A contribution-based framework for the creation of semantically-enabled web applications

Mariano Rico , David Camacho , Óscar Corcho

Computer Science Department, Universidad Autónoma de Madrid, 28049 Madrid, Spain
Ontology Engineering Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Spain

A B S T R A C T

We present Fortunata, a wiki-based framework designed to simplify the creation of semantically-enabled web applications. This framework facilitates the management and publication of semantic data in web-based applications, to the extent that application developers do not need to be skilled in client-side technologies, and promotes application reuse by fostering collaboration among developers by means of wiki plugins. We illustrate the use of this framework with two Fortunata-based applications named OMEMO and VPOET, and we evaluate it with two experiments performed with usability evaluators and application developers respectively. These experiments show a good balance between the usability of the applications created with this framework and the effort and skills required by developers.

Keywords:
Contributely-collaborative systems
Wiki-based applications
Semantic web technologies
Semantic web applications

1. Introduction

In the last years a large number of ontologies has been made available on the Internet, and sources of semantic data have also had a large growth [4], especially in the context of the LinkedData initiative (see http://linkeddata.org), which has seen the emergence of a good number of SPARQL Endpoints [12]. However, this wealth of information still remains mostly hidden behind these SPARQL Endpoints, ontology libraries and ontology search engines, and are not used extensively in semantically-enabled web applications. This is due to the fact that, on the one hand, there is an increasing difficulty in the design of attractive and easily reusable web applications where a wide set of client-side technologies (e.g. HTML, Javascript, CSS, DHTML, Flash, or AJAX) and server-side technologies (e.g. ASP, JSP, JSF, .NET) need to be used, converting web designers in skilled programmers as pointed by Rochen et al. [15]. And on the other hand, the complexity of some semantic web technologies still represents a hard adoption barrier for any web application developer.

As an example, let us suppose that an application developer in a company is interested in creating a prototype of a semantically-enabled web application, so as to make an initial check of the feasibility of this approach and its validity for the type of problem that she is trying to solve. This developer has to master general-purpose server and client web technologies (which she may be probably already aware of) as well as semantic web technologies (which are less frequent among developers). Even a simple application, such as a small prototype, requires a large amount of competencies.

This paper presents Fortunata,[1] a framework designed to facilitate these application development tasks. Fortunata allows developers to build semantically-enabled web applications more easily, by reducing the required competencies in web and

semantic web technologies. The framework provides a programming library able to delegate: (1) the client-side presentation tasks to the wiki-engine on top of which it is built, and (2) the management and publication of semantic data by incorporating a simple set of ontology management functions.

Ours is not the first approach that aims at combining semantic and wiki-based technologies. In fact, this is something that has been covered, at least partially, by semantic wikis, semantic portals and, more recently, Semantic Pipes. However, there are some important differences between all these approaches:

- Semantic wikis [11,3] are focused on the collaborative creation [5,13,17] of semantic data (e.g., RDF triples) and its publication in a wiki-like fashion, normally combined with natural language text. Fortunata is instead focused on the collaborative creation of applications that exploit that data, by means of wiki-based plugins (also known as Fortunata-plugins or F-plugins). That is, semantic wikis are focused on content and are addressed to end users, while Fortunata is focused on applications and are addressed to developers.
- Semantic portals [8] are also focused on the collaborative creation of semantic data, although unlike semantic wikis they are more rigid and enforce the use of specific knowledge models (ontologies), which are converted into forms. These portals normally present data in table-based representations, which are configured by the portal developers with *ad hoc* scripting languages. Besides being focused on applications instead of data, Fortunata provides application developers with the ability to control the data flow between the user's form fields and the published semantic data.
- Semantic Pipes [9] focus on application developers who want to handle semantic data, as in Fortunata. They allow creating semantically-enabled web applications as workflows that connect the inputs and outputs of different semantic data services, and these applications can be contributed for others to be used. Unlike Fortunata, they are not focused on the presentation of data but only on its transformation.

In summary, Fortunata aims at combining the advantages of semantic wikis (using their easy-syntax for rendering information), semantic portals (allowing forms to enter user's data that will be converted to semantic data) and semantic pipes (allowing semantic data transformation), and 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). This is shown graphically in Fig. 1.

Any Fortunata-based web application comprises a set of plugins that: integrate semantic data from any existing source (including other Fortunata-based applications), allow its transformation in different manners, and/or provides presentations for semantic data. While traditional development centralises the source code, applications designed under this architectural paradigm are created in a decentralised way. That is, in traditional development, extending functionality of a semantic portal or wiki typically requires accessing the source code and compiling it, resulting in a new version of the application. Instead plugins allow members of a community to contribute to the creation of new functionality with a minimal degree of

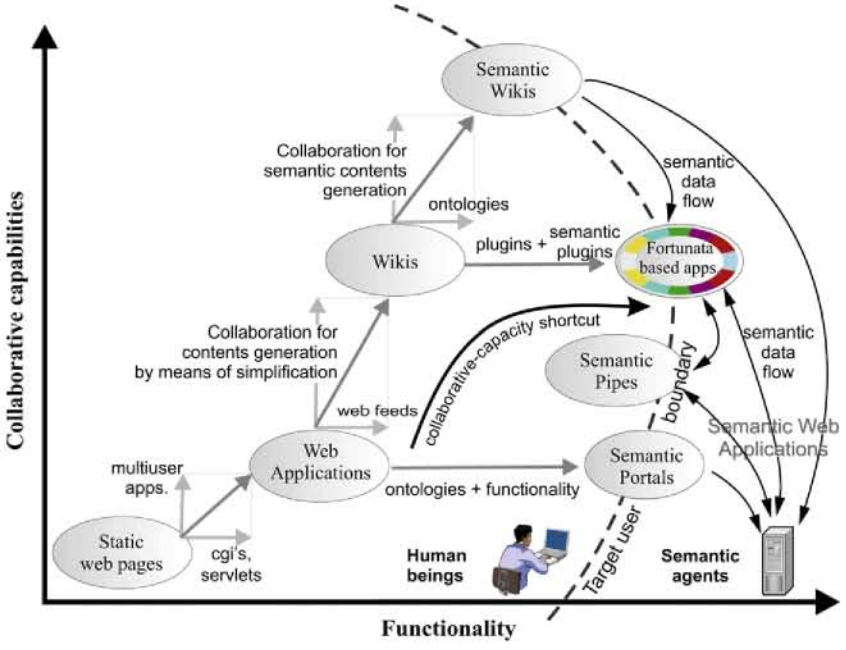

Fig. 1. Semantic web applications: semantic web portals, semantic wikis, and Fortunata-based web applications. The "target user boundary" line separates functionality for humans from functionality for semantic agents.

interdependence (e.g., they do not need to have access to the other plugins' code in order to compile them). When a developer has created and tested a new plugin, the source code is sent to the Fortunata-based wiki administrator. If the code is considered valid and safe, it is compiled and added to the Fortunata-based wiki engine, and is made available to any user or developer.

Following the aforementioned example, after a short initial training on Fortunata, the developer can create its prototype in a short amount of time (as shown in the evaluation section). The Fortunata API hides most error-prone details, allowing developers to focus their efforts in the provision of the required functionality. Obviously, more advanced applications will probably require more advanced competencies in semantic web technologies, but critical and error-prone tasks such as web presentation and publishing of semantic data do not need to be taken into account by the developer, since they are delegated to Fortunata.

To validate our approach, we have carried out two experiments. The first one evaluated the usability of our contribution-based framework at initial design stages (that is, in prototype phases), previous to the creation of more complex applications. A study using Inspection Methods [7] was carried out by usability experts. The results of this study allowed us to improve the system (overcoming some of the limitations of the selected wiki engine on top of which Fortunata is developed), as well as providing future developers with a usability guide specifically oriented to Fortunata-based developers. Following this guide, two applications have been created: OMEMO and VPOET, which are also described in this work. The second experiment was focused on measuring the benefits that the contribution-based strategy provides for developers when using Fortunata. Developers were requested to develop the same application, but were divided into two groups: one of them had to use traditional development tools while the another had to use Fortunata. Results of this experiment show that Fortunata-based developers required the use of fewer tools and needed less time to create their applications.

This paper is structured as follows. Section 2 describes Fortunata, focusing on how it has been built and on how it allows designing semantic web applications based on the contributively collaboration of developers. Section 3 describes OMEMO and VPOET, which are examples of the types of semantic web applications that can be created with this framework. Section 4 describes the two sets of evaluations carried out to check the validity of our approach, in terms of usability and development effort expenditure. Finally, Section 5 summarises the conclusions and future work.

2. Creating semantic web applications by developers contribution

This section provides a technical description of the architecture of Fortunata, the features provided by Fortunata, and a developer use case.

2.1. Fortunata features and architecture

As commented in the introduction, Fortunata aims at combining benefits from wiki-engines and ontology management systems, providing semantic web application developers with the following features:

- **Wiki-style presentation**. Developers do not require competencies in client-side technologies (e.g., HTML, CSS, or Ajax), but only wiki-style syntax, which can be learnt in a short period of time (around 10 min). This is because Fortunata provides developers with simple predefined methods to render a given wiki-text. In order to create applications, forms is another required feature. In this case the wiki-text can specify a form easily. This feature is provided by the wiki engine described later.
- **Semantic data management**. Ontology management systems such as Jena, Sesame, etc., normally provide developers with rather large and rich APIs that allow creating and handling ontologies and other semantic data sources. Fortunata only provides a simplified set of methods to handle these sources, getting an appropriate balance between the richness of the API and the amount of methods that need to be learnt by developers.
- A set of utilities. Limitations of JSPWiki such as data exchange between wiki pages, or a unified types set of user messages, are overcome by using a set of Fortunata utilities.

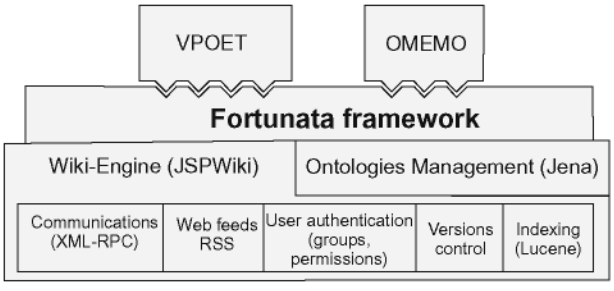

Fig. 2. Fortunata architecture.

In order to support these features, Fortunata has been created on top of the functionality provided by the JSPWiki wiki engine and by the ontology management library Jena, as shown in Fig. 2. Now we describe the different software components of this framework and justify their use for the creation of Fortunata.

JSPWiki (See http://jspwiki.org) is an open source Java-based wiki engine that allows the use of plugins to extend its functionality. Some plugins belong to the "core" library (maintained by the JSPWiki community), while other are "contributions" (maintained by the plugin creators). In 2006 only two wiki engines allowed plugins and forms: JSPWiki (java language) and Twiki (Perl language). As Jena requires java language, our decision was to use JSPWiki as wiki engine.

Besides extensibility, the core implementation of JSPWiki provides other features that can be used by Fortunata developers as well as by Fortunata-based application users. These benefits are briefly summarized as follows:

- Forms creation by means of specific wiki tags. JSPWiki wiki syntax allows an easy creation of forms which links buttons actions to plugins. This key relationship is described in Section 2.2.
- Web feeds (based in the standard RSS) provide users with a subscription mechanism. For example, a user subscribed to a wiki page (which may be generated from a semantic data source and hence represent a change in that semantic data source) will be notified whenever the subscribed page is changed.
- Common access control mechanisms allow managing permissions to see or modify a wiki page, provide user authentication mechanisms, group management, etc., which are common functions to be included in (semantic) web-based applications.
- Version control allows reverting any wiki page to its previous state. This feature encourages users to modify any wiki page with the guaranty that changes can be reverted.
- Link management provides mechanisms to identify "null" links (links pointing to non-existing wiki pages), "inverse" links (pages linking to the current page), or orphan links (pages not linked by any page).
- Indexing, implemented with Lucene, allows searches by keyword in all wiki pages, no matter whether they have been created manually or automatically from the semantic data sources.

Fig. 3. Comparison of the code required to store an instance. The box shows the code provided by a Fortunata-based developer in the "Hello Word" example application. The code in the background is provided by Fortunata.

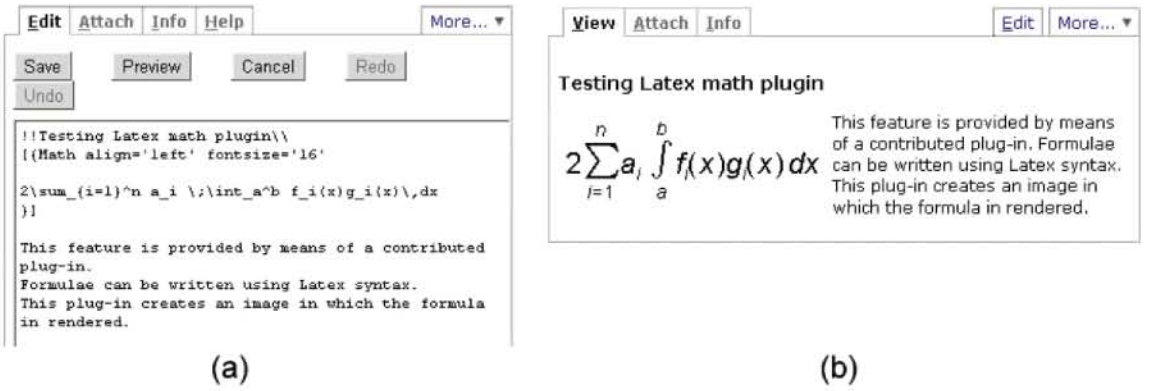

Fig. 4. Usage example of a JSPWiki plugin that generates an image from a Latex formula. (a) "Edition mode" shows how this plugin is invoked. (b) "View mode" showing the result.

Jena (See http://jena.sourceforge.net) is a Java library that provides developers with a programming environment to manage ontologies and semantic data. It can handle different ontology languages, such as OWL and RDFS, as well as different persistence and reasoning models. Fortunata hides this variety to developers by allowing the Fortunata wiki administrator to use a fixed set of options. For example, in the implementation used for our evaluations developers were using a Fortunata framework that was using a file-based persistence model (which is quite natural for wiki-based systems) and OWL DL ontologies.

With that configuration, the methods provided by Fortunata prevents developers from having to do tasks such as managing files, managing Jena models, etc. The developer only has to provide code for the creation of semantic data instances, as well as the code for the creation of the classes and properties. Fig. 3 shows the code provided by the developer to create an instance in the "Hello World" example provided in the developers tutorial.[2] Compare these three lines to the amount of code needed to provide this functionality (in this case provided by Fortunata).

2.2. Plugin development in JSPWiki and fortunata

In this section we describe first how JSPWiki plugins can be created and then we move into the creation of F-plugins.

2.2.1. Contributing functionalities in JSPWiki by means of plugins

Wiki plugins are pieces of code that extend the functionality of the wiki-engine, which is focused on allowing users a simple edition of wiki pages. Wiki plugins automate actions on a wiki page, and they present the result of such action in the "view mode". Examples of these core JSPWiki plugins are: TableOfContents, which generates a TOC from the wiki page contents, or ReferringPages, which finds and list all pages that refer to the current page, etc. JSPWiki "core" comprise 25 plugins, and the set of "contributed" plugins is currently around one hundred (by Nov. 2008).

Fig. 4 shows an example of how a specific plugin that is available in the system can be invoked from any wiki page (on the left part of the figure), and its corresponding result after the invocation with a specific set of parameters (on the right part). The invocation text of the plugin is between "[{" and "}]". This plugin contains arguments that specify a font size, an alignment, as well as a body containing a formula in Latex format. The result of this invocation is the wiki page in "view mode" that can be seen on the right. This plugin is automatically executed each time this wiki page is displayed in "view mode". The plugin execution output results in an image displaying the formula.

To create a JSPWiki plugin, developers only have to create a class implementing the interface `WikiPlugin` (see Fig. 5). This interface requires the implementation of the `execute()` method. This method will be invoked by the wiki-engine in the plugin execution, when a user is viewing the wiki page that contains it. Within that method developers have access to the plugin parameters and values, and have to include the code that performs the plugin operations (or the invocation to the corresponding method).

2.2.2. F-plugins development

In a similar fashion to how plugins are implemented in JSPWiki, the implementation of an F-plugin is done by means of a Java class (see Fig. 5) that implements the interface `WikiPlugin` (from the JSPWiki library). Additionally it extends the class `FortunataPlugin` (from the Fortunata library) which provides developers with useful methods (e.g. `renderWikiText()`) concerning forms management and rendering.

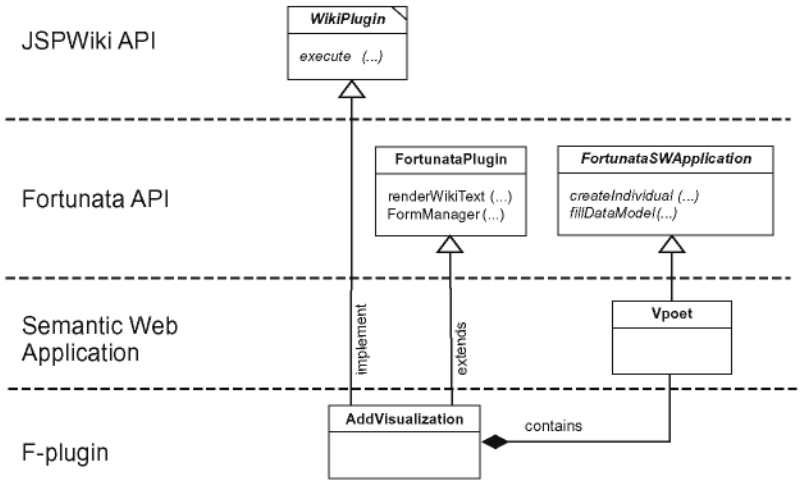

Fig. 5. Classes diagram of Fortunata-based applications. The layer "Fortunata API" shows the main class methods that a developer must implement. In this example, the F-plugin contains one instance of the class Vpoet. Abstract classes and interfaces are written in italics.

Fig. 5 shows the layers involved in the development of F-plugins. The upper layer is the API provided by JSPWiki, which provides the abstract class `WikiPlugin`. Below this layer, the Fortunata API provides a set of generic classes that can be exploited by specific application classes. This is the case of the semantic web application VPOET, which uses the class `Vpoet`, shown in the layer named "Semantic Web application". The last layer, named in the figure "F-plugin", is for specific application plugins. The class `AddVisualization` is an example of the kind of classes that exist in this layer, and how it is related to the other layers.

A semantically-enabled web application is represented by a class derived from the abstract class `FortunataSWApplication`, which provides developers with useful methods as well as force developers to implement the methods `createIndividual()` and `fillDataModel()` concerning semantics persistence. All the plugins in a Fortunata-based application share a semantic web application. In this example, the figure shows the class `AddVisualization`. This class is an F-plugin, and consequently it inherits the methods implemented in the base class `FortunataPlugin` and it is forced to implement three methods (one from the interface `WikiPlugin` and two from the class `FortunataPlugin`). This plugin contains an instance of the class `Vpoet`, which implements two methods from the class `FortunataSWApplication` concerning semantic data management.

The process to create and contribute an F-plugin is detailed in the upper part of Fig. 6, and follows the usual procedure in any plugin-based architecture. First, the developer must create the F-plugin locally (steps 1–3) and perform an adequate number of tests to check that it is working correctly (step 4). Then she must proceed to the publication (step 5) of the plugin source code and of the documentation about its usage. The bottom part of the figure depicts the process to create new functionality by reusing the initial functionality following a "contributively collaboration" schema. It comprises the following steps: installation of an existing F-plugin (step 6), reading and understanding of its associated ontologies (either by manually reading the OWL files, using any off-the-shelf ontology editor, or by means of OMEMO) in order to find the elements that must be added, removed or modified, or in order to decide whether a new set of ontologies has to be imported and used (step 7), local creation of the extended plugin (steps 8–10), local tests (step 11) and publication (step 12). The purpose of this detailed explanation is to show the low complexity of the plugin reuse and contribution process.

Table 1 summarizes the development tasks that are normally associated to the development of a typical semantic web application, and compares the skills that are required to perform these tasks when using a traditional development approach and a Fortunata-based approach. Traditional development requires more competencies (more development tools and roles) than Fortunata-based development. This is one of the main results of the comparison performed with real developers, which is described in Section 4.

3. OMEMO and VPOET: examples of Fortunata-based semantic web applications

This section illustrates how the Fortunata framework can be used to create two prototypical semantically-enabled web applications. These applications are not intended to be original or innovative, since similar types of applications are available in the current state of the art, but we aim at demonstrating that they are easy to implement and extend using our approach. OMEMO is focused on the HTML publication of ontologies (in a similar fashion to systems like OWLDoc (See http://www.co-ode.org/downloads/owldoc/)), and it is interesting as a case study since it exploits many features of the wiki infrastructure, such as orphan links or the simplicity of the wiki syntax. VPOET is focused on semantic data visualization, and especially

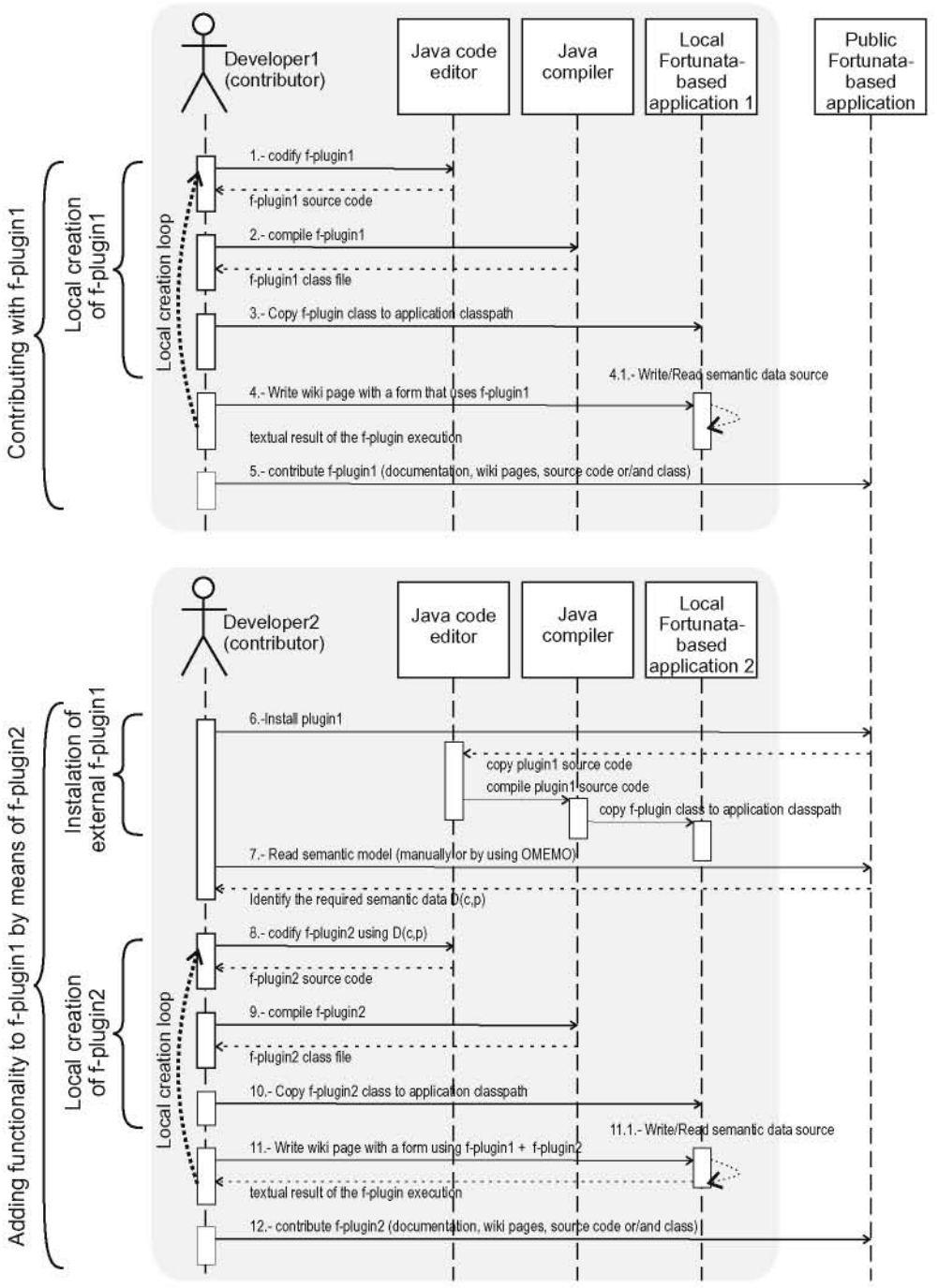

Fig. 6. Sequence diagram of a contribution. Developer1 creates and test (locally) F-plugin1. Once finished, this plugin is contributed. Developer2 takes advantage of this contribution and extends the functionality by means of F-plugin2, which is contributed as well.

exploits the forms provided by the underlying JSPWiki infrastructure and the ontology publication functionality provided by OMEMO.

3.1. OMEMO

OMEMO stands for "Ontologies for MEre Mortals". It is aimed at users with no previous knowledge about ontology languages, whom may find it difficult to understand the knowledge model that an ontology or set of ontologies is providing in languages like OWL or RDF Schema. By using OMEMO, users can browse and navigate through the components (classes,

Table 1
Comparison between traditional development of semantic web applications and the Fortunata-based approach.

Task	Traditional development	Fortunata-based development
Creation of pages	CMS (Contents Managing System)	Wiki
Creation of forms	HTML, CSS, DOM, AJAX	Wiki forms
Creation of web applications	JSP, JSF, .NET technologies	Wiki-engine
Permissions, authentication	Web server administrator competencies	Wiki-engine
Creation of ontologies	Jena	Fortunata
Creation of semantic data	Jena	Fortunata
Publication of ontologies and semantic data	Jena + Web server administration	Fortunata

properties and individuals) of any ontology that they upload to the system. When a new ontology is added to the system, a set of wiki pages is generated automatically, which provide a simplified vision of the ontology components, oriented to show the structure of the information, since the application is focused to non-technical users.

Therefore, OMEMO hides knowledge representation aspects well-known for ontology experts like "range", "domain", the differentiation between datatype properties and object properties, "functional properties", "inverse functional properties", etc.

It is worth noting that an ontology can have different versions, e.g. the FOAF ontology is available in at least two different versions: 20050403 and 20050603. The page-generation process results in a page for the whole ontology, which links to a set of pages for each class, property and individual. Fig. 7 shows a section of the wiki page generated for the class `Person` (version 20050403). Point ❹ indicates that other versions of the FOAF ontology exist and allows the access to those pages through this link. The value of the `interest` Property is `Document` ❸ as defined in FOAF, whereas the value of the `firstName` ❶ property is `Literal` ❷ as defined in the RDFS ontology. Whenever the RDFS ontology is stored in OMEMO, a link (solid underline) to the wiki page appears. Otherwise, the link will be underlined by a dotted-line (orphan link, i.e. a link pointing to a non-existing page). The numbered list in Fig. 8 shows the name of the wiki pages pointed. Finally, these automatically generated pages are not editable, resulting in a non-activated "Edit" button in the wiki page.

It is worth mentioning that users may also create manual pages pointing to these automatically generated pages, if they wish to add extra documentation to these ontologies (e.g., competency questions as identified in many ontology engineering methodologies, details of applications that are using them, etc.). Besides, all these pages (manually or automatically generated by OMEMO) are indexed by the Lucene engine that is part of JSPWiki; therefore, the search facilities provided by JSPWiki include any text available in the original ontologies plus any additional documentation.

A detailed explanation of the process that follows once the ontology's URL is provided by the user can be described as follows:

- A HTTP connection to the specified URL is established. The file containing the ontology is downloaded and stored in a temporal folder.
- The ontology is analyzed to detect which namespaces are used. This allows to link different ontology pages among them. For example, the ontology FOAF refers to the ontology RDFS in the definition of the FOAF:firstName property when the ontology FOAF declares that FOAF:firstName is a RDFS:Literal (see Fig. 7, first row in the table).
- Check 1: Conflict with prefixes. Blank prefixes (namespaces with no prefix defined), duplicated prefixes (ontology O1 defines prefix p for namespace n, and ontology O2 defines the same prefix for a different namespace), and overwritten namespaces (ontology O1 defines namespace n with the prefix p1, and ontology O2 defines namespace n with the prefix p2).

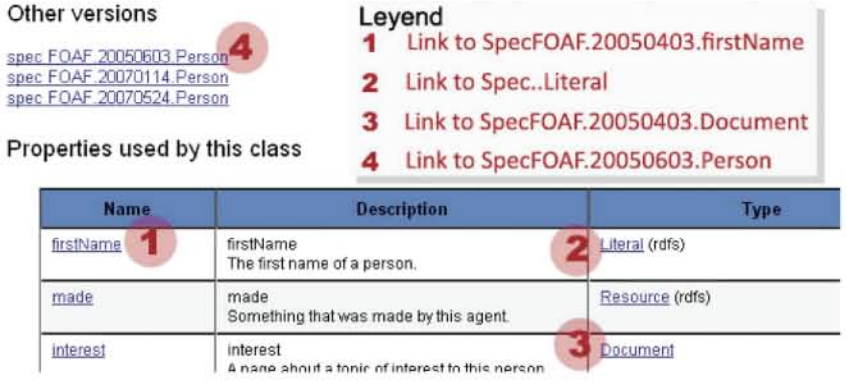

Fig. 7. Snapshot of the OMEMO wiki page for class Person (version 20050403) belonging to the FOAF ontology.

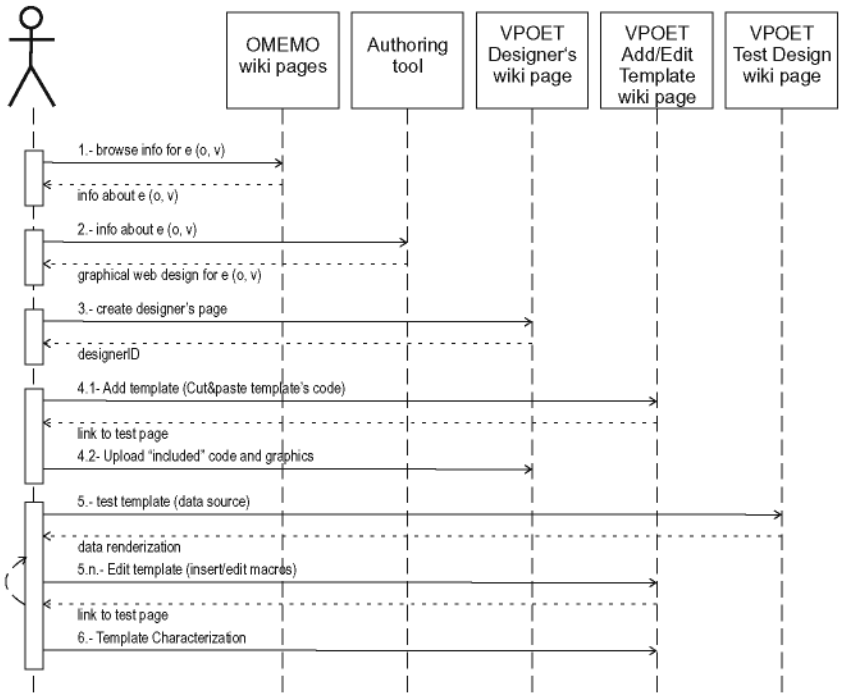

Fig. 8. Sequence diagram for VPOET users.

- Check 2: Ontology versions. O1 at URL1, and O2 at URL2, define the same namespace, but the content is different. For example, two versions of FOAF exist: 20050603 and 20050403. In version 603 there are properties such as FOAF:isPrimaryTopicOf and FOAF:birthday, that are missed in version 403.
- Check 3: Duplicated ontologies. The same ontology may have two or more different URLs. This is the case for Dublin Core, which involves two different URLs: `purl.org/dc/elements/1.1/` and `dublincore.org/2006/08/28/dces.rdf #`. Without this test, this would result in two different entries: dc.20030324 and dc.20060828, but the content would be the same.

Effective solutions to most of these problems have been implemented, resulting in a concrete schema (Spec.prefix.version.elem) for the generated wiki pages, as was shown in the numbered list in Fig. 7.

3.2. VPOET

VPOET [14] allows client-side web designers to create interactive templates for a given ontology component, not just to show semantic data (output templates) but to request data from the user (input templates). These templates can be created by any user, ranging from users with basic skills in client-side technologies, such as HTML or Javascript, to professional web designers.

This is possible because VPOET users only need to embed simple macros in the client-side web code, providing interaction templates for each ontology component. Information about each ontology component may be obtained, for instance, by reading wiki pages generated by OMEMO, although this is not compulsory. Once the interaction template is finished, its creator indicates the features of the template, specifying details such as the template type (input or output), the behavior in case of changes to the font size, sizes (preferred, minimum, maximum), the code-type provided (HTML, Javascript, CSS), or the dominant colors.

As any other Fortunata-based application, all the generated information is published as semantic data, so that it can be used by other internal or external applications. Although VPOET can be used by any user with basic skills in client-side web technologies, it has been created to let professional graphical web designers author attractive designs capable of handling semantic data. From a user point of view, this application is like any other web application, with form elements like text fields, radio buttons, or lists, as explained in the VPOET tutorial.[3]

The process to create a template starts by targeting an ontology component. In our example, the Person element from the FOAF ontology version 20050403 is selected.

The process to create an output template comprises the following steps (see Fig. 8):

1. Getting information about the structure of the targeted element for the given ontology and version, $e\ (o, v)$. That is, to know which sub-elements comprise the element. The visualization provider obtains this information by reading the wiki pages automatically generated by OMEMO. Fig. 7 shows a snapshot of the OMEMO wiki page for the FOAF:Person for this version.
2. Authoring a graphical design in which the semantic data will be inserted. Web designers are free to use their favorite web authoring tool.
3. Choosing an identifier (ID) to create a wiki page with that ID. This wiki page shows information about the VP and its templates.
4. The graphical design comprises a set of files: images, and client-side code such as HTML, CSS, or javascript.
   (a) The client-side code is copied and pasted into the appropriated VPOET form fields.
   (b) Image files or "included" files must be uploaded to the provider wiki page, or uploaded to any web server. In any case, the client code must point correctly to these files.
5. A test loop starts, that uses semantic data sources (typically external to VPOET) containing instances of the targeted element. A substitution process starts:
   (a) Absolute paths must be substituted by a specific macro.
   (b) In the location of the semantic values, specific macros must be inserted.
   (c) The design is tested against the test data sources.
   (d) This loop finishes when the design produces a successful visualization for all the semantic test data sources. For this example, a test source can be http://www.eps.uam.es/~mrico/foaf.rdf. Part (a) of Fig. 9 shows a small part (two instances of Person) of the web page generated by using a given template. Each instance can be rendered individually (circles 1a and 2a), as well as each source (circles 1b and 2c). Circles 1c and 2c show the data stored in the data source about these instances. Part (b) of Fig. 9 shows the rendering of the individual by using the same template. This results in a semantic web browser rendering each source, jumping from data source to data source, for a given template and ontology element.
6. The design is characterized by its creator, providing information about the template features, such as template type (input or output), colors, size policy, or font changes behavior.

Most of the effort required to create a template is in the test loop, especially in the insertion of macros. VPOET has been designed to let users reuse any other template. This is achieved by using: (1) rendering of an element specifying the template (of his/her own or not) and (2) links pointing to data that will render the destination element of a relation specifying the template as previous.

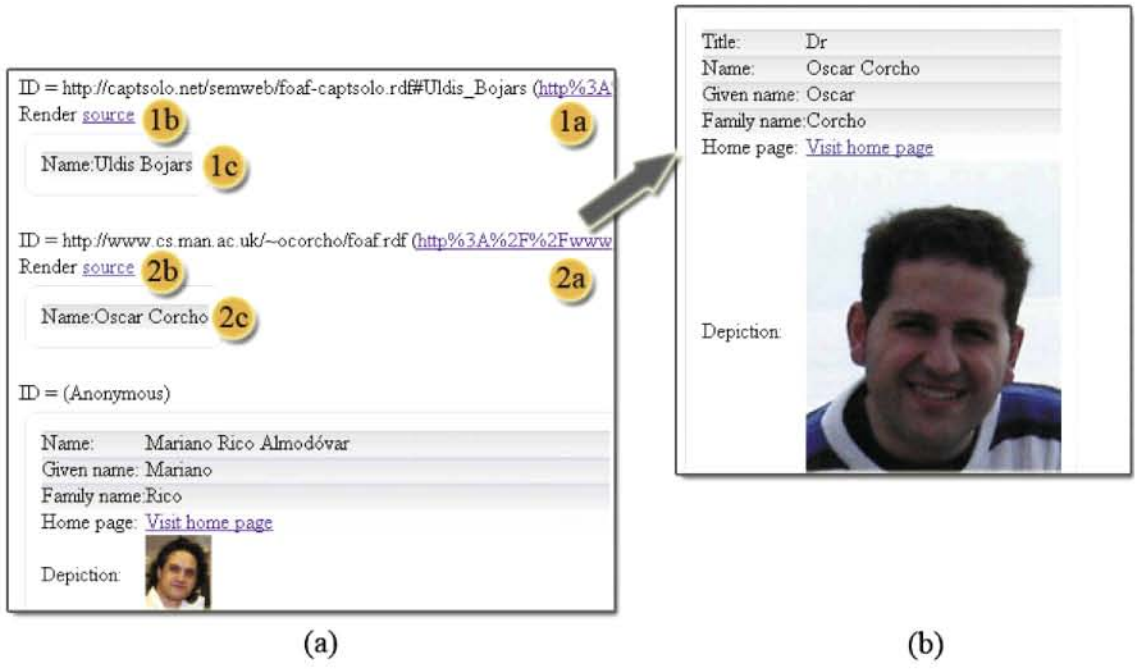

Fig. 9. Testing the example template against a given semantic data source.

4. Evaluating usability and collaborative features

Usability can be 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" [7]. In the context of semantically-enabled web applications, usability is still a major challenge [6]. Hence we have focused our efforts in trying to measure and understand how usable Fortunata-based applications are. For this purpose, we have analysed the usability of the two applications described in the previous section (VPOET and OMEMO), as examples of the common types of applications that can be generated with the Fortunata framework.

The literature on usability identifies two main groups of usability evaluation techniques [7]:

- Test Methods, which are normally applied to running applications (or at least with early-prototypes of these applications), and require real users.
- Inspection Methods, which are normally used during the design phase of these applications to identify potential usability problems, and do not require real users but only usability evaluators.

In our test we applied Inspection Methods, since our focus was to identify potential usability problems in applications during their design phase. Inspection Methods comprise different techniques, such as Heuristic Evaluation (HE), Cognitive Walkthrough and Action Analysis, with different levels of competencies required from the evaluators. The first one (HE) was selected due to the lower competencies required from the evaluators, what means that evaluators did not need to be usability experts, which are difficult to find. In fact, a set of 3–5 evaluators applying this technique can typically identify 75–80% of all usability problems [1]. Evaluators were asked whether the user interface followed some well-known usability principles, and the evaluation results were used to improve Fortunata and to create a "usability guide" for Fortunata-based developers.

The second type of evaluation was focused on the adequacy of the contributively collaboration feature of Fortunata. In this experiment, several developers with similar competencies on client-side technologies and a minimal background on semantic web technologies were selected and divided in two groups (identified respectively as "A" and "B"). A common goal was proposed for both groups, consisting in the incremental creation of three semantically-enabled web applications:

1. Implement a "Personal agenda", with a user interface to provide Name, mail and telephone;
2. Use the data from step 1 to add the necessary functionality to provide a user interface to schedule meetings for a given person;
3. Use the data from step 2 to add the functionality to display the people involved in meetings for a given date.

The first group ("A") had no training on the Fortunata framework, so they were free to use their preferred technologies and development tools. The only restriction for them was that they had to follow the usability "eight golden rules" (described in the next subsection). The second group ("B") received some training (in the form of a 20-min practical tutorial) on the Fortunata framework, and were requested to use this framework and to follow the "usability guide" created from the first experiment. The experiment ended filling in a complete questionnaire with quantitative and qualitative questions.

4.1. Experimental setup

4.1.1. Usability experiment

Five usability evaluators were recruited from our academic institutions. They were requested to evaluate independently early-prototype versions of VPOET and OMEMO. During the evaluation session the evaluator uses the applications several times, from different starting points, and inspects the interactive elements. They had to answer questions related to "the eight golden rules" [16], shown in Table 2.

The questionnaire comprised ten questions from [10]. This is a Likert scale-based questionnaire, i.e. "one based on forced choice questions, where a statement is made and the respondent then indicates the degree of agreement or disagreement with the statement on a 5 (or 7) point scale" [2], with seven possible values for the answers in the range from 1 (hard) to 7 (easy). Besides the choice, each question had an optional comments field. Table 3 shows the questions.[4] Additionally, each evaluator provided us with a list of recommendations to improve usability. These comments and recommendations were analyzed once the independent evaluations were carried out.

4.1.2. Collaboration experiment

For the second experiment, six students were selected from a "Semantic Web Technologies" master's course in Computer Science from one of our Institutions (Universidad Autónoma de Madrid). Three students were assigned to the "A" group (traditional developers) and three were assigned to the "B" group (Fortunata's developers). The questionnaire comprised two main blocks of questions related to complexity, collaboration and contribution. Some questions (Q3–Q6) used the previous Likert-based range values and the rest provides a numerical (continuous) value. These questions are shown in Table 4.

Table 2
Usability "Eight golden rules".

ID	Rule	Description
1	Consistency	There must be consistency in the actions, terminology (messages, menus and help windows) and graphics (colors, layout, and fonts)
2	Universal usability	Each user has a need; therefore the system must provide some facilities in order to transform contents. Not only impaired users, but differences between beginners-experts (beginners need explanations, the experts need shortcuts), or age ranges
3	Informative feedback	Each action in the system must produce a feedback. For common actions not very important, the answer must be small, but infrequent actions or important must produce a higher response
4	Dialogs	Dialogs must be designed to finalize something. Sequences of actions must be organized in groups with a start, middle, and final. For example, the concept of cart in web applications, with visualization for finished stages and pending stages
5	Errors prevention	The system must avoid that users make mistakes, but if the error is produced, the system must provide with a solution simple, constructive and specific
6	Undo	Allow users to undo actions in an easy way. Everything should be undo-able
7	Locus internal control	Support for the locus internal control. The expert users must have the sensation of controlling the tool. Users must start actions, not only respond to them
8	Memory load	Diminish the memory load in the short-term. Avoid multiple windows, codes, or complex sequences

Table 3
Questionnaire for usability evaluators of Fortunata-based applications.

ID	Question
1	*Visibility of system status* The system should always keep users informed about what is going on, through appropriate feedback within reasonable time
2	*Match between system and the real-world* The system should speak the user language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order
3	*User control and freedom* Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo
4	*Consistency and standards* Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions
5	*Error prevention* Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action
6	*Recognition rather than recall* Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate
7	*Flexibility and efficiency of use* Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions
8	*Aesthetic and minimalist design* Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility
9	*Help users recognize, diagnose, and recover from errors* Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution
10	*Help and documentation* Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large

4.2. Experimental results

4.2.1. Usability experiment

For the first proposed experiment, Fig. 10 shows the average values assigned by evaluators to each question and its standard deviation. The average usability value (discontinuous line) was 5.66 (in the range [1,7]), with std. dev. 1.16 (dotted-lines).

Question Q3 ("User control and freedom") had low values due to the lack of undo/redo features. Although the wiki-engine provides some kind of undo by means of control versions, reverting a wiki page to any previous state, the functionality implemented by F-plugins should support this feature more prominently. The current versions of OMEMO and VPOET do not implement this feature.

Many questions (Q2, Q3, Q5, Q8, Q9) had high consensus (low standard deviation) from evaluators (std. dev. = 0.55). The lowest consensus was for Q6 (std. dev. = 0.84).

The (qualitative) recommendations from the evaluators and their relationship to the "eight golden rules" are summarized in Table 5. Some recommendations were added to the Fortunata API (e.g. Rec1), other recommendations were generic guidelines (e.g. Rec2 and Rec3) that do not have a specific implementation in Fortunata. Finally, there was a group of recommen-

Table 4
Questionnaire for semantically-enabled web applications developers.

ID	Evaluation goal	Question
1	Complexity	Hours dedicated to create the application
2	Complexity	Tools used to create the application
3	Complexity	Level of difficulty to create the application
4	Collaboration	Level of difficulty to follow the usability "eight golden rules"
5	Collaboration	Level of dependencies on "previous code"
6	Collaboration	Level of difficulty to share your application's source code

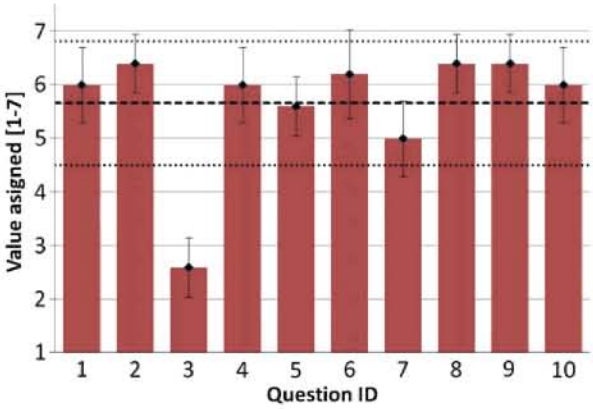

Fig. 10. Questionnaire for semantically-enabled web applications developers.

dations that can be achieved by "hacking" JSPWiki (e.g. Rec4 and Rec5), but other recommendations cannot be followed because the current wiki version (JSPWiki version 2.4) does not support them (e.g. Rec6 and Rec7) or because they are not implemented yet (e.g. Rec8). The "usability guide for Fortunata-based developers", comprising these recommendations can be found at http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=UsabilityRecomendations4Fortunata.

4.2.2. Collaboration experiment

From the questionnaire used in the second experiment (see Table 4), two kind of results can be considered, quantitative (Q1 and Q2 questions), and qualitative (Q3–Q6). Quantitative questions were designed to measure the development effort of Fortunata-based applications (see Fig. 11). The results for Q1 show that the number of development hours (sum of the three contributive steps) reduces about 40% using Fortunata against traditional technologies. The results obtained for Q2 show that

Table 5
Aggregated usability recommendations provided by evaluators.

ID	Recommendation	Rules involved	Solution provided by Fortunata API
Rec1	Execution messages must have a fixed location, font size and colors	#1, #2	Fortunata API provides a unified method for displaying executions messages, with three different warn levels ("success", "warning" and "error"), visually differentiated
Rec2	Use links to data pages/help to avoid remembering codes or identifiers.	#8, #7, #3	General recommendation with no effects on Fortunata API
Rec3	Forms should fit in a single page (avoiding page scrolls). Tabs usage is recommended for large forms	#5, #7, #8	General recommendation with no effects on Fortunata API
Rec4	Redirection by means of links	#6	There is no dynamic-redirection. Therefore, the execution of a plugin cannot change the wiki page. The solution is to place a link to the destination page in the plugin's execution message text
Rec5	Improve form features	#4	JSPWiki imposes some restriction to forms: (1) only buttons can fire plugins, (2) Javascript is not allowed, (3) There are no lists. Evaluators did not find severe usability problems
Rec6	Dynamic change of the skin	#2	The current version of JSPWiki supports skins, but it cannot be changed dynamically. Most skins support successfully changes in the font size
Rec7	Undo/Redo features	#6	The wiki-engine provides with a version control feature for wiki pages, allowing undo/redo for wiki contents. However, concerning F-plugins functionality, undo/redo must be implemented by the plugin's developer
Rec8	Advanced Editors (e.g. colored source code, auto fill text fields)	#5, #4	This functionality has not been implemented in JSPWiki yet by any contributor. These features would improve VPOET templates editor

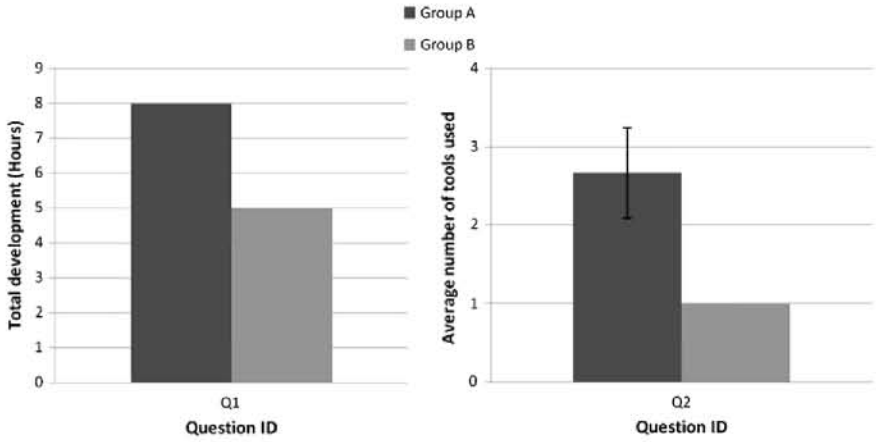

Fig. 11. First two questions of the developers questionnaire. Total development time (Q1) and average number of tools used by developers (Q2) for two kinds of developers: control group (A) and Fortunata-based group (B).

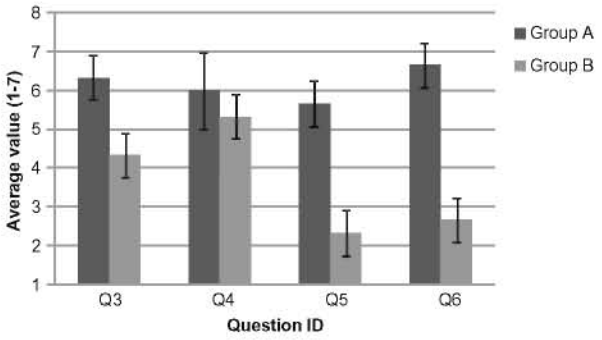

Fig. 12. Likert questions of the developers questionnaire.

the number of development tools reduces about 60% (the tool used for the three developers belonging the "B" group was just the Fortunata API, which produces a std. dev. = 0.0).

Fig. 12 shows the results of the rest of questions. In all of them the reduction by using Fortunata is between 10% (Q4) and 60% (Q6). Group A had uniform consensus for all the questions (std. dev. = 0.58) except Q4 (std. dev. = 1.0). Group B had the same uniform consensus for all the questions (std. dev. = 0.58) except Q2 in which the agreement was complete (std. dev. = 0.0).

5. Conclusions and future work

The work presented in this paper is focused on the provision of a simple and easily extensible programming framework that facilitates the creation of semantically-enabled web applications, while at the same time allows developers to contribute new applications that can be reused by others. The focus of our work is hence on: (1) simple and collaborative development environments, (2) facilities for reusing the contributed functionality, and (3) minimal dependencies between developers.

To achieve these requirements, Fortunata takes advantage of the functions provided by an open source wiki-engine and an ontology management library (Jena), simplifying their use by hiding their complexity to non-expert developers.

A usability study has been carried out in order to ensure that Fortunata-based applications fulfil basic usability requirements. The results of this study provided some improvements to the Fortunata API as well as a usability guide for Fortunata developers. Following this guide, an initial study with real developers has been used to measure quantitative (complexity of Fortunata-based development) and qualitative (contributively collaborative facilities) aspects of Fortunata. Results show good levels of usability values and a reduction on the software development effort.

As a proof-of-concept, two applications (VPOET and OMEMO) were created by using the Fortunata framework. The experiments show that this infrastructure is useful to implement real, reusable, and shareable semantically-enabled web applications. However, these applications also point out some of the main drawbacks of Fortunata, which are mainly due to the

limitations inherited from JSPWiki in forms (e.g. no lists, no javascript support, no advanced editors, no dynamic page redirection). Some of these limitations have been overcome and others will be addressed in future versions.

Future versions of Fortunata will overcome these drawbacks, providing users with better user interfaces. We will also incorporate some of the ideas obtained from the current work on Semantic Pipes, especially in what respects to the integration of data that is coming from external sources, on the declarative creation of transformational workflows for semantic data, and on the visualization providing better user interaction by means of integration with VPOET interactive templates. Finally, some of the work that we are currently doing is not focused on the platform itself but on one of the applications that were evaluated: VPOET. We are now proposing mechanisms to select the most appropriated template for a given user profile, considering aspects like its visual impairments, its interaction device, and its aesthetic preferences. This experiment, addressed at real user needs, will show the power of reusing different types of visualizations for the same semantic data, which is something that is already been shown in VPOET and that has only been exploited so far by semantic portals.

References

[1] K. Baker, S. Greenberg, C. Gutwin, Heuristic Evaluation of Groupware Based on the Mechanics of Collaboration, Lecture Notes in Computer Science 2254 (2001) 123–140.
[2] J. Brook, Usability Evaluation in Industry, first ed., Taylor & Francis, 1996 (11 Jun 1996 Ch. SUS – A quick and dirty usability scale, pp. 189–194).
[3] M. Buffa, F. Gandon, G. Ereteo, P. Sander, C. Faron, Sweetwiki: a semantic wiki, Web Semant 6 (1) (2008) 84–97.
[4] M. d'Aquin, E. Motta, M. Sabou, S. Angeletou, L. Gridinoc, V. Lopez, D. Guidi, Toward a new generation of semantic web applications, Intelligent Systems IEEE 23 (3) (2008) 20–28.
[5] T. Gruber, Collective knowledge systems: where the social web meets the semantic web, Journal of Web Semantics 6 (1) (2008) 4–13.
[6] T. Heath, J. Domingue, P. Shabajee, 2006. User interaction and uptake challenges to successfully deploying semantic web technologies, in: Proceedings of the Third International Semantic Web User Interaction Workshop (SWUI2006), Fifth International Semantic Web Conference (ISWC2006), 2006.
[7] A. Holzinger, Usability engineering methods for software developers, Communications of the ACM 48 (2005) 71–74.
[8] H. Lausen, Y. Ding, M. Stollberg, D. Fensel, R. Lara, S.-K. Han, Semantic web portals: state-of-the-art survey, Journal of Knowledge Management 9 (5) (2005) 40–49.
[9] C. Morbidoni, D.L. Phuoc, A. Polleres, G. Tummarello, Previewing semantic web pipes, in: S. Bechhofer, M. Hauswirth, J. Hoffmann, M. Koubarakis (Eds.), Proceedings of the Fifth European Semantic Web Conference (ESWC2008). No. 5021 in LNCS, Springer, 2008, pp. 843–848.
[10] J. Nielsen, R.L. Mack, Usability Inspection Methods, Wiley & Sons, Inc., 1994 (Ch. 2: Heuristic Evaluation).
[11] E. Oren, R. Delbru, K. Moller, M. Volkel, S. Handschuh, Annotation and navigation in semantic wikis, in: ESWC Workshop on Semantic Wikis, 2006.
[12] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, Tech. Rep., W3C Recommendation, 2008.
[13] D. Richards, A social software/Web 2.0 approach to collaborative knowledge engineering, Information Sciences 179 (15) (2009) 2515–2523.
[14] M. Rico, D. Camacho, Corcho Óscar, VPOET: using a distributed collaborative platform for semantic web applications, in: C. Badica, G. Mangioni, V. Carchiolo, D. Burdescu (Eds.), Proceedings of the Second International Symposium on Intelligent Distributed Computing (IDC'2008). No. 162 in Studies in Computational Intelligence, Springer, 2008 pp. 167–176.
[15] R. Rochen, M. Rosson, M. Pérez, End user Development of Web Applications, Springer, 2006 (Chapter 8, pp. 161–182).
[16] B. Shneiderman, C. Plaisant, Designing the user interface:strategies for effective human–computer interaction, Adison-Wesley, 2005.
[17] J. Yong, W. Shen, Y. Yang, Special issue on computer-supported cooperative work: techniques and applications, Information Sciences 179 (15) (2009) 2513–2514.