall status of the OWiki installation is consistent at the first deployment, *assuming that domain input ontology was consistent*. The process is in fact a straightforward translation of classes and relations into pages and links.
In the example, the class *Beer* defines three properties: *name*, *beer_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 adds 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*. The *InclusionList* controller is not used in this example. 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. Finally, the relations between the individuals of the classes *Beer*, *Award* and *Awarding* will be processed to add information to each beer that won a prize: that information is "flattened " from the awarding event individual, as explained in Section 3.1.1.
A point is very important: there is a default mapping between classes of the domain ontology and elements in the GUI ontology based on *the type of the properties*. The name of a property or its meaning in a specific domain is not relevant.
There is actually a configuration file that specifies, for each type, which widget to use and how to configure it. In the previous case, for instance, there was an association between enumerations and radio buttons. That mapping is deployed whenever a class has a property which may only have a finite set of values, regardless of the actual domain ontology. In fact, a change in the OWiki configuration file would be reflected in using a different widget for the same property.

4.2. Forms customization and filling

Furthermore OWiki includes a configuration interface that allows users to set a domain-specific mapping between the input (domain and GUI) ontologies, and to configure the overall organization of the form and its formatting properties.
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. Figure 3 shows a simple example of a customized form: 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.

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 our future activities is the implementation of such an editor within the OWiki framework.

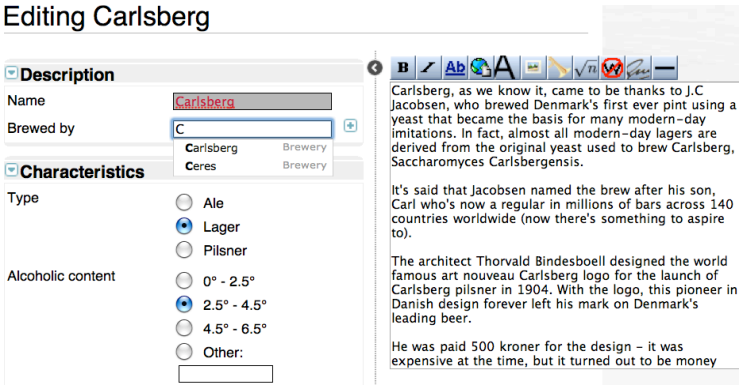

Fig. 3 A customized form generated by OWiki.

4.3. 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 class 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 corresponds to a field in the form that, in turn, corresponds to a parameter and its value in the domain ontology. As expected, the data properties 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]] }}
```

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 any impact on the 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.
Thus, we can conclude – going back to the distinction between "wikis for ontologies" and "ontologies for wikis" proposed in the related works section – that OWiki currently belongs to the second group and does not properly use the wiki to build and update ontologies. 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.

5. The Architecture of oWiki

OWiki is an integrated framework composed of 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 (http://jena.sourceforge.net) and OWLAPI (http://owlapi.sourceforge.net);
- 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 overridden 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 (http://pellet.owldl.com), that is exploited to extract information about the instances in the ontology. Ranges of some properties, as well as their values, are in fact derived from subsumptions or other relations expressed in the ontology itself.

The web-service actually generates templates from the ontological data, that are later sent to the PHP module and stored in 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 (http://mootools.net), in charge of actually generating and delivering forms. It is strongly based on the Model-View-Controller (MVC) pattern and its internal architecture can be divided in four layers:

- *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 of 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, containing atomic and complex widgets, manipulators and decorators.
- *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.

6. OWiki in real-life scenarios

OWiki is successfully used for two instantiations of community-driven semantic wikis: *PoiStory* (http://www.poistory.it) and the *ACUME2 Editor* (http://acume2.web.cs.unibo.it). This section briefly describes these projects sketching out the role of OWiki. More details are in [Di Iorio et al. 2010].

PoiStory is an enhanced wiki environment that allows users to write, share and print customized touristic guides. A set of OWL ontologies drives the overall content management process, according to the schema described in Section 3. In fact, the main ontology within PoiStory models all concepts of the domain (Points of Interest, touristic data, locations, itineraries, guides, etc.) that are manipulated by OWIki: the instances of the domain ontology are mapped into wiki pages, their properties are mapped into fields of infoboxes and indirect properties are *flattened*. When editing a page, the PoiStory editor shows a textarea, where users can freely modify the content, and a form, where users can add ontological data that will be displayed in the infoboxes. These forms are *automatically* generated by OWiki too. Notice also that PoiStory users can customize forms (if they have access permissions to do so) and change both object properties (adding typed links to pages corresponding to objects in the ontology) and data properties (changing atomic values in the form) in a transparent manner.

The *ACUME2 Editor* is a customized wiki platform for the collaborative development of the *ACUME2 epistemological grid*. Such a grid is a table of definitions of terms created by researchers and connected each other. The goal is to build an infrastructure where researchers from various disciplines can communicate without ambiguities. The epistemological grid in fact provides a free-text description of the term for each discipline and each term, and generates a network of labeled references to other terms and concepts. The *ACUME2 Editor* is an instantiation of OWiki. The domain-ontology in this context models *concepts* and *terms* and their connections. Such ontological data are *automatically* converted into wiki pages, infoboxes and forms (to edit properties) by OWiki. The result is a friendly environment that researchers could use to further extend the grid producing a rich and cross-disciplinary graph of relations among terms and concepts.

7. Conclusions

One of the criticisms to the Semantic Web is that the process of creating ontologically sound content is still complex and in the hand of expert users. Semantic wikis are a valid solution to address such issue: they in fact exploit the free and open editing model of wikis to make users create a semantic knowledge-base easily. OWiki makes a further step towards the same goal: allowing users to write ontological content within *not-natively-semantic* wikis too, though automatically generated forms and templates.

Forms, templates and ontologies are strictly connected in OWiki: templates are automatically generated from ontological data (definition of classes and properties), forms are automatically generated from templates, and eventually ontological data (individuals and property values) are populated through these forms.

Such a strong connection does not exist between the free-form textual content of a page and the structures (templates and forms) that store ontological data with-

in OWiki. The latter are actually used to create and manipulate semantic content, while the wiki text is mainly used to add notes and further explanations. The next step of our research will be to integrate these two editing approaches, letting authors use both plain wiki textareas and templates/forms to edit the same semantic data in a fully synchronized environment.

Acknowledgment

The authors would like to thank Silvia Duca and Valentina Bolognini for their previous works on GAFFE, that originated OWiki.

References

[Berners-Lee et al. 2001]
Berners-Lee T, Hendler J, Lassila O (2001) The Semantic Web. In Scientific American, pp. 34-43.

[Munat 2004]
Munat B (2004) The Lowercase Semantic Web: Using Semantics on the Existing World Wide Web.

[Buffa et al. 2006]
Buffa M, Gandon F, Ereteo G et al (2008) SweetWiki: A semantic wiki. In Journal of Web Semantics, v. 6(1), pp. 84-97.

[Völkel et al. 2006]
Völkel M, Krötzsch M, Vrandecic D et al (2006). Semantic wikipedia. In Proceedings of the 15th international conference on World Wide Web (WWW 2006), pp. 585-594. Edinburgh, Scotland.

[Koren 2008]
Koren Y (2008) Semantic forms. http://www.mediawiki.org/wiki/Extension:Semantic_Forms.

[Schaffert 2006]
Schaffert S. (2006) IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06), pp. 388-393. Manchester, UK.

[Passant and Laublet 2008]
Passant A, Laublet P (2008) Towards an Interlinked Semantic Wiki Farm. In 3rd Semantic Wiki Workshop (SemWiki2008), co-located with ESWC2008. Tenerife, Spain.

[Bolognini et al. 2009]
Bolognini V, Di Iorio A, Duca S et al (2009) Exploiting Ontologies To Deploy User-Friendly and Customized Metadata Editors. In Proceedings of the IADIS Internet/WWW 2009 conference. Rome, Italy.

[Di Iorio et al. 2010]
Di Iorio, A., Musetti, A., Peroni, S., Vitali, F. (2010) Crowdsourcing semantic content: a model and two applications. In Proceedings of the 3rd International Conference on Human System Interaction (HSI'10). Rzeszów, Poland.