EXPLOITING ONTOLOGIES TO DEPLOY USER-FRIENDLY AND CUSTOMIZED METADATA EDITORS

Valentina Bolognini
DSAW
University of Bologna
vbologni@cs.unibo.it

Angelo Di Iorio
Department of Computer Science
Univrsity of Bologna
diiorio@cs.unibo.it

Silvia Duca
DSAW
University of Bologna
silvia.duca@unibo.it

Alberto Musetti
Department of Computer Science
Univrsity of Bologna
musetti@cs.unibo.it

Silvio Peroni
Department of Computer Science
Univrsity of Bologna
speroni@cs.unibo.it

Fabio Vitali
Department of Computer Science
Univrsity of Bologna
fabio@cs.unibo.it

ABSTRACT

The Model-View-Controller is a well known architectural pattern, derived from software engineering theories, that provides a clear separation between three dimensions: "how data is modelled" (model), "how the model is shown in output (view)" and "how input events affect the inner state of the model (controller)". In this paper we employ the MVC pattern as the basic principle for the implementation of a sophisticated, user-friendly, generic, customizable metadata editor. The flexibility of the MVC approach is shown in Gaffe (Generator of Automatic Form - Final Edition), a metadata editor that uses different OWL ontologies to create custom forms for the input of instance values for an ontological metadata schema. The separation of the ontologies makes it possible to create metadata editors for any schema, and to customize the forms themselves to provide the best user experience in filling in the required metadata values. Gaffe exists in two different implementations: *Gaffe for Word Processor* (an Open Office and Microsoft Office plug-in) and *Ontological Wiki* (a MediaWiki plug-in), used to embed structured values to an electronic document according to the corresponding syntax. Forms are fully customizable both in appearance (supporting a variety of widget types, as well as grouping, labelling and positioning) and in help features (such as default values, warning messages, pre-processed proposed values, validity verification, etc.).

KEYWORDS

Model/View/Controller, Semantic Forms, Gaffe.

1 INTRODUCTION

Since its very first introduction [Reenskaug, T.], the Model-View-Controller (MVC) architectural pattern has been considered as one of the best way to build GUI-based applications, characterized by strong interaction with users. The two main advantages of MVC are, as known:

- the possibility to develop and use different views for the same model, without modifying anything of the model itself;
- the possibility to change the operations on the model – in terms of storing, connecting, conceptualizing data, etc. – without re-writing all the views but simply adapting them.

MVC have been adopted in many different scenarios, from on-line banking to enterprise web sites, from wikis to CMSs, from stand-alone applications to distributed ones. This paper investigates a new domain of application for such paradigm: *the design and implementation of metadata editors*, i.e. applications that enrich digital documents with user-specified metadata . We show how the MVC pattern provides an *easier* way to develop these applications and allows designers to produce more flexible and extensible tools.

The most common approach to help users in writing metadata, in fact, is to provide them with customized forms, created from a specific data model. That approach is adopted by word processors – such as Microsoft Word and OpenOffice - and wikis, for example. Such metadata editors are usually fully embedded in the content editor, with several disadvantages for the users:

- the interface cannot be customized for different users, according to their needs and preferences;
- metadata values are strictly bound to a specific, and usually not modifiable, metadata schema;
- values edited through a particular editor cannot be exported into a different one.

While these editors embed metadata directly in the final document (often encoded in a format-specific way), other tools exist that provide users with a complex infrastructure to store, retrieve and search metadata. That is the case of DMSs (Document Management Systems) such as [WordPress] or [MediaWiki].

Although more extensible, such solutions do not address a very relevant issue: persistent association of a document with its relevant metadata. Since all metadata are stored in the DMS environment only, and are associated to the document as external database rows, it is impossible to copy/move/download/send a document from the DMS to another location, while keeping all metadata with it.

By embedding metadata information into the documents themselves, without relying on external infrastructures, on the other hand, it becomes easier to preserve metadata across different systems and for a long period. The non-customizability of the model and the interface, on the other hand, is still an open issue.

The solution proposed in this paper consists in a model to create arbitrary metadata editors by exploiting OWL ontologies [W3C OWL Working Group] to select and combine a large number of metadata schemas (for instance document-related schemas such as Dublin Core [Dublin Core Metadata Initiative] or FRBR [IFLA] and other types of schemas such as SKOS [Miles, A. *et al.*] and FOAF [Brickley, D. *et al.*] and to make relationships among these schemas explicit. Relations among OWL entities make it possible to generate high-level descriptions of metadata coming from different basic schemas, that can be directly mapped into different customized schemas. At the same time, ontologies can be exploited to describe editors interfaces and to "connect" these interfaces with the actual metadata. Our model makes it even possible to develop metadata editors by simply instantiating a particular GUI ontology – describing forms through OWL classes and properties – and by relating each individual of that ontology with a particular class or property of the domain ontology. The existence of two separate ontologies, one devoted to describing the metadata domain and the other describing the form-based interface rests on a strong MVC-inspired approach, which allows the system to be associated to arbitrary metadata schema and yet to be used in plenty of ways.

We refer to the model describing metadata editors with an ontological MVC approach as the *Generator of Automatic Forms – Final Edition*, or *Gaffe*. Out of this model, we developed two different applications, called *Gaffe for Word Processor* (a plug-in for both Open Office and Microsoft Office documents) and *OWiki* (for Ontological Wiki, a MediaWiki plug-in), which are used to embed structured metadata into digital documents, office documents and wiki pages alike. Both these applications are presented in this paper.

The paper is structured as follows: in section 2 we show a few related works about metadata editors; in section 3 and 4 we give with more detail about the MVC architectural pattern and the Gaffe model based on it; we then present the two applications based on Gaffe and propose future developments in section 5. We finally bring our conclusions in section 6.

2 RELATED WORK

Adding metadata into a document is a long and tedious task, especially if not using special software developed specifically to make this job easier and more practicable. Luckily nowadays we can find many tools, such as metadata editors and automatic content processing mechanisms, that offer a good way to bring about this task.

Some such editors were developed to add metadata belonging to a specific schema, without relying on Semantic Web technologies, such as OWL ontologies. For example, DC-dot [Powell, A.], developed by the University of Bath, retrieves a Web page and automatically proposes metadata according to the Dublin Core standard. The generated metadata can then be edited using the form provided, with optional, context-sensitive help available while editing. Among the drawbacks, DC-dot only provides text areas for each Dublin Core field, and allows no domain-specific constraints, defaults or controlled vocabulary for local customizations of the overall Dublin Core standard.

Another tool that is specific to a particular metadata schema – again, Dublin Core – is [Metamaker]. The editors allows the user to create metadata values from scratch by using a simple web form and save them in a number of different formats (in particular, HTML, XHTML, XML, RDF or AGRIS AP), but improves over DC-dot because it employs controlled vocabularies of frequently used terms (taken from the AGROVOC and AGRIS thesauri) to allow for standardized description of the relevant fields.

Schema-agnostic editors have been developed as well. TKME [Schweitzer, P.] is an application that allows the creation and modification of metadata values referring to no particular schema as long as a hierarchical tree structure is provided. Unfortunately, even if TKME does not constrain values to a predefined schema, it does not offer any way to customise its interface to suit to specific schemas, nor to help the user to adopt and align to local metadata policies.

Associating ontologies to metadata editing is possible with Metasaur [Kay, J. *et al*.]. Metasaur provides a general visualisation tool for ontologies describing a domain. Besides building lightweight ontologies for existing metadata schemas, the user can enhance the ontology with additional constraints and controlled vocabularies. In fact, it is possible to add metadata by selecting individual classes and properties of all the pre-loaded ontologies. Unfortunately, Metasaur generates the user interface in a completely automatic way, based only on the provided ontological classes and properties, and does not allow the user to model or customize the form itself to help the user in providing the actual values associated to the documents.

Some approaches exist that address metadata editing within Web applications, such as Wikis. For example, SemanticForms [Koren, Y.] automatically generates forms allowing users to insert structured data, such as metadata, in wiki pages. Developed as an extension of MediaWiki, Semantic Forms does not take as input an OWL/RDF file but a MediaWiki template source code, and requires the Semantic MediaWiki [Krötzsch, M. *et al*.] extension. Semantic MediaWiki gives the possibility to refer to the content of wiki pages according to the semantic annotations stored therein. As such, Semantic MediaWiki works on the top of structured content, that can be created either manually or through automatic processes, and Semantic Forms provides users with a simplified interface for the authoring and customization of such structured content. Forms, on the other hand, are not generated automatically, nor through the help of semantic web technologies, and as such require still a good dose of technical and manual intervention.

Similar to the previous one, WikiTemplate [Haake, A. *et al*.] allows users to access a page in two modes: when a page is viewed it is formatted according to its view template, that organizes the metadata as tables on the page itself; when it is edited, a set of editable text areas is supplied, each corresponding to an element of the template. End users can freely edit each text area of the form, while *tailors* can combine different pieces into complex forms to better match both the structure of the content and the practices of the local community.

3 DESIGNING SOPHISTICATED METADATA EDITORS

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.
- *Customizability*: instead of generating a static form – strictly dependent on a given schema – the editor should be customizable for schemas and users' needs and preferences.
- *Proactivity*: the editor should provide users with facilities that simplify the authoring process, such as pre-filled form fields, suggestions, default values, access to environment variables, and so on.
- *Validation*: the editor should apply validation mechanisms to check the correctness of the values.

A solid approach to flexible interfaces consists of adopting the "Model-View-Controller" (MVC) pattern [Gamma, E. *et al*.], 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:

1. *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.
2. *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.
3. *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.

4 EXPLOITING ONTOLOGIES TO DESIGN METADATA EDITORS: GAFFE

We propose to use ontologies to address the issue of associating structured content to electronic documents in order to help authors to create effective relationships between metadata instances and their schemas.

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* (*Automatic Generator of Form – Final Edition*). 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.

5 DELIVERING GAFFE FOR HETEROGENEOUS SCENARIOS

In this section we introduce two different implementations of metadata editors based on the *Gaffe* model. The first is called *Gaffe for Word Processor* and is meant for metadata editing inside a word processor such as Microsoft Word and Open Office Writer. The second is called *Ontological Wiki* and it is used to embed structured content into a MediaWiki page according to the syntax of wiki templates.

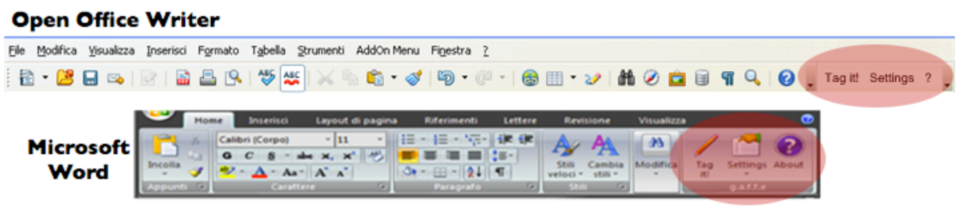
5.1 Gaffe for Word Processor

OpenOffice.org and MS Office editors do allow metadata (through the so-called properties of a document), but use a fixed and flat metadata schema. Their schema is not particularly rich and although it is adaptable for all kinds of contexts and requirements, it requires significant work from the creator of the metadata values themselves, to create not only the values, but also the names of the properties and the constraints on their values. Furthermore, even if it is possible to specify customizable properties, all the metadata always refer to the document itself, and as such it is impossible to clearly and univocally associate properties to concepts and entities referred to by the metadata, different from the document itself For example, a property *affiliation* exists in MS Word, which is probably meant to refer to the affiliation of the agent specified in the *author* property, but this is not clearly specified, and no similar subproperties are allowed for other contributors such as Manager.

The Gaffe architecture has been the basis for the development of *Gaffe for Word Processor* (GaffeWP), whose purpose is to catalogue document collections with semantic metadata, embedded in the document itself. It is a plug-in for both OpenOffice Writer and Microsoft Word that allows adding metadata using customised metadata schema.

Clearly, metadata persistence is a fundamental need we want to preserve given the many ways we use to exchange documents: by copy, by mail and by download. Usually, considering the context of the Document Management System, these operations cause the loss of metadata when they are not embedded in the documents themselves. In order to prevent these kinds of problems, GaffeWP can store and preserve metadata within the documents themselves.

Figure 1: the GaffeWP interface (for both OpenOffice Writer and Microsoft Word) to access the metadata forms.


Open Office Writer
File Modifica Visualizza Inserisci Formato Tabella Strumenti AddOn Menu Finestra ?
Tag it! Settings ?
Microsoft Word
Home Inserisci Layout di pagina Riferimenti Lettere Revisione Visualizza
Tag it! Settings About

As shown in Figure 1, GaffeWP allows adding metadata through the “Tag It” button. After this action, it will open a form – for example, the one shown in Figure 2 based on the Dublin Core schema [Dublin Core Metadata Initiative] – generated by using the ontologies introduced by Gaffe. The metadata values are stored in two different ways:

- by inserting them within the document, in order to guarantee the metadata persistence in every possible scenario; OpenDocument e Office Open XML actually use a subdocument (meta.xml and core.xml respectively) in which metadata is stored, and allow for custom properties to be specified.
- by storing them as domain ontology instances in an OWL database to increase effectiveness and collaborative sharing of data using Semantic Web technologies.

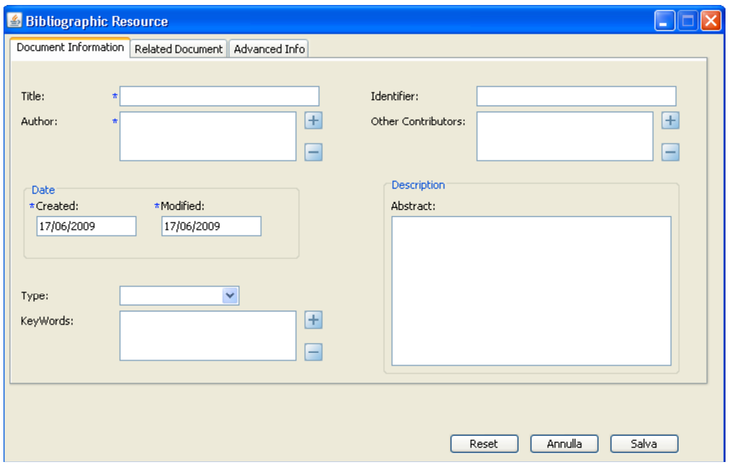
Apart from helping users offering suggestions for metadata values, based on previous savings, it is also possible to use different domain ontologies contemporaneously, and each of them can be associated to different views depending on a particular context or user needs.

As said, display processes are used by GaffeWP in order to provide help to users. A pre-display process shows default values when a form has just been created, using the javascript function specified by the property “defaulting” defined in the instance ontology. The application can use system information and word processor or author-defined data to insert a right default value.

Going beyond the default value issue, to specify metadata about individual items of a large collection of documents is a very long process, even characterized by repeated actions: probably, the metadata author must insert many times the same value in different documents (for example, the author's name). In order to keep editing actions more faster, GaffeWP helps user showing suggestions while he is typing. The suggestion are old metadata values belong indexed through an chronological history.

Another main feature concerns the mechanisms for verifying the correctness of the proposed metadata values.

Figure 2: the GaffeWP form used to add Dublin Core metadata to an office document.

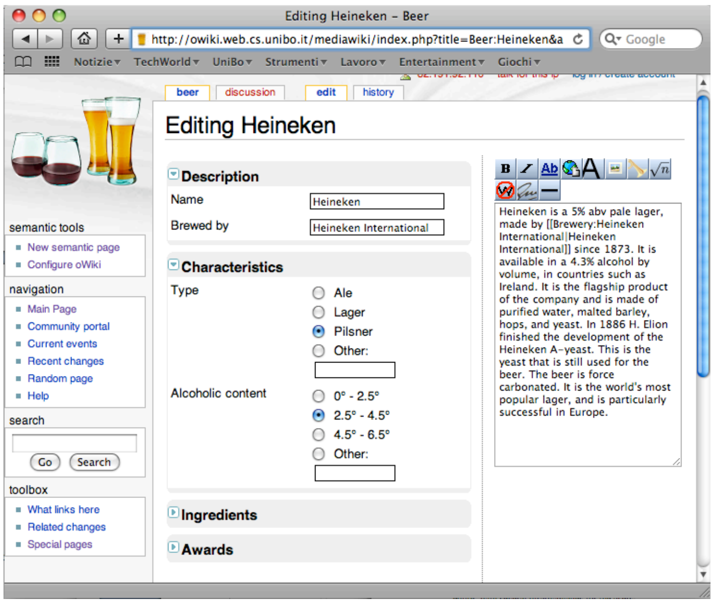


Created: 17/06/2009
Modified: 17/06/2009

These checks, created through both the instance and the domain ontologies, refer to four different kinds of constraints:

- the *type* specification (string, integer, data, etc.) of metadata values that can populate an instance;
- the *requirements* of a value, in terms of mandatory, optional and recommended actions;
- *controlled vocabularies*: the instance ontology may specify a structured list of literal values for a metadata item. With a controlled vocabulary, only specified values are allowed.
- *ontologies*: the domain ontology may define the properties of domain elements and provide restrictions on allowed metadata values.

Figure 3: the OWiki editing interface for the content and the structured data of the MediaWiki page.



http://owiki.web.cs.unibo.it/mediawiki/index.php?title=Beer:Heineken&a

Editing Heineken

Description

Name: Heineken
Brewed by: Heineken International

Characteristics

Type: Ale / Lager / [✓ Pilsner] / Other:
Alcoholic content: 0° - 2.5° / [✓ 2.5° - 4.5°] / 4.5° - 6.5° / Other:

Heineken is a 5% abv pale lager, made by [[Brewery:Heineken International|Heineken International]] since 1873. It is available in a 4.3% alcohol by volume, in countries such as Ireland. It is the flagship product of the company and is made of purified water, malted barley, hops, and yeast. In 1886 H. Elion finished the development of the Heineken A-yeast. This is the yeast that is still used for the beer. The beer is force carbonated. It is the world's most popular lager, and is particularly successful in Europe.

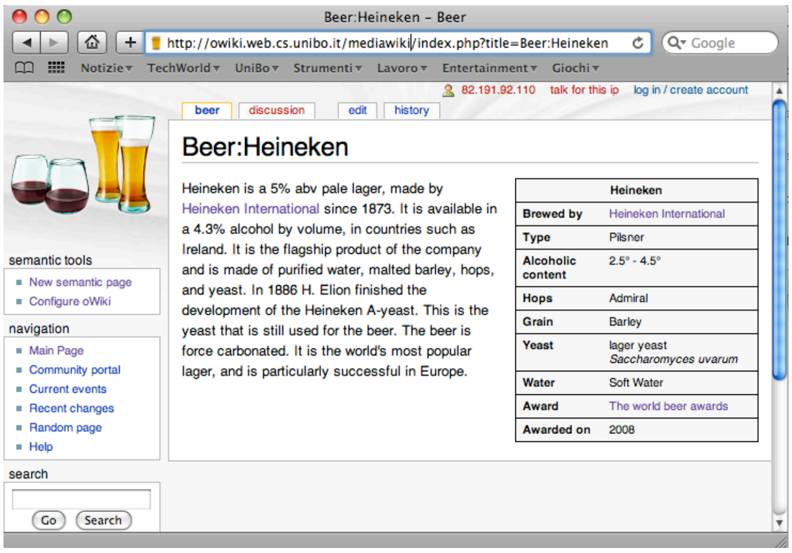
5.2 OWiki: Ontological Wiki

Ontological Wiki (oWiki) is a Semantic Web application, made using the Gaffe model, developed as a MediaWiki plug-in. It is a dynamic system able to include ontological data structures within wiki pages, used also to simplify the content creation through complex forms.

The major benefit in using this application is the possibility to create semantic data without changing the approach in which a user creates wiki content, because the generation of semantic content is done by the application itself, hiding the actual complexity behind this operation. It allows adding metadata to web documents through a dynamically-created form, as shown in Figure 3. It is possible to specify a customizable structure for displaying all the metadata (e.g., the MediaWiki templates).

Basing all the infrastructure on the Gaffe model, oWiki automatically creates a particular instance ontology – starting from the other two, the domain and GUI ones, specified during the plug-in installation process – for each wiki page created by the content authors. The first time a wiki page of a specific domain concept is edited, oWiki shows a very basic form and saves it as a special page. This page can then be organized as a new form by adding dynamic behaviors, moving buttons, changing the field order and so on.

Figure 4: A MediaWiki page showing content and structured data related to a specific domain (in this case, beers).



Beer:Heineken

Heineken is a 5% abv pale lager, made by Heineken International since 1873. It is available in a 4.3% alcohol by volume, in countries such as Ireland. It is the flagship product of the company and is made of purified water, malted barley, hops, and yeast. In 1886 H. Elion finished the development of the Heineken A-yeast. This is the yeast that is still used for the beer. The beer is force carbonated. It is the world's most popular lager, and is particularly successful in Europe.

Heineken	
Brewed by	Heineken International
Type	Pilsner
Alcoholic content	2.5° - 4.5°
Hops	Admiral
Grain	Barley
Yeast	lager yeast *Saccharomyces uvarum*
Water	Soft Water
Award	The world beer awards
Awarded on	2008

The user data values are inserted within the document by filling the form. Differently from existing semantic wikis, that store the semantic data within the content of the document, oWiki provides semantic data as tabular structured (infoboxes, in the MediaWiki jargon). This is done in two steps:

- the page content is stored in the standard MediaWiki database, using the usual wiki text representation, in order to create the page content and the relative infobox, as shown in Figure 4;
- the metadata is also stored as specific OWL assertions according to the domain ontology, and is handled by an external Java web service.

5.3 Future works

We are currently working on some extensions to Gaffe. The first and most relevant evolution for all the Gaffe-based applications is to have a mechanism to digitally certify the metadata specified by a user. When describing the lifecycle of a document directly in its metadata, it is important to record who added each item to it. Moreover, in many context it is often required to certify the added data, for example by adding a digital signature. To this end, a new subproject has been started that will develop extensions for the *Certified Gaffe*.

Another ongoing activity is to develop a standalone application, *Metagaffe*, that is meant to be a form for creating domain and instances ontologies. As described in the previous sections, Gaffe allows us to create complex forms starting from a domain and a set of widgets, and relating them through an instance ontology that defines both the form layout and the logical operations. We now need to develop two specific domain and instance ontologies to make forms to create new domain and instance ontologies for Gaffe itself.

The long term evolution of Gaffe is the development of a GUI ontology that can describe a lot of possible interface scenarios, from the desktop applications to the Web-related ones, to have many other possibility, apart from by using forms, in displaying and modifying metadata and documents themselves. One of the ongoing European project concerning these topics is the *Interactive Knowledge Stack* [IKS Project]. Its final goal is to provide an open source platform for semantically enhanced content management systems.

6 CONCLUSIONS

This paper discussed a possible instantiation of the MVC pattern for implementing metadata editors. The main issue that inspired this research was the difficulty to build generic and schema-independent editors. Nowadays, in fact, most metadata are handled by *ad hoc* tools very complex and difficult to be reused in multiple contexts. The main contribution of this work was to show how ontologies can be exploited to easily create generic but powerful editors. The automatic generation of these editors strongly relies on a clear distinction between the (meta)data (model) and their presentation to the users (view). That is why very strong analogies exist between MVC and the GAFFE framework.

Two actual implementations of such a generic framework were also presented. Delivered in different scenarios - web applications and stand-alone editors - the ontological MVC approach proved to be flexible and extensible, and to radically simplify the management of metadata.

ACKNOWLEDGEMENT

The authors wish to thank Microsoft Italy, for the partial support to this project, and Valentina Presutti and Aldo Gangemi of the CNR Semantic Technology Laboratory in Rome for their helpful comments and suggestions.

REFERENCES

Brickley, D., Miller, L. (2007). *FOAF Vocabulary Specification 0.91*. http://xmlns.com/foaf/spec/.

Dublin Core Metadata Initiative (2008). *DCMI Metadata Terms*. http://dublincore.org/documents/dcmi-terms/.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, New York, NY, USA.

Haake, A., Lukosch, S., and Schummer, T. (2005). Wiki-templates: adding structure support to wikis on demand. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*. ACM Press, New York, NY, USA, 41-51.

IFLA (2009). *Functional Requirements for Bibliographic Records: final report*. http://archive.ifla.org/VII/s13/frbr/frbr_2008.pdf.

IKS Project. http://www.iks-project.eu.

Kay, J. and Lum, A. (2003). *An Ontologically Enhanced Metadata Editor*, University of Sidney, School of Information Technologies, Technical Report 541. http://www.it.usyd.edu.au/research/tr/tr541.pdf .

Koren, Y. (2008). *Semantic forms: Creating forms for MediaWiki*. http://www.mediawiki.org/wiki/Extension:Semantic_Forms.

Krötzsch, M., Vrandecic, D., Völkel, M. (2006). Semantic MediaWiki. In the *Proceedings of the International Semantic Web Conference 2006 (ISWC 2006)*. Athens, GA, USA.

MediaWiki. http://mediawiki.org.

Metamaker. http://www.fao.org/aims/tools/metamaker.jsp.

Miles, A., Bechhofer, S. (2009) *SKOS Simple Knowledge Organization System Reference*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/skos-reference/.

Powell, A. (2000). *Dc-dot, Dublin Core Metadata Edito*r. UKOLN, University of Bath. http://www.ukoln.ac.uk/metadata/dcdot/.

Reenskaug, T. (1979). Model-View-Controller: XEROX Park 1978-1979. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

Schweitzer, P. (2007). *Tk Metadata Editor*. http://geology.usgs.gov/tools/metadata/tools/doc/tkme.html.

W3C OWL Working Group (2009). *OWL 2 Web Ontology Language Document Overview*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/owl2-overview/.

WordPress. http://wordpress.org.