A Platform for the Development of Semantic Web Portals

Oscar Corcho
University of Manchester
School of Computer Science
Oxford Road, Manchester, United Kingdom
+44(0)1612756821
Oscar.Corcho@manchester.ac.uk

Angel López Cima(*), Asunción Gómez-Pérez
Universidad Politécnica de Madrid
Facultad de Informática. Campus de Montegancedo, s/n.
28660 Boadilla del Monte, Madrid, Spain
+34913367467
{alopez, asun}@fi.upm.es

ABSTRACT
A Semantic Web portal is a Web application that offers information and services related to a specific domain, and that has been developed with Semantic Web technology. For the time being, the main difference with respect to a traditional Web portal is based on technological aspects: traditional Web portals are based on standard Web technology (HTML, XML, servlets, JSPs, etc.); semantic portals are based on that technology plus the use of Semantic Web languages like RDF, RDF Schema and OWL. This paper describes a unifying architecture for both types of portals, based on the MVC paradigm, which is implemented in the ODESeW framework. ODESeW has been used successfully in the development of a set of portals for the management of European R&D projects and for the management of research groups.

Categories and Subject Descriptors
H.3.5 [Online Information Services]: Data sharing, Web-based services

General Terms
Management, Documentation, Experimentation, Security,

Keywords: Semantic Web portal, ODESeW, Intranet.

1. INTRODUCTION
An application framework is a set of libraries or classes used to implement the standard structure of a type of applications. Having reusable code in a framework means that much time is saved for developers, since they do not need to rewrite large amounts of standard code for each new application developed.

Application frameworks are also defined as software components that model and solve a specific type of problem, providing a set of extensible and configurable components and an engine to coordinate and execute them. These components will be extended in a specific problem by developers.

Both definitions make it clear that application frameworks aim at reducing the amount of effort needed for developing and maintaining software, as part of the philosophy of rapid application development (RAD).

In Web application engineering, there are many open-source and commercial frameworks available for the development of standard Web applications. Among them we can cite frameworks like: Turbine [25], Struts [23], JSF [11], Millstone [15], Wicket [27], etc. All these frameworks aim at easing the development of Web applications, by providing reusable, configurable and extensible components that are commonly used in such applications.

In Semantic Web application engineering there are fewer frameworks available, due to the fact that this area is less mature. And in many cases we cannot talk yet about frameworks, but about specific applications that have been developed from scratch or by reusing some existing components, but without the notion of comprehensive application development frameworks. Some of these emergent frameworks are: the KAON portal [12], OntoWebber [10], Rhizomik [18], Duontology [4], etc.

Most of the applications developed in this area are the so-called *knowledge portals* or *semantic portals*. They refer to ([13][22]) knowledge-based Web sites for corporate access to information and applications. In [13] they are defined as Web applications that "*provide the means to select, classify and access, in a semantically meaningful and ubiquitous way, various information resources (e.g., sites, documents, data) for diverse target audiences (corporate, inter-enterprise, e-marketplace, etc.)*."

Though both Web and Semantic Web application development frameworks provide interesting features for the rapid application development, they also share the fact that they are not specialised for the development of domain-specific applications. That is, they only contain generic components that can be included in Web and Semantic Web applications and these components have to be extended by developers when they want to create a specific application in a domain. The places where the framework can be extended are known as extension points [3].

From this comment it seems interesting to have also reusable extensions or configurations of such application development frameworks for those types of applications that a set of developers normally have to create. In this paper we are interested in showing how we have configured and extended a Semantic Web application development framework for the creation of the Intranets and Extranets of several European R&D projects. The application development framework that we have used is ODESeW, whose earlier version was already described in [5].

This paper is structured as follows. Section 2 describes the main features of ODESeW. Section 3 describes how ODESeW has been used to create the Semantic Web sites of R&D projects in the European Union context, with examples extracted from the Web sites of Esperonto [6], Knowledge Web [14] and OntoGrid [16]. Finally, section 4 concludes and outlines future work.

2. The ODESeW Semantic Web application development framework
ODESeW (Semantic Web Portal based on WebODE) was first described in [5] as a front-end of the WebODE ontology

ICWE'06, July 11-14, 2006, Palo Alto, California, USA.
ACM 1-59593-352-2/06/0007.

engineering workbench that could be used for the automatic generation of knowledge portals for Intranets and Extranets, using the same assets and knowledge and providing different functions in each case:

- If the knowledge portal is used as an Intranet, corporate users can insert and update content as content providers, browse the content that they have inserted or that other corporate members have inserted there, and perform searches and queries on that content. The ontologies is used for indexing and searching the knowledge asset more efficiently.
- If the knowledge portal is being used as an Extranet, external users will only edit very restricted parts of the content stored in the portal, and browse, query and search only the content identified as public content by the content providers.

Besides the aforementioned content provision, visualization, and access functions, ODESeW provided management services to select the ontologies to be used as the basis for the portals, to configure read/write permissions over the information, etc.

This first version of ODESeW provided generic views for content visualisation: hierarchical concept trees, instance lists, instance attribute and relation visualisation and edition functions, and RDF, RDF Schema and DAML+OIL export functions. These views were generic enough to show all the information needed for providing and accessing content. Therefore, the setup and maintenance of a knowledge portal required a very low management effort, which mainly consisted on selecting the set of ontologies to be used in the portal, and extending and modifying the default user accounts and read/write persmissions on the ontology components for both the Intranet and the Extranet.

While the main advantages of having generic views are low setup and maintenance efforts, there is an important tradeoff with respect to its extensibility in order to deal with institution-specific requirements. In this sense, creating specific views that were needed in some of the portals was a time-consuming task, since the developer needed to have in-depth knowledge of the API provided by the portal, of its internal architecture and of the interaction between the different components.

The current version of ODESeW is not so much focused on the development of Web portals based on Semantic Web technology, but on the provision of a framework for building Semantic Web applications. Hence, it provides reusable but easily extensible views for different types of applications and users. With this approach we keep the idea of having a low setup and maintenance effort while allowing the creation of personalised views with a view model and view composition model, and the specification of navigation and visualisation models for different types of users.

In the following sections we describe the main components of this Semantic Web application development framework.

2.1 Architecture Outline

The architecture of ODESeW 2.0 is based on the design pattern Model-View-Controller (MVC) [7], which is currently widely used for developing Web applications.

This pattern or architectural paradigm divides functionality among three types of objects (the view, the model and the controller, as shown in figure 1), which are involved in maintaining and presenting data to minimize the degree of coupling between the objects. One of the advantages of using this paradigm is that the clearly separation of these three objects makes it very useful for developing applications where the same information has several visualisations. The objects are described as follows:

- A model represents business data and business logic or operations that govern access and modification of this business data. The model notifies views when it changes and provides the ability for the view to query the model about its state. It also provides the ability for the controller to access application functionality encapsulated by the model.
- A view renders the contents of a model. It accesses data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller.
- A controller defines application behavior. It dispatches user requests (button clicks, menu selections, form input texts, etc.), also known as user gestures or actions, and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a Web application, they are HTTP GET and POST requests to the Web tier. A controller selects the next view to display based on the user interactions and the outcome of the model operations.

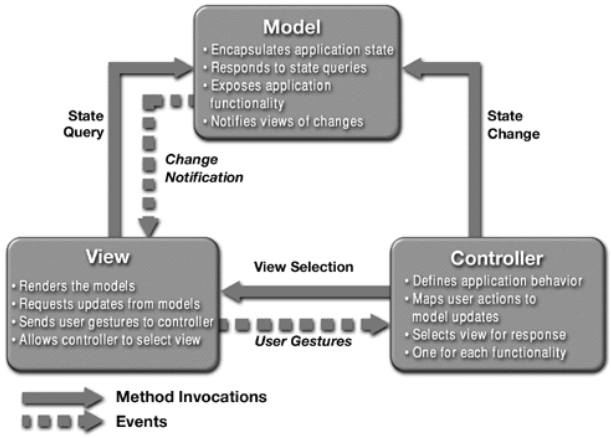

Figure 1. Model-View-Controller pattern (from [20]).

Let's see now how each of these components are implemented in the context of ODESeW, and which functionalities are available for the design of Semantic Web Intranets and Extranets.

2.1.1 Data Model

The ODESeW Data Model contains the information that the knowledge portals show and the information that the portals use for their management functions. It is divided in two submodels, as shown in figure 2: the Domain Model and the User Model.

All these submodels are coordinated by the Data Model Manager, which receives state change requests from the controller and is used to feed the queries made by the views. All the state change requests are filtered by the Permission Layer, which takes into account the user permissions and profile.

We now describe each of the components of the ODESeW data model.

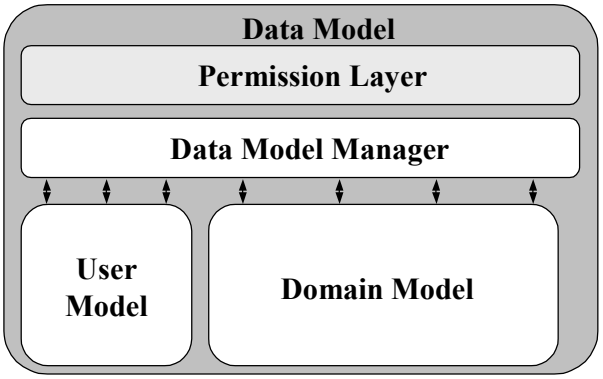

Figure 2. The ODESeW Data Model and its components.

2.1.1.1 Domain Model

The Domain Model consists of a set of domain ontologies, which are the backbone of the information shown in the Semantic Web applications generated with ODESeW.

Ontologies are accessed using WebODE. As an ontology server, WebODE provides the functionalities required by ODESeW, namely the retrieval of ontology components and the storage and retrieval of concept and relation instances from distributed ontologies. Besides, it provides support for some additional functions, such as ontology export to RDF, RDFS and OWL.

The knowledge model of WebODE (and hence of the ODESeW Domain Model) is described in [2]. Many types of ontologies can be imported in WebODE by means of its ontology import services (e.g., ontologies implemented in RDF, RDF Schema, DAML+OIL, OWL, UML, etc.).

2.1.1.2 User Model

The User Model contains user profiles, normally organised according to the group and role to which they belong, and which are tightly related to the application domain. This model is represented using an ontology about users (and optionally groups and roles), which contains authentication information about users, information about the roles that they play in their organisations, permissions to access specific parts of the information, to present differen types of views, etc.

There are two types of permissions assigned to individual users, groups or roles, each of them with different granularity:

- Read permissions. They involve an information flow from the domain model to a view. These permissions are assigned over instances and instance attributes and relations.
- Write permissions. They involve an information flow from the controller to the domain model. These permissions are assigned over concepts and ontologies.

In both cases, permissions are assigned by the application administrator.

2.1.1.3 Data Model Manager

The Data Model Manager is the module in charge of coordinating the access to the domain and user models. As we will describe in section 2.3, it is also in charge of coordinating the actions of the External Information Gateway when a user makes a request that triggers the execution of an action over an external resource in order to fill in information from the domain model.

This module is generic, since the only pieces of information that need to be configured in order to use it in an application are: the ontology that specifies user profiles, the ontologies that are included in the domain model, and the reference to the connectors to external information sources, as described in section 2.3.

2.1.1.4 Permission Layer

The Permission Layer filters all the requests to the Data Model Manager, according to the read and write permissions of the Web application user (either if it is in the context of the Intranet or of the Extranet).

The process followed to assign user profiles is similar in the Intranet and Extranet applications. In the case of the Intranet, the user will authenticate in the application home page by providing its user name and password, and its user profile will be determined according to the information available in the user model. In the case of the Extranet, the user will be given a default profile (*guest*). In both cases a session is maintained for the user during its visit to the application.

Once the user profile has been determined and the session has been created, the user profile information (with its information about permissions) will be used by this component whenever a request is received by the data model manager.

2.1.2 Views

As aforementioned, the main purpose of views is the renderisation of the content available in the data model. In Web applications like the ones that ODESeW is used for, views are also known as Web pages or documents.

ODESeW provides a set of reusable views and mechanisms for Web developers to ease the communication with the Data Model, so as to retrieve information from the ontologies stored in it. Two groups of views can be identified in ODESeW:

- Views for human agents. They are focused on the generation of HTML documents that Web browsers in the client platform can render and show to the user.
- Views for software agents. They are focused on the generation of documents in Semantic Web languages like RDF, RDF Schema and OWL.

The first group of views are described using state-of-the-art Web application design technology, such as JSP (Java Server Pages [19]), and Tag Extension [19] in conjunction with EL (Expression Language [19]) and JavaBeans [8]. These technologies allow defining reusable operations for accessing information stored in the domain ontologies.

Some of the default views available in the platform are:

- Upper Term View. It is a generic view for rendering different types of ontology components (concepts, attribute types, attributes, relations, and instances). This view is highly reusable and has a low maintenance, but reduces the usability of the views, since it does not provide application-specific information about the term that it is rendering (for instance, if we are rendering an instance of a book, it might be interesting to show not only the list of attributes that the book has, but also to provide an image for the book coverpage, the attributes presented in a specific order, etc.).
- Term View. It is a generic view for rendering the information of an object, adapted to the ontology component that it is displaying (a concept, an attribute type, an attribute, a relation,

or an instance). This view is less reusable and normally has to be extended by the application developer for different types of concepts, as aforementioned.

Other default views available are: ontology concept trees, instance lists, etc.

The ODESeW platform contains a set of engines (encapsulated in the Controller) that are able to interpret the views described at run-time so as to render appropriately the information coming from the Data Model, according to the desired views.

2.1.3 Controller

The ODESeW Controller is responsible for several functions, and is at the core of the platform. It receives the user request, which contains the actions to be performed, and completes or checks the request with the information model in the Data Model (including both the domain model and the user model). Then it reads and executes the navigation and composition model, described below, and returns the next view that should be rendered for the user.

We describe the ODESeW Navigation and Composition Model, and then the steps followed by the Controller to execute actions.

2.1.3.1 The Navigation Model

The navigation model represents the navigation of a user through the application. This model is explicitly separated from the design of views so that changes in the navigation do not affect the implementation of views. Besides, it allows representing declaratively the navigation of a user, enabling in this way an easy study of the behaviours of the user of an application.

The navigation model is a directed named graph where nodes represent views and edges represent navigation actions from one view to another.

- **Nodes** have 2 attributes: "*precondition*" and "*view URL*". The first one specifies preconditions to allow the execution of a view and the second one specifies the location of the view. The view can be abstract, what means that it cannot be rendered directly and has to be specialised by other views.
- **Edges** identify actions that can be performed by the user from a view. Besides redirecting users from a view to another, edges are attached to a task execution: instance edition, instance removal, message sending, etc. Edges can be concatenated to perform different tasks in one navigation step.

The navigation model also allows describing specialisation/generalisation relations between two views (defined with the *subclass-of* relationship). A view is a specialisation of another if it visualises the same content as the parent view but providing more specific visualisation items. For instance, a default view may be used to render attribute values and for other more specific types of values, such as e-mail addresses, URLs, image files, sound files, video files, etc., other more specific views can be created.

Figure 3 shows an example of a navigation model with 9 views defined and several types of actions and specialisation/generalisation relations defined between them.

2.1.3.2 The Composition Model

The composition model is similar to the navigation model, though its rationale is different. Basically, it allows including a set of views inside another one and is normally used when complex sets of information have to be presented at once to the user.

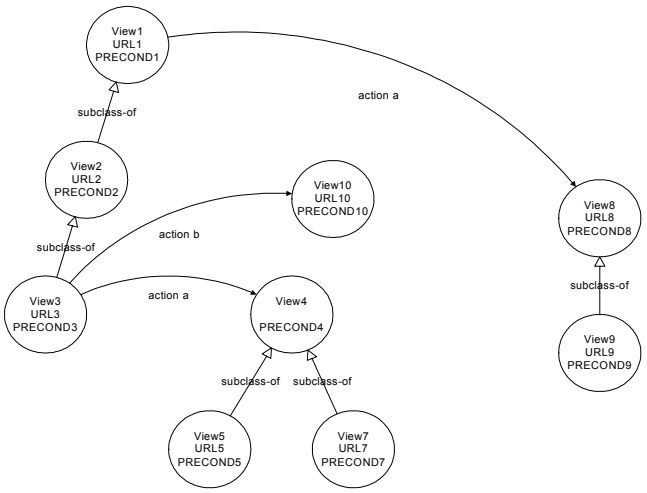

Figure 3. Example of a navigation model.

One common example of the use of the composition model is the main view of the application (aka the application home page). Here the developer normally includes a header, a footer and an index. All these components can be different views that are composed to create a unique one.

The elements used in the composition model are the same as those for the navigation model: views are represented as nodes, with the attributes "*precondition*" and "*view URL*"; views can be specialised with other views; and actions are represented as edges. The only constraint in in the type of actions that can be represented in this model, which only consists in the action of inclusion of a view inside another.

2.1.3.3 Controller Execution

Actions received by the Controller contain two elements: task and control flow operation. The task is the specific operation to be performed, while the control flow operation specifies what to do after the execution of the task.

Developers can use any of the default tasks provided by the ODESeW platform or create new ones, either from scratch or by reusing and extending any of the default ones. The following default tasks are available:

- *sewView*. It is an empty action that renders the view specified in the user request by redirecting users to it.
- *sewRemove*. It deletes the set of concept and relation instances specified in the user request.
- *sewEdit*. It updates or creates the set of concept and relation instances specified in the user request.
- *sewSearch*. It searches for a set of concept and relation instances that satisfy the query.
- *sewRouter*. It is an empty action (that is, it does not perform any action on the data model), which is used to execute another action from a list specified in the user request. These actions have a guard condition, and the sewRouter task selects the first one whose guard condition is satisfied.
- *sewLogin*. It authenticates a user and loads in the user session his/her profile.

With respect to control flow operations, there are four available:

- *Forward*: the user request is concatenated to another action or view.

- *Redirect*: the user request ends and a new user request starts. This new request consists in showing another view or performing another action.
- *Include*: the execution of a new action or view is included in the original view or action, so that it will be performed later.
- *Empty*: the execution ends without any more control flow actions. In fact, a view is actually defined as a rendering action, optionally followed by other additional include actions, and which has an empty control flow at the end.

When a user requests an action from a view, the Controller executes the navigation model, following these steps:

- Identify the view from which the user request is originated, and find it in the navigation model.
- Find the requested action in the source view. The action can be defined explicitly in the source view or in any of its ancestor views.
- Select the target view for the requested action. In the navigation model, an action applied to a specific view may have several target views, and at least one of them has to be selected. To perform this selection, the Controller verifies whether the precondition of any of the target views specified in the action is satisfied given the request parameters. If no precondition is satisfied, an exception raises.
- Find whether any of the specialisations of the selected target view is also valid. Once the controller found a valid candidate view, it will try to find another one among its specialisations. To do this, the Controller checks the preconditions of the view specialisations. If any of them is satisfied, that view is a new valid candidate view and the Controller repeats this step with its children views, until a valid view does not have more specialisations or none of the preconditions of its specialisations are satisfied. The last valid candidate view is the final target view.

Let us see an example based on the navigation model presented in figure 3. Let us assume that the user requests the ***action a*** from the view ***View3***, and that the parameters of the request satisfy the preconditions ***Precondition4***, ***Precondition8*** and ***Precondition9*** and do not satisfy the preconditions ***Precondition5*** and ***Precondition7***.

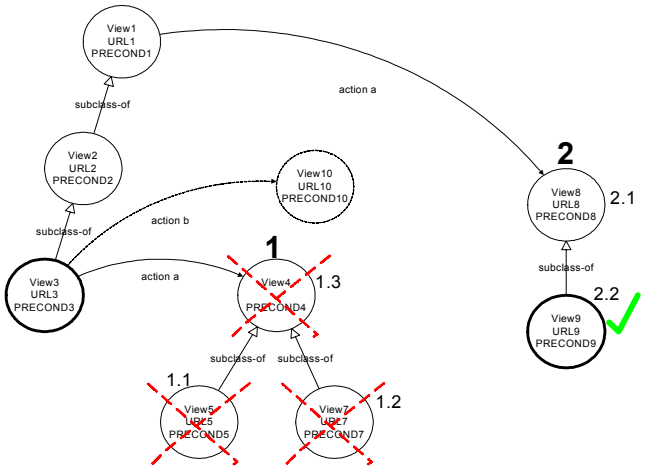

Figure 4. Example of a navigation model execution.

First, the Controller finds the source view (***View3***). Taking into account that the user wants to perform ***action a***, the possible candidate views are the ***View4*** and ***View8***.

The first candidate to be checked is ***View4***. However, ***View4*** is abstract, so the Controller has to check the preconditions of its specialisations (***View5*** and ***View7***). Neither of them satisfy the preconditions, so ***View4*** nor its specialisations are valid target views. Hence, View4 is discarded by the Controller and the next candidate view (***View8***) is analysed. The ***Precondition8*** is satisfied, hence the ***View8*** is a valid candidate view. Then, the Controller starts looking for its specialisations (***View9***). The precondition of ***View9*** is also satisfied and, since ***View9*** does not have specialisations, the final target view for the execution of ***action a*** from ***View3*** is the ***View9*** (see figure 4).

2.2 ODESeW Extensions to the MVC Architecture

ODESeW provides two extensions to the MVC design pattern: the External Information Gateway, which is used to feed the data model with information available in external information sources, hence improving the interoperability of ODESeW applications with other similar applications; and the Notification Service, which is used for sending asynchronous messages about changes in the data model.

The complete ODESeW architecture is depicted in figure 5, including these two modules.

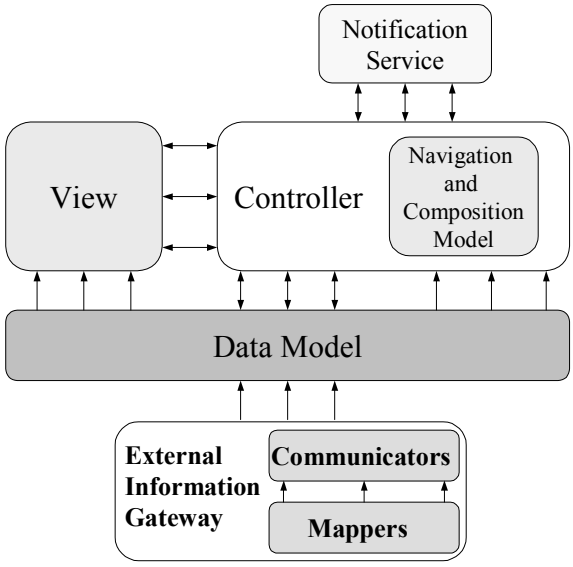

Figure 5. ODESeW extended-MVC design pattern.

2.2.1 External Information Gateway

This system collects information from external sources and maps it to the domain model, regardless of the communication protocols (HTTP, FTP, CORBA, Web services, etc.) and formats (relational databases, texts in natural language, XML documents, RDF files, etc.) needed to access such information.

The External Information Gateway works as follows. Some of the domain ontologies used in the application may have connectors attached. These connectors identify the information sources that can provide information about the instances of some of their classes or relations, in case these are needed in a user query. When the Data Model Manager receives a query that involves instances from such concepts or relations, the External Information Gateway is contacted so that the information from the information source is provided as if it was available inside the internal ODESeW data model.

The External Information Gateway supports two information provision models. They are used depending on the characteristics of the information sources (availability, cost model, processability of information, dynamicity of information, etc.), and are specified by the application developer. They are the following:

- **Runtime provision model**. The external information source is accessed on real time when the application requests information from it. This model is used with information sources that provide a low latency between the request and the response and where the information changes frequently.
- **Cached provision model**. The external information source is accessed periodically and the information retrieved is stored locally. When the user requests this information it is provided from the local store. This model is used with information sources that provide a big latency between the request and the response, and when the information does not change frequently.

To retrieve information from external sources, the External Information Gateway uses two types of components:

- Communicators. They provide an abstraction layer on top of the different communication protocols that may be needed to access information sources.
- Mappers. They deal with representation and semantic mismatches between the external information source and the domain model.

2.2.1.1 Communicators

Communicators connect to the external information source using a specific protocol (HTTP, FTP, etc.), and provide also an abstraction layer over the access interface (CORBA, Web services, etc.). They are also in charge of abstracting the provision model used for accessing the information available in the external source (runtime or cached).

ODESeW includes by default two communicators (for the protocols HTTP and FTP), using the runtime and cached provision models. Application developers can also create new communicators and plug them easily into the system.

2.2.1.2 Mappers

Mappers transform the information from the external source into concept and relation instances of the domain ontologies, overcoming any mismatches that may occur between the origin and target models.

A mapper is configured with the following elements:

- Input. It identifies how to obtain the information from the external information source.
- Output. It identifies the result that will be obtained as a result of the execution.
- Mapping. The mapping is the function that transforms the input to the ouput.
- Mapping Engine. The mapping engine interprets the specified mappings and transforms the input to the output.
- Instance consistency checker. The mapper, optionally, can have a consistency checker that resolves conflicts with other information available in the domain model of the application.

The mapper is coordinated with type of communicator that it uses to get the information from the external source. Hence if the communicator retrieves the information on runtime, then the mapper also executes on runtime, while if the communicator caches the information, the mapper also creates a cache of the output result of the mapping process.

Mapper chains can be created to improve reusability of mappers that perform simple mapping functions. In these chains the output of a mapper is used as the input of the next one in the pipeline.

Finally, off-the-shelf mapping engines and consistency checkers can be used or new ones can be created from scratch. Mapping engines are normally reusable across applications. The following set of default mapping engines are provided in ODESeW:

- A XSLT engine.
- An engine to transform RDF triples to RDF/XML documents.
- An engine to transform HTML documents to XHTML documents.
- An extended XSLT engine that allows executing RQL queries [26].

Web applications can extend these mapping engines or create new ones. On the contrary, consistency checkers are less reusable across applications and no default consistency checkers are provided in the framework.

It is very important to remark that the framework is in charge of coordinating the different components and each application can implement its own components or reuse the default ones.

2.2.2 Notification Service

The Notification Service is an asynchronous system that can be used to send and receive messages based on the subscribe/notify model. Any system can subscribe to any set of topics from those available in the system (the list of topics is dynamic, that is, it can be updated at any time by any of the systems that make use of it). When a system sends a message regarding a specific topic, all the systems subscribed to that topic receive a notification with this message. The service uses Java Message Service [20].

In ODESeW the Notification Service is widely used by the Controller, which sends messages whenever a user visits a view, edits or removes an instance, or when a message has to be sent as part of an action. Three default topics are available in ODESeW:

- *NewInstance*. It is used when a user creates an instance in the domain model. The message sent to the Notification Service contains: the user inserting the instance, the timestamp and the instance itself.
- *UpdateInstance*. It is used when a user modifies the value(s) of any of the attributes or relations of an instance from the domain model. The message sent to the Notification Service contains: the user updating the instance, the timestamp, the old instance with the old attribute and relation values and the new instance with the new attribute and relation values.
- *RemoveInstance*. It is used when a user removes an instance from the domain model. The message sent to the Notification Service contains: the user removing the instance, the timestamp and the instance removed.

3. Applying ODESeW for the development of R&D projects Web portals

Different versions of the ODESeW platform have been used in the development of the Web portals of diverse European R&D projects.

The most relevant were already pointed out in the introduction (Esperonto [6], Knowledge Web [14] and OntoGrid [16]). Furthermore, the platform has been also used in part of the AgentLink III portal [1] and in the web site of the Ontological Engineering Group at UPM [17].

All these applications have been developed as knowledge portals with a twofold function: first, to serve as an Intranet for the compilation of all the knowledge generated, and second, to serve as an Extranet for the dissemination of the project results.

3.1 R&D project Web portal design

The following tasks were perfromed for the development of the portals: model the domain, identify the types of users, identify the content to publish, identify the site map of the portal content and implement the visualisation. In the following section we focus on the domain and navigation models.

3.2 Domain of the R&D projects

The R&D project domain had been described with five ontologies: *project*, *documentation*, *person*, *organization*, and *meeting*. These ontologies describe respectively R&D projects and their structure, documents that are generated in a project, people and organizations participating in it, and meetings (administrative, technical, etc.) held during a project lifecycle. Each portal extends these ontologies according to its characteristics.

According to the ODESeW data model, all the ontologies belong to the domain model except for the ontology about persons, which is partially used to define the user model of the application.

An important aspect that makes ODESeW more robust than other existing technologies and platforms for Semantic Web application development is that changes in the ontologies are automatically incorporated in the application. Therefore, even if these ontologies do not normally change due to the fact that they have been already used for several years and projects, any change in the ontology can be made at any time.

3.3 Users of the R&D Projects

In projects like Esperonto and OntoGrid, which involve a smaller amount of partners, there are three different types of user profiles that are considered for the generation of specific types of views and navigation models. These are:

- Guest users. These are users that are external to the project and only visit its Extranet.
- Project partners. These are users that belong to any of the organisations involved in the project and that work on it, accessing its Intranet and performing tasks of visualisation and edition of instances.
- Project officer. This is a user profile for which the most important information to be shown are the project deliverables and other administrative documents.

Each of these user profiles has different navigation models and have also different levels of permissions in the portal. For instance, the guest user cannot insert instances in the portal, cannot access the restricted and private deliverables, can only access the PDF versions of public deliverables, etc.; the Intranet users have full read and write permissions for all the concepts, instances and attributes in the five ontologies; and the project officer can access all types of deliverables but cannot insert nor edit instances in the portal.

In larger projects like the Knowledge Web network of excellence, the number of user profiles is larger. Besides the previous user profiles we identify different types of area managers (Industrial Area Manager, Educational Area Manager, Research Area Manager, and Management Area Manager). Each of these managers have different navigation models as well, and different sets of permissions so that they can read and modify different types of documents.

3.4 Content and navigation

The key content to be visualised in all projects is: list of partners, list of persons working for each partner, list of deliverables, workplan and related events. Besides, the home page shows direct links to the most recent and more requested information.

Most of the content accessible from the home page is indexed by ontology concepts and consists of links to instances. For example, the list of partners is the list of instances of the concept *Organization*, the list of deliverables is the list of instances of the concept *Deliverable*, etc. Therefore, the default navigation model has been created according to concepts and instances. In the navigation model, there is a node reprensenting the home page and a navigation edge, *viewTerm*, to an abstract node *Term*, which is specialised in *Concept* and *Instance*. These two nodes are also specialized in different nodes according to the information that can be requested: Organization, Person, Deliverables, Workpackages, etc. Besides, from the *Concept* view there are links to a list of instances and from the *Instance* view there are links to related instances.

To satisfy the need to have the information available for software agents, these views have another link to the implementation in RDF of the visualised list of instances. This link is modelled as an edge from the node *Term* to the abstract node *RDF*, which has three specialisations depending on whether it has to provide all ontology instances, all concept instances or only an instance.

Figure 7 shows the top nodes of this navigation model and, as a representation of specilization, the nodes that visualized a instance and a list of instance of concept Organization.

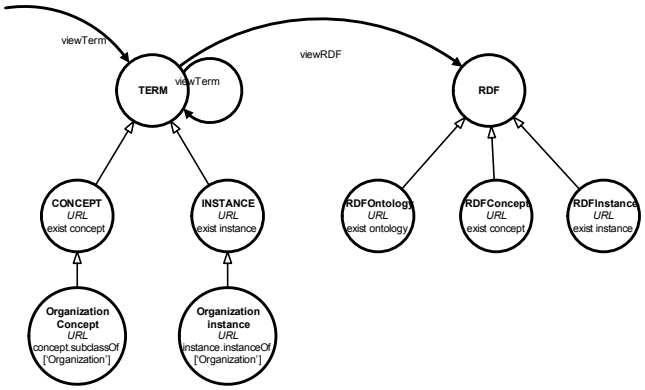

Figure 6. Reusable fragment of a navigation model.

Besides the previous views, which are used throughout the portal, there are also specific views that are predefined for different types of users. For instance, there is a specific view for the project officer that generates a report with all the deliverables produced in the project. And there is a specific view for guest users that generates a report with all the public deliverables produced. These views are specified in the model as specialized nodes in which the precondition of each node is a user type check.

4. Conclusions and Future Work

We have presented the main features of the Semantic Web application development framework ODESeW, focusing on how it extends the Model-View-Controller design pattern and how it eases the interoperation with other similar applications, by means of the External Information Gateway.

Besides, we have described how this framework has been configured for the creation of the Web portals of several EU R&D projects, facilitating many of the tasks done by project partners and providing a common view for all of them, according to the usual structure that they share.

We have not provided a comparison with other similar approaches. Only a few project Web portals are based on similar technology (the OntoWeb portal [21] and the SWWS portal [24], among others). The main differences between these portals and ODESeW are related to the fact that interoperation with other information sources cannot be easily performed with them and that the Intranet features are less advanced.

Among the features that will be included in the next versions of ODESeW, we can cite the following:

- Improvements in user authentication. We will incorporate single sign-on capabilities, using not only username and password pairs, but also user credentials (digital certificates, Intranet user profiles, etc.). With these we aim at making it easier to integrate the project Intranet with each of the organisation's Intranets.
- Inclusion of localisation functions, so as to have more types of external users than the default user guest.
- Provision of a more powerful set of mappers and communicators, including state-of-the-art ontology mapping techniques for a better interoperation with other knowledge-based portals, as well as other communication protocols to deal with Web Services and Semantic Web Services.

5. ACKNOWLEDGMENTS

This work has been supported by the EU IST Network of Excellence Knowledge Web. We would like to thank all the project partners from this and other projects (Esperonto and OntoGrid) for their comments, which have been so useful to improve the ODESeW technology and its usability.

6. REFERENCES

[1] AgentLink: http://www.agentlink.org/

[2] Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A. WebODE in a nutshell. AI Magazine 24(3):37-48. Fall 2003

[3] Cavaness C. Programming Jakarta Struts.O'Reilly. November 2002. ISBN 0-596-00328-5

[4] Contreras J, Benjamins VR, Blázquez M, Losada S, Salla R, Sevilla J, Navarro D, Casillas J, Mompó A, Patón D, Corcho O, Tena P. A Semantic Portal for the International Affairs Sector. EKAW'04. Springer-Verlag. Lecture Notes in Computer Science (LNCS) 3257:203-215. October 2004.

[5] Corcho O, Gómez-Pérez A, López-Cima A, López-García V, Suárez-Figueroa MC. ODESeW. Automatic Generation of Knowledge Portals for Intranets and Extranets. The Semantic Web - ISWC 2003. Springer-Verlag LNCS 2870: 802-817.

[6] Esperonto. http://www.esperonto.net/

[7] Gamma E, Helm R, Vlissides J, Jhonson R. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1995.

[8] Hamilton G. JavaBeans v1.01. 1997 http://java.sun.com/products/javabeans/

[9] Java Message Services. http://java.sun.com/products/jms/tutorial/

[10] Jin Y, Decker S, Wiederhold G. OntoWebber: Model-Driven Ontology-Based Web Site Management. 1st International Semantic Web Working Symposium (SWWS'01), Stanford University, Stanford, CA, July 29-Aug 1, 2001.

[11] JSF 1.1. Craig McClanahan, Ed Burns, Roger Kitain. JavaServer Faces. http://jcp.org/en/jsr/detail?id=127

[12] KAON. http://kaon.semanticweb.org

[13] Karvounarakis G, Christophides V, Plexousakis D, Alexaki S (2000) Querying community web portals. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. http://www.ics.forth.gr/proj/isst/RDF/RQL/rql.pdf

[14] Knowledge Web. http://knowledgeweb.semanticweb.org/

[15] Millstone. http://www.millstone.org/

[16] OntoGrid. http://www.ontogrid.net/

[17] Ontological Engineering Group. http://www.oeg-upm.net/

[18] Rhizomik. http://www.rhizomik.net/

[19] Roth M, Pelegrí-Llopart E. JavaServer Pages Specification. Version 2.0. 2003 http://java.sun.com/products/jsp/

[20] Singh I, Stearns B, Johnson M. Designing Enterprise Applications with the J2EETM Platform, Second Edition. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

[21] Spyns P, Oberle D, Volz R, Zheng J, Jarrar M, Sure Y, Studer R, Meersman R (2003). Ontoweb - a Semantic Web Community Portal. Fourth International Conference on Practical Aspects of Knowledge Management (PAKM), 2-3 December, 2002, Vienna, Austria, pp. 189-200

[22] Staab S, Angele J (2000) AI for the Web - Ontology-based Community Web Portals. 17th National Conference on Artificial Intelligence and 12th Innovative Applications of Artificial Intelligence Conference (AAAI 2000/IAAI 2000), Menlo Park/CA, Cambridge/MA, AAAI Press/MIT Press.

[23] Struts. http://struts.apache.org/index.html

[24] SWWS: http://swws.semanticweb.org/

[25] Turbine. http://jakarta.apache.org/turbine/

[26] Walsh N. RDF Twig: Accessing RDF Graphs in XSLT. Extreme Markup Languages, Canada, 2003

[27] Wicket. http://wicket.sourceforge.net