

A Tool Suite to Enable Web Designers, Web Application Developers and End-users to Handle Semantic Data

Mariano Rico, José Antonio Macías, David Camacho

Computer Science Dept.
Escuela Politécnica Superior
Universidad Autónoma de Madrid, Spain

Óscar Corcho

Ontology Engineering Group
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid, Spain

Current web application development requires highly qualified staff, dealing with an extensive number of architectures and technologies. When these applications incorporate semantic data, the list of skill requirements becomes even larger, what represents a high adoption barrier for the development of semantically-enabled Web applications. This paper describes VPOET, a tool focused mainly on two types of users: web designers and web application developers. By using this tool, Web designers do not need specific skills in semantic web technologies to create web templates for handling semantic data. Web application developers incorporate easily those templates into their web applications, by means of a simple mechanism based in HTTP messages. And end-users can use these templates by means of a Google Gadget created to this end. As web designers play a key role in the system, an experimental evaluation has been carried out, showing that VPOET provides good usability features for a representative group of web designers in a wide range of competencies in client-side technologies, ranging from amateur HTML developers to professional web designers.

Web application development is becoming increasingly difficult, especially when focusing on designing attractive and reusable web applications. Web application developers need to be skilled in a wide set of client-side technologies (e.g., HTML, Javascript, CSS, DHTML, Flash, AJAX) and server-side ones (e.g., JSP, ASP, .NET), addressing an extensive number of programming languages (e.g., Java, Python, Ruby, PHP). The fast and divergent development of client-side technologies, which have formed the basis for the emergence of the Web 2.0 concept (O'Reilly, 2005) and attitude (Davis, 2005), has increased the importance of having experts in this type of technologies. In this paper, we refer to these client-side experts as web designers. This heterogeneity and complexity converts web designers in skilled programmers, as pointed out by Rochen et al (Rochen, Rosson, & Pérez, 2006), and increases the development cost of such applications.

The situation becomes even more complex when Web application developers want to incorporate Semantic Web data in their applications. Although many domains have a lack of ontologies, semantic technologies are mature enough and there is an increasing number of ontologies publicly available. According to d'Aquin et al, in 2007 there were around 23,000 ontologies available on the Internet (d'Aquin et al., 2007), and this number is growing quickly, especially due to the Linked Open Data initiative (Bizer, Heath, & Berners-

Lee, 2009).

Therefore, this wealth of information still remains hidden to most Web application developers and end-users. On the one hand, semantic web technology experts are not usually focused on producing attractive and reusable web applications what makes it difficult for end-users to access this information (Macías & Castells, 2007). For example, early work on Semantic Web portals showed that usability aspects were not well considered in general in semantic web portal technologies (Lausen et al., 2005). From our experience with semantic web portals (Corcho, López-Cima, & Gómez-Pérez, 2006), we did conclude that such portals required very skilled people to maintain them, and demanded more flexible frameworks in order to create complex applications combining traditional web development and semantic data management. On the other hand, Web application developers need to improve their skills with those for the understanding and management of ontologies and semantic data, in order to produce semantically-enabled web applications (Oren, Heitmann, & Decker, 2008) (d'Aquin et al., 2008).

Our approach aims at hiding much the complexity of semantic web technologies to the parties responsible for bringing the Semantic Web to end-users. In our previous work (Rico, Camacho, & Corcho, 2008) we divided the skills required to create a semantically-enabled web application into different groups of competencies, identifying the roles (profiles) involved. Two of these roles were identified as developers. The first of them had competencies in semantic web technologies and was low-skilled in web application development. The second developer profile had minimal skills in semantic web technologies but high competencies in web application development. For the first developer profile we built

This work has been partially funded by the Spanish Ministry of Science and Innovation under the projects HADA (TIN2007-64718), METEORIC (TIN2008-02081) and DEDICON (TIC-4425).

a wiki-based framework named **Fortunata** (Rico, Camacho, & Corcho, 2010), which allows semantic web application developers to create, configure and activate pieces of code that run on top of the framework, in a collaborative way. The experimental evaluation of this framework showed that the development cost and the required competencies for handling this framework were lower when compared to traditional technologies. We measured the usability of Fortunata from the developer's perspective by means of early-prototypes of Fortunata-based application, and that information was useful to identify usability pitfalls and successfully solve most of them, as well as to create a "Fortunata-based-application developers guide".

For the second developer profile, the most common currently available, we provide them with a programming-language independent mechanism (based in HTTP messages) to use 'semantic templates'. These templates are created by web designers by using a Fortunata-based application named **VPOET** (Rico et al., 2008), which plays a key role between the two developer profiles described.

The novelty of this paper is the description of the final version of VPOET, and the analysis of VPOET from the perspective of its users.

VPOET lets web designers create semantic templates to present and edit semantic data, even if they are low skilled in semantic web technologies. VPOET may have been implemented with any traditional combination of the aforementioned server-side and client-side technologies. However, the advantage of using Fortunata as a basis for the development of VPOET is that, as we said previously, this framework simplifies the development of semantically-enabled web applications.

To support some of the tasks required by VPOET's users we have also implemented another Fortunata-based tool called **OMEMO**. This application is aimed at web designers with no knowledge about ontology languages. Instead of opening the ontologies in specialized tools, non-skilled users can browse ontology components (classes and properties) by means of a set of wiki pages that are automatically generated whenever an RDFS or OWL ontology is included in the system. This application is similar to those provided for ontology developers by ontology edition tools like Protégé, SWOOP or NeOn Toolkit, or those provided as on-line services, like OWLDoc. However we considered it interesting to have this ontology visualizer highly coupled with VPOET, since understanding the structure of a given ontology component is one of the major tasks to be accomplished by a web designer who is designing a presentation template for an ontology component.

Our hypothesis is that web designers with no previous knowledge about semantic technologies, but skilled in client web technologies in the range from low to experts, can create easily VPOET templates.

To prove the validity of our hypothesis we carried out a usability and user satisfaction evaluation with a representative group of web designers, ranging from low-skilled to advanced. In this evaluation, all these web designers had a common objective: to create a presentation template for a specific

ontology component. Once this objective was accomplished each of them had to answer a detailed set of questionnaires. The analysis of their answers revealed good values for usability and user satisfaction in the whole representative range of web designers, validating our hypothesis.

In the rest of this article we start with a brief description of VPOET architecture. Then the focus of the paper is centred in the two perspectives of VPOET: the one provided to web designers and the one provided to web developers. Next section shows the experimental evaluation of the usability and user satisfaction concerning VPOET. Then, a section showing related work, and we finish with a conclusions and further work section.

VPOET architecture and roles involved

Fortunata¹ is a Java library built on top of the JSPWiki wiki engine. The main features of this engine are its support for the management of forms and its extensibility capabilities by means of plugins. Fortunata simplifies the creation of semantically-enabled web applications by delegating to the underlying wiki engine the client-side presentation and server-side publication of semantic data. The creation of pages is done with a wiki-based syntax, which has predefined constructs to create links, sortable tables, tables of contents, etc. The publication of semantic data is done automatically by the system.

A Fortunata-based application consists of a set of wiki pages that contain regular wiki code intertwined with calls to Fortunata plugins (F-plugins). For instance, VPOET is a Fortunata-based application that consists in four interrelated wiki pages and seven F-plugins. Fig. 1 shows the architecture of VPOET and the roles involved.

Table 1 provides details about these roles, focusing on the activities that they perform, the skills required to perform such activities, and the benefits achieved by this approach. In summary the roles are these:

- *Web application developer* (devel1) creates semantically-enabled web applications by using the templates stored in VPOET. This role only requires basic HTTP management skills in any programming language in order to make a simple HTTP call to get a visualization for the semantic data specified. Playing this role we have created an application oriented to end-users, based in Google Gadgets technology, which enable end-user to handle semantic data in their web pages. This application, named GG-VPOET, is described later.

- *Web designer* (user2) uses VPOET to create templates to present semantic data (output templates), or request data to the user that will be converted to semantic data (input templates). This role requires skills in web design (HTML, CSS, AJAX, etc.).

- *End-user* (user1) uses his/her web browser to use the application created by devel1. In this case, by using the VPOET Google Gadget (GG-VPOET) end-users can handle semantic data in their own pages or any Google product such as Google Docs, Google Pages, or iGoogle.

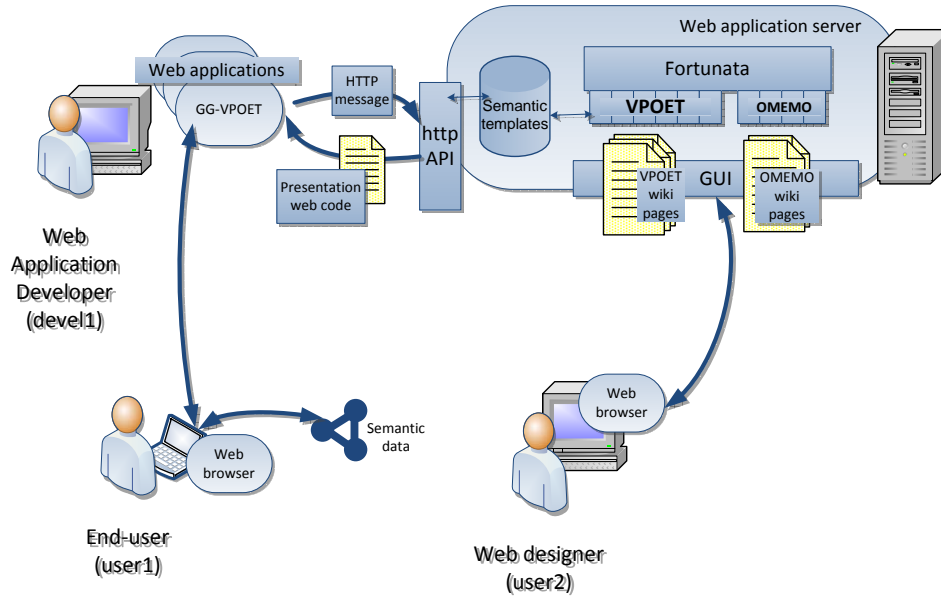


Figure 1. : VPOET architecture and actors involved.

Table 1
: Description on the roles involved in VPOET

Role	Activities	Requirements	Benefits
devel1	Web application developer. Integrates VPOET templates in any web application in order to create semantically-enabled web applications	HTTP management skills in any programming language	No client technologies skills are required
user2	VPOET user. Uses VPOET to design web templates	Web design skills	No semantic web technologies skills are required
user1	VPOET Google Gadget end-user. Uses the VPOET Google Gadget to visualise semantic data	HTML basics (cut & paste simple generated code)	No client or semantic technologies skills are required

VPOET for web designers: Enhancing the presentation and capture of semantic data

VPOET enables web designers (*user2* in Fig. 1), also known as “Template Providers”, to create web templates for a set of ontology components. There are two types of VPOET templates: output templates are intended to present semantic data, input templates are intended to request data from users. For example, let us imagine that we want to create output and input templates to render and create, respectively, semantic data for the concept *Person*, belonging to the FOAF ontology, one of the most popular concepts in the Semantic Web with 60 million instances available². Output templates can be used to render any data source containing instances (individuals) of this class, or any subclass if there are no more specialized templates for them. Input templates are intended to present a form to request data that will be converted to an instance of *Person*.

VPOET is focused on web designers, who should be able to author attractive designs capable of handling semantic data. Hence, VPOET only requires basic skills in client-side

technologies (e.g., HTML, Javascript). As described in following sections, the most difficult task to be performed by web designers is to embed some semantic data management macros in the client-side web code. Hence there is little training needed to start creating templates (a 30-minute online tutorial³ is enough, as demonstrated in our evaluation).

Although it will be described in the next section, Web application developers can use the templates stored in VPOET in their own web applications. From the point of view of end-users who browse semantic data sources through the visualizations generated by output templates or who have to introduce data through input templates, a VPOET-enabled application is like any other Web application, with information shown in simple tables or advanced client-side elements such as Flash.

In the following subsections we will describe the process to be followed in order to create VPOET output and input templates, how to reuse output templates, and we will also provide some additional information about the related Fortunata-based tool OMEMO which can be also used in this process.

Creation of VPOET output templates

The process to create an output template starts with the selection of the corresponding ontology component to be visualized. As an example we show the steps to be followed to create an output template for the concept `Person` from the FOAF ontology:

1. Getting information about the structure of the targeted component. The VPOET user needs to know the structure of this component: its properties, sub-components, etc. Although this information can be obtained by directly analyzing the corresponding ontology in its implementation language (e.g., RDFS, OWL) or by using ontology visualization tools, typical VPOET users are not skilled to achieve this task. We have developed an specific tool named OMEMO (described later) to achieve this task. Figure 6 shows a snapshot of the information provided by OMEMO for the class `FOAF:Person`.

In this case, the VPOET user can see that `FOAF:Person` comprises properties such as `firstName` or `interest`, as well as `firstName` is a `Literal` belonging to the `rdfs` ontology. The link in `Literal` will open the web page for this element.

2. Authoring a web design (with any web authoring tool) where semantic data will be inserted. Table 2 shows a initial example of a web design. The left-most column shows the HTML code, whereas the right-most one shows how it is rendered in a web browser.

3. Choosing an identifier for the template, which will generate the corresponding wiki page in the system with information about the template, as described in step 8.

4. Cutting & pasting the web design code into the appropriate VPOET form fields, with the appropriate HTML, CSS and Javascript code. If the code includes references to files (e.g., images), these can be uploaded into the VPOET or any web server.

5. Replacing any absolute paths in the web design code (e.g. path to images or Javascript files) by the macro `OmemoBaseUrl`, which expands into the appropriate URL at runtime, depending on the URL of the server where the system is running. A list with the available macros can be found in table 3. Our experience has shown that this small set is enough to cover the most common needs for the creation of VPOET templates.

6. Replacing any dummy semantic values in the web design code by specific macros that insert the actual property values at runtime. For example, the dummy text “this is the name” in table 2 should be replaced by the macro `OmemoGetP`, which gets the values of the property indicated. The result of this replacement is shown in Fig. 2. Other macros like `OmemoConditionalVizFor` and `OmemoGetLink` allow reusing existing VPOET templates, and are described later.

7. Testing the design with semantic-data sources (typically external to VPOET) that contain instances of the targeted component. Fig. 3 shows the kind of results that can be achieved when a given output template is tested with data from a given data source (in this case <http://ishtar.ii>

.uam.es/fortunata/foaf.rdf) which contains some instances of the class `FOAF:Person`.

8. Describing the designed template with the following information: template type (input or output), behavior in case of changes to the font size, sizes (preferred, minimum and maximum), code-type (HTML, Javascript, CSS), and dominant colors. This information is included in the corresponding template wiki page, and is useful to discover at runtime the most appropriate template for a given user or interaction device, as described in the conclusions and future work section.

Reusing templates in VPOET

VPOET users can exploit two macros that allow reusing existing VPOET output templates belonging to him/her or to any other template provider, so that it is possible to create compositions of templates and hence more modular designs. This is normally used with properties that connect an individual with another individual (e.g., with the relation `FOAF:knows`). However, it may be also used to reuse specific templates for some types of property values. For example, if a VPOET output template for visualizing e-mails with Javascript obfuscation to avoid SPAM has been created and is available in the system, we may reuse this template when a class contains a property showing the e-mail address of a person or organization. This can be used as well to hide details in order to get a compact visualization. Clicking in a small icon can expand or substitute the visualization area to display details. These expand and/or contraction features require AJAX knowledge or third parties javascript libraries.

The first macro, `OmemoGetLink` (see table 3), specifies a property that is substituted at runtime by a link that points to the VPOET communication servlet, with the appropriate parameters, for rendering the destination instance. For example, the property `FOAF:knows` establishes a relation between an origin (`FOAF:Person` in this case) and a destination (another `FOAF:Person`). If we use this macro for this property, we get a link capable of rendering any “known” person.

When a given property has no value, the macro `OmemoGetP` returns the string “N.A.”. To avoid this effect, the macro `OmemoConditionalVizFor` checks if the property value exists and, if it is the case, uses an existing template to show the property value (in this case, a simple text renderer). This is also applicable with relations that link the instance with other instances.

The code required to create the template shown in Fig. 3 is shown in Fig. 4. Macros are framed within a thick rectangle, and reused templates are framed within a thin stroke.

Creation of VPOET input templates

VPOET input templates are HTML forms that convert the entered values into semantic data (RDF). As in output templates, an input template is intended for an ontology component (which usually comprises a set of ontology sub-components). For example, an input template for the ontology component `FOAF:Person` must provide a form with

Table 2

: Initial output template example. Left: initial HTML code. Right: template rendered in a web browser.

Code	Rendering
<pre> <table style='background: #F0F0F0; padding: 1px; border: thin solid #DDDDDD; margin-left: 1em'> <tr> <td>Name:</td><td>this is the name</td> </tr> <tr> <td>Given Name:</td><td>this is the givenname</td> </tr> <tr> <td>Family name:</td><td>this is the family_name</td> </tr> <tr> <td>Home page:</td><td>http://somewhere.com</td> </tr> <tr> <td>Depiction:</td><td></td> </tr> <tr> <td>Knows:</td><td>list of known people</td> </tr> </table> </pre>	

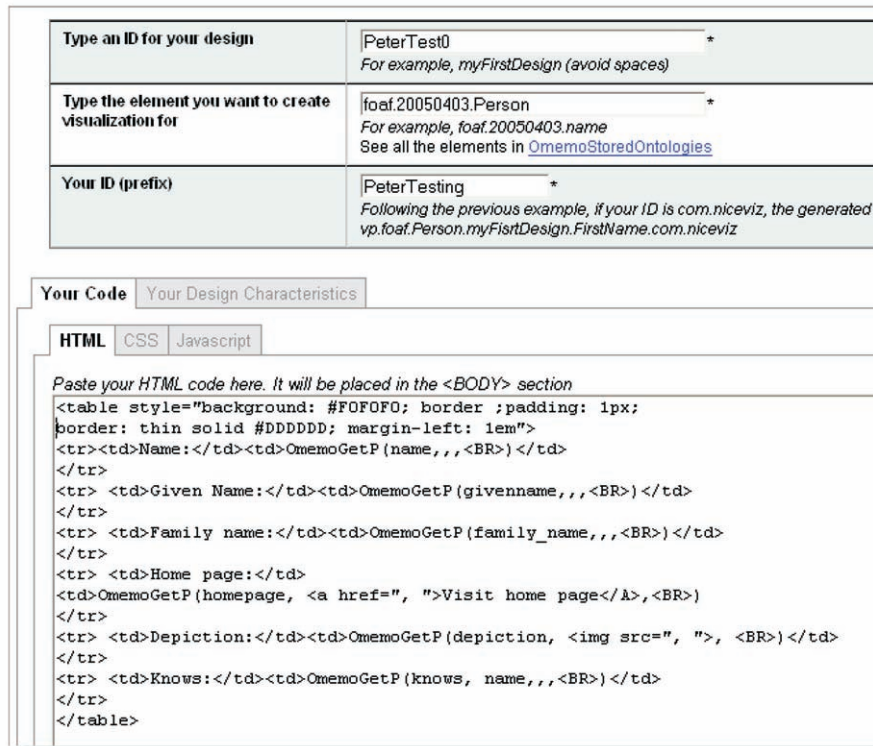


Figure 2. : Placing the source code in VPOET. Some code is replaced by macros.

input fields for firstName, knows, etc. The process to create an input template is similar to the one required to create output templates. The main differences are the following:

- The template source code does not contain macros, although it must follow some conventions: (1) the HTML code must contain an HTML form with the action attribute pointing to an specific servlet provided by VPOET, (2) the form must contain a hidden input field named “ontoelem”, whose value is the name of the ontology component for which this

template has been created, (3) the HTML form controls (i.e. input fields, radio buttons, etc.) must have an attribute name that corresponds to the corresponding component (e.g., property) in the ontology. This can be referenced to with the following notation: `ontologyAlias.version.elem` (this is the OMemo syntax, described in the next subsection), or with the corresponding URI.

- The test phase does not use external semantic data sources. However, as the VPOET servlet generates semantic

Table 3
: Main set of macros available for VPOET template providers.

Macro	Arguments	Explanation
OmemoGetP	propName	It is replaced by the property value propName. As most properties can be multivalued, the next version is used.
OmemoGetP	propName, prefix, posfix, separator	Indicated for multivalued properties, it is replaced by the whole set of values, starting with the code specified in prefix, finishing with the code in posfix, and placing the code in separator between each property value. See an application example in figure 2, in the code placed in the HTML tab.
OmemoGetP	relation, propPref, prefix, posfix, separator	Indicated for relations, it shows the value(s) of the property propPref (preferred) of the entity target of the relation. See an application example in figure 2.
OmemoBaseURL	No arguments	It is replaced by the URL of the server where VPOET is running. See an application example in figure 4
OmemoConditionalVizFor	propName, designerID, designID	It renders the property propName only if it has a value, using the template indicated by designerID and designID.
OmemoGetLink	relationName, designerID, designID	It is replaced by a link capable of displaying components of the type pointed by the relation relationName using the template indicated by designerID and designID. See an application example in figure 4

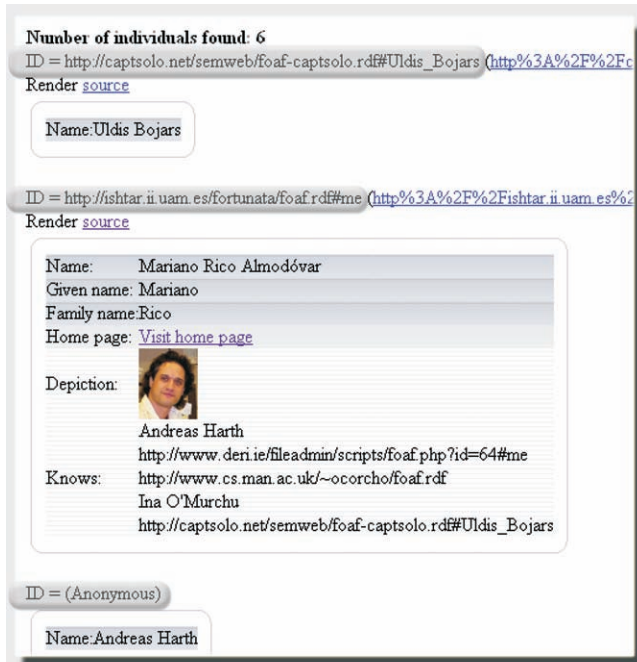


Figure 3. : Testing a sample output template against a given semantic data source. The balloons show different individuals, even anonymous. Non anonymous individuals can be rendered individually if the link in the right side of the balloon is clicked.

data, this source can be rendered by a VPOET output template. Therefore, it is recommended to create an output template for each input template, for testing proposes. Typical errors such as wrong property names can be detected with this strategy.

Fig. 5 shows an example of a VPOET input template for the ontology class FOAF.20050403.Person. The figure shows a “plus sign” on the right of some fields. This is automatically generated by VPOET when a property can have multiple values. Similarly the icon on the right of the property knows is also automatically generated by VPOET. When a user clicks on it, it opens an “instance browser”, a web page that allows end-users to select a set of instances from the corresponding target class. These instances may be available in the system (and then the system shows them in the list) or the end-user may write their URI directly, if they are in a external semantic data source.

As such, input templates are very similar to what has been traditionally done in semantic portals for the introduction of semantic data according to a given ontology model. Hence there is not much innovation on this side, apart from the fact that semantic portals are usually more rigid and do not allow using properties for an individual that are not connected to the class that the individual belongs to. This is something that can be done easily in VPOET.

OMEMO: wiki-based documentation of ontologies

OMEMO is another Fortunata-based application. It stands for “Ontologies for MEre MORTals”, and it is aimed at users with no skills in ontology languages, as mentioned in the introduction. With OMEMO users can find out which compo-



Figure 4. : Advanced example showing macros (thick rectangles) and templates reuse (thin rectangles).

Figure 5. : An example of VPOET input template in action.

nents (classes and properties) are defined in a set of ontologies.

OMEMO generates a set of wiki pages that describe any

Figure 6. : Snapshot of the OMEMO wiki page for class Person (version 20050403) of the FOAF ontology

given OWL or RDF Schema ontology. An ontology may have different versions, which can be distinguished from one another by its publication date or, as proposed in the current OWL 2 working draft (Draft, 2009), by specific versioning properties. In the following example, we consider the FOAF ontology 20050403.

OMEMO only requires the URL of a given ontology, and the generation process results in a page for the ontology and another page per each class and property. Carrying on with the example, Fig. 6 shows a section of the wiki page generated for the class Person (version 20050403). Point ④ indicates that other versions of the same ontology exist

and lets us access those pages through those links. Property `interest` has type `FOAF:Document` (point ❸), and property `firstName` (point ❶) has type `RDFS:Literal` (point ❷). These solid links indicate existing component wiki pages; otherwise, links will be underlined by a dotted-line.

It is worth mentioning that the pages generated by OMEMO are automatically indexed by JSPWiki using the Lucene engine, as it happens with any other wiki page generated manually. Hence search facilities provided by JSPWiki include the text in these pages. Besides, any manually-created wiki page can link to any of these automatically-generated pages.

VPOET for web application developers: including VPOET templates in web applications

As shown in Fig. 1, Web application developers (represented by *devell*) can exploit VPOET templates in order to create easily semantically-enabled web applications.

For example, a web page about the city of Madrid would include the usual HTML code to show the text, multimedia objects and links to internal or external URLs. We can also include additional calls to VPOET templates that link some of those pieces of text, multimedia objects and links with the visualization of semantic data available in the system or in any other external semantic data source. From this description, this configuration may seem to be similar to that of semantic wikis. However, in these systems the visualization of semantic data is done always by reference to internal semantic data and by means of predefined tables, which is not the case with the use of applications exploiting VPOET.

Now we describe how to include VPOET templates into any web application. Developers can use VPOET templates by means of an ad-hoc communication servlet that lets clients make HTTP GET/POST requests with variable parameters in order to facilitate queries like “render the semantic data at URL Z by using the output template X created by designer Y”. This request is codified as `http://URL-to-servlet/VPOetRequestServlet?action=renderOutput&designID=X&provider=Y&source=Z`. In this example, the semantic data are referenced by means of the `source` parameter (GET message), but can be included in the HTTP message (POST message). The complete syntax of these requests is shown in table 4.

Semantic data can be included or not in the parameters of the HTTP message. SPARQL endpoints⁴ provide a similar functionality, but they are oriented to much more specialized developers (those capable of writing the SPARQL queries that must be specified in the HTTP message). As we aim at a wider audience with little or no knowledge about Semantic web technologies, we preferred to provide the current syntax.

HTTP messages with the specified syntax, can be sent to the VPOET communication servlet by other programs written in any programming language, or by Javascript applications executed in a web browser. However, browsers are more limited than other applications because they suffer security limitations due to the fact that communication is re-

stricted to the server which holds the web application. This is the case for Tabulator (Berners-Lee et al., 2006), which requires reducing the security restrictions of the browser. Our approach does not have this problem because communications between the end-user and the different semantic data sources are centralised by VPOET.

VPOET Google Gadget (GG-VPOET)

Besides the previous HTTP-based mechanism, and aimed at decreasing even more the required skills needed to use VPOET templates, we have created a Google Gadget called GG-VPOET⁵. By using this gadget, any developer or end-user can render easily a semantic data source or provide a web interface to create semantic data. GG-VPOET, as any other Google Gadget, can be inserted into an ordinary web page or in Google products such as iGoogle, Google Desktop or Google Pages.

This gadget is configured by end-users by providing simple parameters such as the template ID or the web designer ID, which can be obtained reading the VPOET “designs list” wiki page. Fig. 7 shows this gadget in action using a given output template for the ontology component `foaf:Person`. The left-most window shows how this gadget can be inserted in a web page (and how it is configured), whereas the right-most one shows this gadget in a proprietary tool such as Google Pages.

Experimental evaluation

The evaluation of our work has focused on providing evidence that supports our principal aim concerning ease-of-use and expressiveness of VPOET. As we have shown, VPOET has two faces, the first one intended for web designers and the second one intended for web application developers. The first is provided through a web application and the second is provided through an HTTP API. Although both faces can be evaluated, we have focused on the first one. The HTTP API was informally discussed with some developers, and they did applaud the simplicity of the HTTP calls and the high functionality provided. Although there are techniques to measure the quality of object oriented APIs (Bansiya & Davis, 2002), we do not have any reference on quality metrics for HTTP APIs. Therefore, we trust in the informal evaluation of the VPOET API and focus our study in measuring quality features of web designers using VPOET, specifically usability clues and user satisfaction with the user interface.

In this context, usability is defined as “the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment” (Holzinger, 2005).

Description of the experiment

In order to have a real feeling about VPOET users’ perception, we selected fifteen postgraduate students to carry out this experiment. Some of them had basic technical background and knowledge about client-side web languages such

Table 4
: Parameters accepted in the HTTP GET/POST request.

Parameter	Value	Explanation/Example
action	renderOutput	Request a visualisation for the components object in the data source given in parameter source
	renderInput	Request a visualisation to request data for the component object from the user
object	prefix.class[.ver]	Example: foaf.Person
	prefix.relation[.ver]	Example: foaf.firstName
source (GET only)	URL	URL of the semantic data source
[provider]	ID	Identifier of the visualization provider. For example: user3.test
outputFormat	HTML	Default value
	XHTML	XHTML is used by WAP 2.0 mobile phones

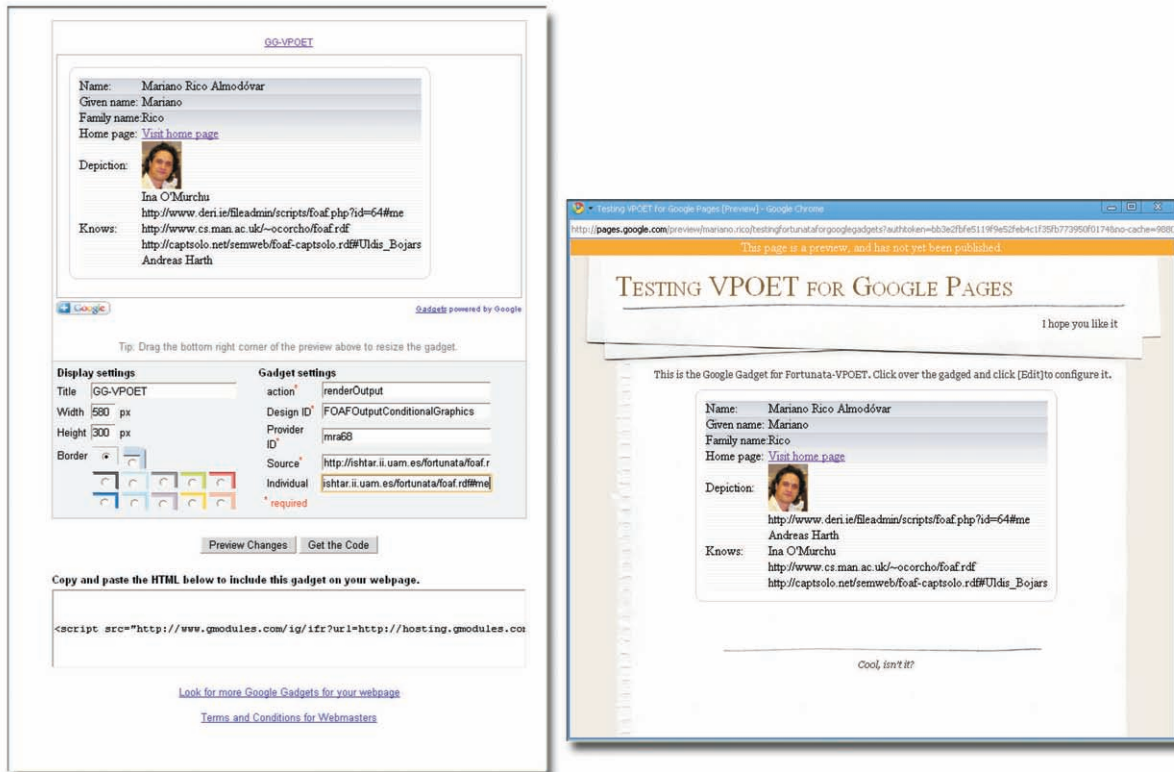


Figure 7. : Using GG-VPOET in different application oriented to end-users. Left: a personal page. Right: Google Pages

as HTML, CSS or Javascript. Some other had large experience in web design.

We provided all of them with a 30-minute talk about the task that they had to accomplish playing the role of web designers, giving a general overview of VPOET and OMemo. The talk used an online tutorial in which concepts such as class, instance, value, property, relation, were explained with practical examples. More advanced concepts such as conditional rendering or templates reusing also were covered by the online tutorial and were used by the participants. After this introductory talk, we explained them the main objective of the evaluation: creating a VPOET output template

for the class `Person` defined in the ontology FOAF (version 20050403). We made explicit that the time needed to perform this task was not going to be evaluated, so they could take as long as needed, but emphasizing that the objective was to create attractive visualizations of the ontology component specified. Links to existing semantic data sources were supplied in the wiki in order to help them test their designs.

Some of the common errors that they made were those related to the wrong display of multi-valued properties such as `Person.knows`, or to the incorrect visualization of missing values for properties such as `Person.title`. Once every user finished and tested his/her template, each participant

Table 5
: Topics in the skills questionnaire

ID	Client-side web technologies
Q1	HTML
Q2	CSS
Q3	DHTML (Javascript + DOM)
Q4	AJAX Basics (Javascript + asynchronous invocation + DOM)
Q5	Advanced AJAX Level 1 (AJAX basics+management of XML/JSON)
Q6	Advanced AJAX Level 2 (AJAX basics+API's usage = mashups)
Q7	Flash basics
Q8	Advanced Flash (XML data exchange and/or communication with Java/Others)

filled in a detailed questionnaire⁶ of 49 questions comprising different features such as (a) skills in client-side technologies, (b) usability concerning VPOET, and (c) user's satisfaction concerning the User Interface of VPOET. Most of these questions were based on standard questionnaires from Perlman site⁷.

Evaluation

Choosing the number of participants. To achieve a 95% confidence level for a given mean with error less than 1%, it is required to take 15 measurements at least (Efron & Tibshirani, 1986). This assumes intervals based on a normal population distribution for mean.

$$n = (z^* \cdot \sigma / m)^2$$

In this equation, z^* is the upper $(1 - C)/2$ critical value for the standard normal distribution (C is the confidence level), σ stands for the standard distribution (sample mean), and m stands for the margin of error.

Evaluation of skills in client-side technologies. The user's skills were calculated as the sum of the numerical values freely auto-assigned by each user (depending on his/her level of competence, from 0 to 5) on the client side issues described in the questionnaire. These issues are shown in table 5. An additional question to measure the level of competencies in semantic web technologies was removed from the results due to none of the participants had competencies in this topic.

Evaluation of usability. The questions about the usability of VPOET were taken from a standard test called "Practical Heuristics for Usability Evaluation" (Perlman, 1997). This test includes 13 questions ranging from 1 (bad) to 5 (good), which provides a useful measure of the user's perceived usability. The results of this test are shown in Fig. 9a and Fig. 8a. Fig. 9a shows that three types of users can be identified: those with basic skills (range 0-20, with 5 users), medium (20-40, with 7 users), and advanced (greater than

40, with 3 users). Fig. 8a shows the average value assigned by the participants to the questions related to usability in the questionnaire.

The analysis of usability was based on the average value assigned in this test, which was 4.1, with a standard deviation of 0.595. This shows high usability values for the whole range of participants, although the most skilled users assign slightly lower usability values. A possible explanation is that skilled users are more demanding, and this results in slightly lower evaluations, but even these advanced users provide high usability values. These results confirm that VPOET is a useful tool even for less skilled designers.

Evaluation of user's satisfaction. In order to evaluate the user's satisfaction concerning the VPOET user interface, we used a slightly modified version of the standard test "User Interface Satisfaction" (Chin, Diehl, & Norman, 1988). The standard version includes 27 questions, but it was reduced to 24 due to overlaps with the usability test described previously. Valid responses to these questions were positive integers ranging from 0 (not satisfied at all) to 7 (completely satisfied). The results are shown in Fig. 9b and Fig. 8b.

The average value for user satisfaction was 6.06, with a standard deviation of 0.53. Besides, the results show the dependency between the user's satisfaction and his/her skills. It is worth noting that the user satisfaction depends on the user skills in the same way that usability, that is, higher-skilled users assign a slightly lower value to satisfaction.

Treats to Validity. The next list summarises the context in which these results must be considered:

- The justification for the number of participants assumes intervals based on a normal population distribution for mean.
- The task proposed to the participants was the creation of a simple template, and many complex elements such as transitive relationships, axioms and constraints were not considered. Therefore the validity of this evaluation is restricted to the creation of presentation templates for simple ontology components.
- The effects of the underlying framework (Fortunata) impose several restrictions to the web application user interface. These restrictions have an effect in the web designer perception so that, these results must be considered in the context of Fortunata-based applications. A new version of VPOET, built with traditional web technologies, although more expensive in terms of development cost, could result in a tool better evaluated by web designers.

Analysis results. In summary, from the experimental evaluation with 15 participants, we can conclude that VPOET provides good average values for usability (8.2 in a 0-10 range) and user satisfaction concerning the user interface (8.7 in a 0-10 range) for a wide range of user competencies, what confirms our main hypothesis, that is, VPOET is a usable tool for web designers in a wide range of client-side technologies skills.

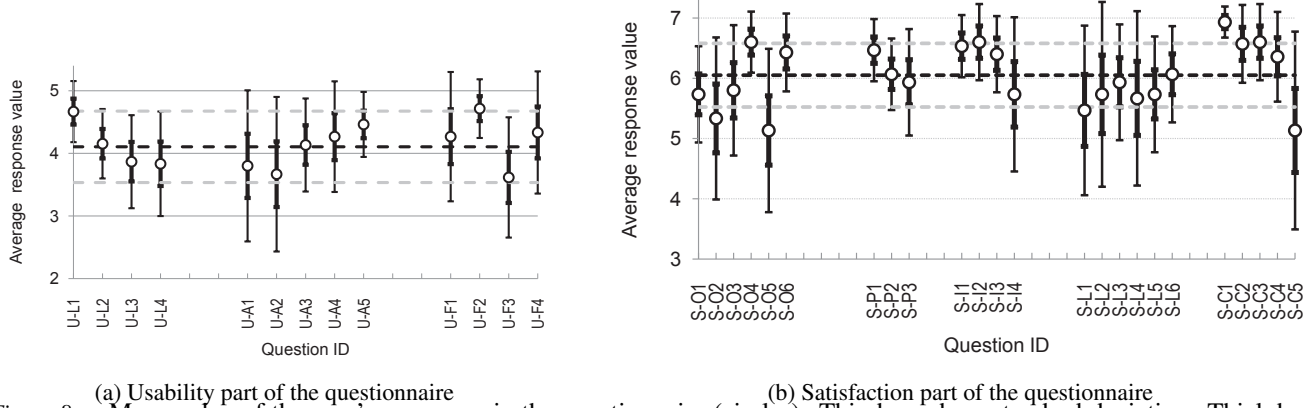


Figure 8. : Mean value of the user's responses in the questionnaire (circles). Thin bars show standard deviation. Thick bars show the 90% confidence interval of the mean. Dark dotted line shows the average usability value, and light dotted lines show standard deviation bounds.

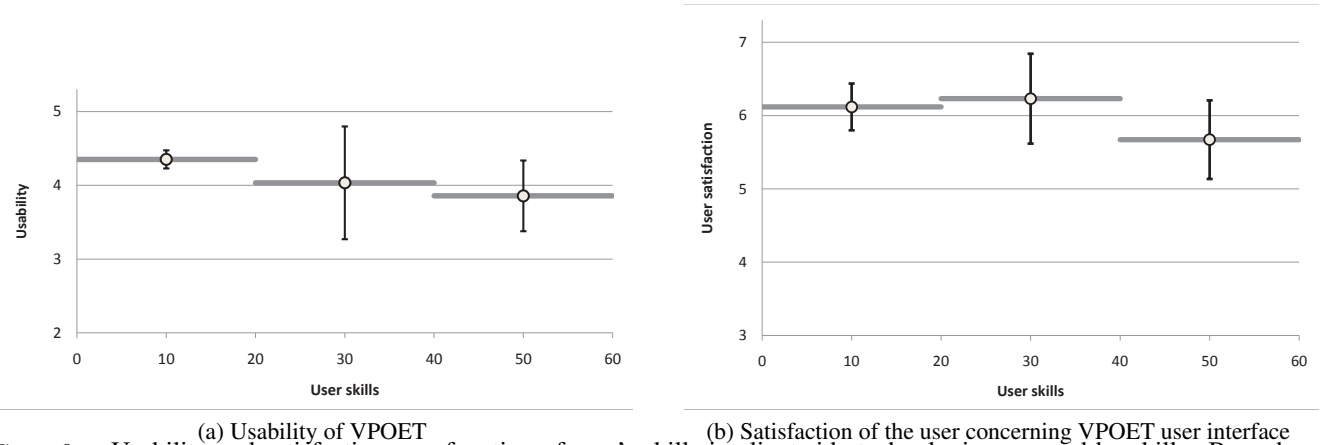


Figure 9. : Usability and satisfaction as a function of user's skills in client-side technologies grouped by skills. Bars show standard deviation.

Related Work

Our approach has some relationships with work performed in the context of semantic wikis, semantic portals, semantic pipes and semantic-web browsers, as well as End User Development (EUD).

Semantic Wikis (Oren, Delbru, Moller, Volkel, & Handschuh, 2006) are collaborative web applications that allow creating semantic data (e.g., RDF triples) and publish it in a wiki-like fashion, normally combined with natural language text. These systems require users trained in the annotation of natural language text and knowledge about the annotation terms to be used (ontologies terms). The wiki pages generated by these systems normally show both the natural language text and the semantic information in a predefined way. Although some systems provide users with templates, there is a one-to-one relation between an ontology term and a template. That is, the template for an ontology term is used to render all the ontology term individuals. VPOET externalizes this functionality, providing ways to access the tem-

plates and allowing a one-to-many relation between ontology terms and templates. Wikis can exploit VPOET templates easily (Rico, Camacho, & Corcho, 2009b).

Users can also handle semantic data in a *non-explicit* way, i.e. navigating through a web application as usual, but where semantic data are generated under the hood. This is the case for **Semantic Portals** (Lausen et al., 2005). These portals normally present data in table-based representations and request data to users by means of forms. Both of them are configured by portal developers with ad-hoc scripting languages. VPOET follows this non-explicit approach, allowing forms to request data from users (input templates) and a template based way to render semantic data (output templates) instead of scripting languages. The concept of semantic templates can be found in pOWL (Auer, n.d.), which provides users with a concept similar to VPOET templates, but much simpler; and Rhizome (Souzis, 2006), in which the ZML language is defined, but it is oriented to programmers instead of web designers with no skills in programming or semantic

web technologies, as is the case for PEGASUS (Macías & Castells, 2007), that also require technically skilled users.

A very recent approach close to both Semantic Wikis and Semantic Portals is the one from **Semantic Pipes** (Le-Phuoc, Polleres, Hauswirth, Tummarello, & Morbidoni, 2009). It provides a friendly web interface for users who want to handle semantic data, allowing the creation of semantically-enabled web applications as workflows that connect the inputs and outputs of different semantic data services. These applications can be contributed by others to be used under a philosophy very close to VPOET templates. However, unlike VPOET they are not focused on the presentation/request of data but only on its transformation. In this sense, VPOET is complementary to Semantic Pipes, and could be added to them as a visualization service of the intermediate or final results. A detailed comparison between VPOET templates and those from Semantic Media Wiki, Fresnel, and Rhizome can be found at (Rico, Camacho, & Corcho, 2009a).

In summary, VPOET aims at combining the advantages of semantic wikis (allowing collaborative features to create templates), semantic portals (allowing forms to enter user's data that will be converted to semantic data) and semantic pipes (allowing semantic data transformation), minimizing their drawbacks (uncontrolled edition of semantic data in semantic wikis, difficult transformation of user's input in semantic portals and lack of focus on presentation in semantic pipes).

Our approach can be also compared to some of the efforts in the Semantic Web area on the development of browsers that are able to show or that allow users navigating through semantic data. These browsers are normally called **Semantic-Web browsers** (Quan & Karger, 2004). These features are used by VPOET web designers and GG-VPOET end-users. Web designers take advantage of these features during the "testing loop" - i.e. when web designers have to test the output template against a set of semantic sources to discover pitfalls in the template. GG-VPOET users exploit these features each time the Google Gadget renders a given semantic data component due to the browsing facilities provided. It must be noticed that these "browsing facilities" are different from the ones provided by semantic-web browsers because browsing is limited to a given type of semantic individuals (the one of the selected output template). The extension to support the renderization of heterogeneous data will be studied in the near future.

Concerning the architecture of these browsers, two main groups exists: browser-centered and server-centered. Tabulator (Berners-Lee et al., 2006) belongs to the former group. DISCO (Bizer & GauSS, 2007) belongs to the latter group, and the same applies to VPOET templates. Browser-centered applications create some security problems, only solved by reducing the browser security level. Server-centered browsing does not have this limitation because browsing is indeed done by the server. Compared to DISCO, VPOET templates work with both URIs and URLs, however DISCO works only with URIs. Tabulator and DISCO render semantic data sources in a specific and non changeable way.

Many other semantic data visualization tools and lan-

guages exist, but are intended for professional users with high or medium skills on semantic web technologies. The most remarkable tools are IsaViz (Pietriga, 2001), which renders graphics in SVG ⁸ format, or ClusterMaps (Fluit, Harmelen, & Sabou, 2002). Some proposed visualization specifications are Graph-StyleSheets (GSS⁹), used by IsaViz; or Fresnel (Bizer, Lee, & Pietriga, 2006), used by Haystack (Quan & Karger, 2004) and PiggyBank (Huynh, Mazzocchi, & Karger, 2005). Fresnel requires advanced technical skills, so that it is not designed for web designers or common users. Although one of the features of PiggyBank is Semantic-Web browsing, it renders semantic data in a predefined way (determined by Fresnel) as the aforementioned semantic-web browsers. Compared to these systems, the main benefit of VPOET is that VPOET macros are simple and allow web designers with no technical knowledge about semantic web technologies a simple and efficient way to handle semantic data sources.

Another close related research field is End User Development (EUD), specifically in the area of mash-ups (Huynh, Miller, & Karger, 2008), where conceptual frameworks such as meta-design (Fischer & Giaccardi, 2006; Fischer, Giaccardi, Ye, Sutcliffe, & Mehandjiev, 2004) are well-defined and different types of users and developers have been identified. Although our approach is close to EUD, we consider a more traditional separation between the end users of our tools and the developers of semantic web applications.

Conclusions and future work

In this paper we have presented VPOET, a semantically-enabled web application designed to lower the adoption barrier of semantic data for web designers and common web application developers. VPOET allows web designers with no knowledge about semantic web technologies to create web templates capable of handling semantic data. VPOET *output* templates can render a given semantic component from a given semantic data source. VPOET *input* templates provide a web interface to request data from users, creating or updating a semantic data source. All these templates can be easily exploited by third parties by using simple HTTP calls. As an example of this, a Google Gadget has been created, which can be easily inserted into any user web page, providing a simple tool to handle semantic data sources.

VPOET have been built by using the Fortunata framework, a wiki-based infrastructure that joins the benefits of wiki systems and an ontology management system. As the creation of semantic templates is the cornerstone of VPOET, for the widest audience, from amateur to professional, an experimental evaluation have been carried out. The evaluation shows that VPOET is considered as highly usable by web designers in a wide range of skills. It also shows that users are satisfied with VPOET user interface in the same skills range.

Our future work will be mainly focused on adaptivity, which involves the adaptation of the user interface to a user's profile (device used, user impairments, and aesthetic preferences). Another research line is the creation of renders capa-

ble of displaying heterogeneous semantic data, and renders specialised in displaying dozens, hundreds or thousands semantic data. Some web designers have also identified missing useful macros that should be implemented, as well as some needs such as communication between templates that would allow communicating two GG-VPOET templates located in the same web page.

Footnotes

¹More details can be found at <http://ishtar.ii.uam.es/fortunata>.

²Data from <http://esw.w3.org/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics>, accessed in July 2010

³The VPOET tutorial is available at <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOETTutorial>

⁴The SPARQL protocol is available at <http://www.w3.org/TR/rdf-sparql-protocol/>

⁵The VPOET Google Gadget (GG-VPOET) is available at the Google Gadgets Directory (<http://www.google.com/ig/directory?q=vpoet&type=gadgets>)

⁶The questionnaire is available at <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=UsabilityAndUserSatisfactionOfVPOET>

⁷Web-based user interface evaluation with questionnaires is available at <http://oldwww.acm.org/perlman/question.html>

⁸SVG is available at <http://www.w3.org/Graphics/SVG>

⁹GSS is available at <http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html>

References

- Auer, S. (n.d.). *powl Ů a web based platform for collaborative semantic web development*. Available from <http://powl.sourceforge.net/overview.php>
- Bansiya, J., & Davis, C. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 4–17.
- Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., et al. (2006, November). Tabulator: Exploring and analyzing linked data on the semantic web. In <http://swui.semanticweb.org/swui06/program.html> (Ed.), *Proceedings of the 3rd international semantic web user interaction workshop (swui)*. Athens, Georgia, USA. Available from <http://swui.semanticweb.org/swui06/papers/Berners-Lee/Berners-Lee.pdf>
- Bizer, C., & GauSS, T. (2007). *Disco - hyperdata browser*. See <http://sites.wiwiwss.fu-berlin.de/suhl/bizer/ng4j/disco/>. Available from <http://sites.wiwiwss.fu-berlin.de/suhl/bizer/ng4j/disco/>
- Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data - the story so far. *International Journal On Semantic Web and Information Systems*, 5(3), 1–22.
- Bizer, C., Lee, R., & Pietriga, E. (2006). Fresnel: A Browser Independent Presentation Vocabulary for RDF. *Proceedings of the Second International Workshop on Interaction Design and the Semantic Web*, 158–171.
- Chin, J. P., Diehl, V. A., & Norman, K. L. (1988). Development of an instrument measuring user satisfaction of the human-computer interface. In E. Soloway, D. Frye, & S. B. Sheppard (Eds.), *Interface evaluations. proceedings of acm chi'88 conference on human factors in computing systems* (pp. 213–218). (June 15–19, 1988. Washington, DC, USA)
- Corcho, O., López-Cima, A., & Gómez-Pérez, A. (2006). A platform for the development of semantic web portals. In *Proceedings of the 6th international conference on web engineering (icwe2006)* (pp. 145–152). New York, NY, USA: ACM Press.
- d'Aquin, M., Baldassarre, C., Gridinoc, L., Angeletou, S., Sabou, M., & Motta, E. (2007). Characterizing Knowledge on the Semantic Web with Watson. In *Proceedings of the 5th international workshop on evaluation of ontologies and ontology-based tools (eon2007)*, *iswc/aswc* (Vol. 329, pp. 1–10). CEUR Workshop Proceedings. Available from <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-329/>
- d'Aquin, M., Motta, E., Sabou, M., Angeletou, S., Gridinoc, L., Lopez, V., et al. (2008, May/June). Toward a New Generation of Semantic Web Applications. *Intelligent Systems, IEEE*, 23(3), 20–28.
- Davis, I. (2005, 07). *Talis, web 2.0 and all that*. Available from <http://iandavis.com/blog/2005/07/talis-web-20-and-all-that?year=2005&monthnum=07&name=talis-web-20-and-all-that>
- Draft, W. W. (2009). *Owl 2 web ontology language* (Tech. Rep.). W3C. Available from <http://www.w3.org/TR/owl2-overview/>
- Efron, B., & Tibshirani, R. (1986, Feb). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science 1 (1)*, 54–75. Available from <http://www.jstor.org/stable/2245500>
- Fischer, G., & Giaccardi, E. (2006). Meta-design: A framework for the future of end-user development. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), (pp. 427–457). Springer.
- Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A., & Mehndijev, N. (2004). Meta-design: a manifesto for end-user development. *Communications of the ACM*, 47(9), 33–37.
- Fluit, C., Harmelen, F. van, & Sabou, M. (2002). Ontology-based information visualisation. In V. Geroimenko (Ed.), (p. 36–48). Springer Verlag.
- Holzinger, A. (2005). Usability engineering methods for software developers. *Communications of the ACM*, 48, 71–74.
- Huynh, D., Mazzocchi, S., & Karger, D. (2005). Piggy bank: Experience the semantic web inside your web browser. *LNCS. Proceedings of the International Semantic Web Conference (ISWC)*, 3729, 413–430.
- Huynh, D., Miller, R., & Karger, D. (2008). Potluck: Data mash-up tool for casual users. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4), 274–282.
- Lausen, H., Ding, Y., Stollberg, M., Fensel, D., Lara, R., & Han, S.-K. (2005, May). Semantic web portals: state-of-the-art survey. *Journal of Knowledge Management*, 9(5), 40–49. Available from <http://dx.doi.org/10.1108/13673270510622447>
- Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., & Morbidoni, C. (2009). Rapid prototyping of semantic mash-ups through semantic web pipes. In *Www '09: Proceedings of the 18th international conference on world wide web* (pp. 581–590). New York, NY, USA: ACM.
- Macías, J., & Castells, P. (2007, Jul). Providing end-user facilities to simplify ontology-driven web application authoring. *Interacting with Computers*, 19(4), 563–585.
- O'Reilly, T. (2005, 09). *What is web 2.0. design patterns and business models for the next generation of software*. Available at

- <http://oreilly.com/web2/archive/what-is-web-20.html>. Retrieved on 2009-08-03.
- Oren, E., Delbru, R., Moller, K., Volkel, M., & Handschuh, S. (2006). Annotation and navigation in semantic wikis. In *Eswc workshop on semantic wikis*. Available from <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-206/>
- Oren, E., Heitmann, B., & Decker, S. (2008). Activerdf: Embedding semantic web data into object-oriented languages. *Web Semant.*, 6(3), 191–202.
- Perlman, G. (n.d.). *Web-based user interface evaluation with questionnaires*. Available from <http://oldwww.acm.org/perlman/question.html>
- Perlman, G. (1997). Practical usability evaluation. In *Chi '97: Chi '97 extended abstracts on human factors in computing systems* (pp. 168–169). New York, NY, USA: ACM. (Los Angeles, USA. April 18-23)
- Pietriga, E. (2001). *Isaviz: A visual authoring tool for rdf*. see <http://www.w3.org/2001/11/isaviz/>. Available from <http://www.w3.org/2001/11/IsaViz/>
- Quan, D., & Karger, D. (2004). How to Make a Semantic Web Browser. In *International world wide web conference. proceedings of the 13th international conference on world wide web. session: Semantic interfaces and owl tools*. (ISBN:1-58113-844-X)
- Rico, M., Camacho, D., & Corcho Óscar. (2008). VPOET: Using a Distributed Collaborative Platform for Semantic Web Applications. In C. Badica, G. Mangioni, V. Carchiolo, & D. Burdescu (Eds.), *Intelligent distributed computing, systems and applications. proc. 2nd international symposium on intelligent distributed computing (idc'2008)* (pp. 167–176). Springer. (ISBN: 978-3-540-85256-8)
- Rico, M., Camacho, D., & Corcho Óscar. (2009a). Macros vs. scripting in VPOET. In *5th Workshop on Scripting and Development for the Semantic Web (SFSW2009) at the 6th Annual European Semantic Web Conference (ESWC)*. *CEUR online proceedings, Volume 449*.
- Rico, M., Camacho, D., & Corcho Óscar. (2009b). VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis. In *Fourth Workshop on Semantic Wikis: The Semantic Wiki Web (SemWiki2009) at the 6th Annual European Semantic Web Conference (ESWC)*. *CEUR online proceedings, Volume 464* (pp. 186–190).
- Rico, M., Camacho, D., & Corcho Óscar. (2010). A Contribution-based Framework for the Creation of Semantically-enabled Web Applications. *Journal of Information Sciences*, 180(10), 1850–1864.
- Rochen, R., Rosson, M., & Pérez, M. (2006). End user Development of Web Applications. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), (p. 161-182). Springer.
- Souzis, A. (2006). Bringing the wiki-way to the semantic web with rhizome. In M. Völkel & S. Schaffert (Eds.), *Semwiki2006, proceedings of the first workshop on semantic wikis*. CEUR-WS.org. Available from <http://www.ceur-ws.org/Vol-206/paper19.pdf> (Budva, Montenegro, June 12, 2006)