An Ontological SW Architecture Supporting Agile Development of Semantic Portals

Giacomo Bucci, Valeriano Sandrucci, and Enrico Vicario

Dipartimento di Sistemi ed Informatica
Università degli Studi di Firenze, Italy
{bucci,sandrucci,vicario}@dsi.unifi.it

Abstract. Ontological technologies comprise a rich framework of languages and components off the shelf, which devise a paradigm for the organization of SW architectures with high degree of interoperability, maintainability and adaptability. In particular, this fits the needs for the development of semantic web portals, where pages are organized as a generic graph, and navigation is driven by the inherent semantics of contents.

We report on a pattern-oriented executable SW architecture for the construction of portals enabling semantic access, querying, and contribution of conceptual models and concrete elements of information. By relying on the automated configuration of an Object Oriented domain layer, the architecture reduces the creation of a cooperative portal to the definition of an ontological domain model. Application of the proposed architecture is illustrated with reference to the specific case of a portal which aims at enabling cooperation among subjects from different localities and different domains of expertise in the development of a shared knowledge base in the domain of construction practices based on mudbrick.

Keywords: Pattern-Oriented SW Architectures, Semantic Web Technologies, Ontologies, Semantic Portals, SW Engineering.

1 Introduction

Technologies for ontological modelling and reasoning devise a new paradigm for the organization of software architectures with high degree of interoperability, maintainability and adaptability [1], [2].

In fact: ontologies natively allow representation of explicit semantic models combining non-ambiguity of technical specification with the understandability needed to fill the gap between technicians and stakeholders; ontologies enable the application of methods and tools off the shelf supporting reasoning for information search, model validation, and inference and deduction of new knowledge [3]; ontologies naturally fit into a distributed context, enabling creation of reusable models, composition and reconciliation of fragments developed in a concurrent and distributed manner [4], [5]; last, but not least, ontologies can model evolving domains, thus encouraging an incremental approach that can accompany the evolution towards shared models.

In particular, this potential appears well suited for the organization of web portals, where ontologies provide a paradigm to consistently design, manage, and maintain information architecture, site structure, and page layout [6]. This has crucial relevance

J. Filipe et al. (Eds.): ICSOFT/ENASE 2007, CCIS 22, pp. 185–200, 2008.

in the construction of portals with weblike organization [7], where pages are organized in the pattern of a generic graph, and navigation is driven by the inherent semantics of contents more than from a hierarchical upfront classification.

In [8], a semantic portal is presented which supports unified access to a variety of cultural heritage resources classified according to a public unifying ontology. The portal is developed using standard ontological technologies and SWI-prolog to support semantic search and presentation of retrieved data.

In [9], a portal based on ontology technologies is used as a basis to support the contribution of contents by a distributed community. The proposed architecture assumes that contribution is limited to individuals, and the investigation is focused on different techniques supporting the determination of a rank of relevance for the result-set of semantic queries.

In [10], a declarative approach to the construction of semantic portals is proposed, which relies on the definition of a suite of ontologies created by the portal programmer to define domain concepts and contents, navigation structure and presentation style.

In [11], the declarative approach of [10] is enlarged into a framework based on ontologies supporting the construction of a web application combining information and services. The framework implements the Model View Controller architectural pattern [12]. While the model is declared using ontologies, views are implemented with existing presentation technologies, and in particular JSP, which mainly rely on the OO paradigm. To fill the gap between the two paradigms [13], the developer is provided with a suite of predefined JSP components assuming the responsibility of the adaptation.

In this paper, we address the development of a portal enabling semantic access, querying, and contribution of both domain individuals and concepts. To this end, we propose an executable SW architecture based on standard languages and components off the shelf, which reduces the creation of a cooperative portal to the definition of an ontological domain model, and which is implemented with components that can be efficiently reused in a variety of data intensive and knowledge based applications. We report on the usage of the architecture in the case of a cooperative portal on mudbrick construction practices, that we call Muddy. In so doing, we also highlight the application programming model that permits the construction of a portal using the proposed architecture.

The rest of the paper is organized in 5 sections. After a brief overview of our perspective on the ontological paradigm in SW architecture, we introduce the Muddy case and we analyze its requirements, identifying abstract roles and use cases (Sect.2). We then expound the architectural design and some salient traits of its implementation, and we identify the roles involved in its application (Sect.3). Finally we describe the Muddy portal (Sect.4) and draw conclusions (Sect.5).

2 Requirements Analysis

2.1 The Ontological Paradigm

Ontological technologies mainly originate with the intent to contribute to the realization of the Semantic Web [14]. This denotes an evolution of the current web, in which information is sematically defined so as to enable automated processing.

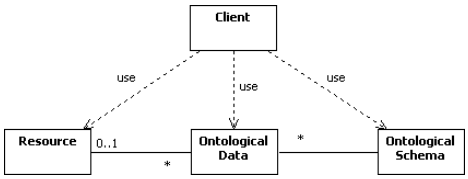

Fig. 1. Resources and meta-data relations

The Semantic Web paradigm thus emphasizes the relation between information and meta information, which distinguishes the concepts of Resource, Ontology Data, and Ontology Schema, as illustrated in Fig. 1. Resources are any information object on the web: a file, an HTML page, an image, a movie. In the semantic web perspective, each Resource will contain its Semantic Data. The Ontology Schema is a conceptualization shared among users, which captures the intensional part of the model, defining types of entities and their possible relations (concepts). Ontology Data (individuals), are the extensional part of the model, classifying Resources as realizations of the concepts in the Ontology Schema. Client is a Semantic Web reader and can be a human or an application. In both the cases, Client is interested to access Resources and their related semantic data.

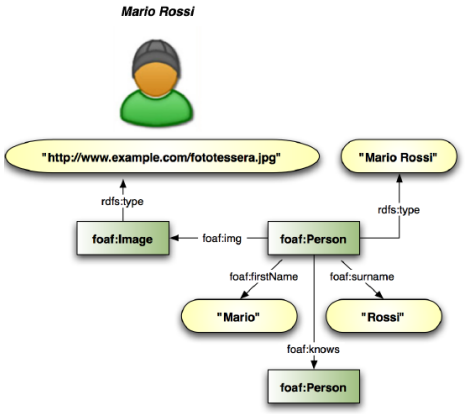

Fig. 2. Semantic annotation of a web resource

Fig. 2 exemplifies the concept: in this case the resource is a picture (fototessera.jpg) located at an http address (http://www.example.com). The public ontology named FOAF associates the resource with a set of concepts supporting interpretation by a SW agent: the picture (foaf:Image) is referred to (foaf:img) to a person (foaf:Person), with name (foaf:firstName) ”Mario” and surname (foaf:surname) ”Rossi”, who is in relation with (foaf:knows) other persons (foaf:Person).

Ontological technologies comprise a rich framework of paradigms, languages, and components off the shelf, which can serve beyond the specific intent of the Semantic Web, and may become an effective pattern for the organization of complex SW architectures.

2.2 A Cooperative Portal about Mudbrick Construction Practices

The Muddy project aims at supporting explicit and shared representation of knowledge about construction practices based on mudbrick. This is motivated by the widespread use of this building technology (estimates indicate that 30% of world population still live in mudbrick habitations), by the rich variety of different practices developed in different localities, and by the high energetic efficiency and low environmental impact which make it a sustainable practice.

In particular, the Muddy project aims at developing a web portal based on ontological models, enabling cooperation among subjects from different localities and different domains of expertise, in the development of a shared model and in the contribution of concrete contents for it.

2.3 Abstract Roles

Analysis of functional requirements of the Muddy project in the light of the organization of information according to the ontological paradigm of Fig. 1, identifies roles, users' needs, and use cases generalized beyond the limits of the specific context of use [15]. These roles are outlined in Fig. 3

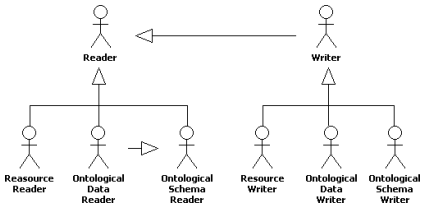

Fig. 3. Abstract roles

Resource Readers correspond to readers in the conventional web. They are interested in accessing information, using meta-information to maintain context and to access resources, through direct links or search engines. In the case of Fig. 2 they could for instance be interested in the web resource http://www.example.org/fototessera.jpg. In a similar manner, Resource Writers are also enabled to insert, update and delete resources.

Ontological Schema Readers take the role that [16] qualifies as second-level reader: they are interested in understanding the organization of concepts more then their concrete realizations. They need to navigate in ordered manner and search classes and properties of an ontological model. In the example of Fig. 2 they would be interested in the structure of the FOAF ontology more than in the concrete resource fototessera.jpg.

Ontological Schema Writers also modify models, taking part to the definition of the strategy of content organization. In particular, they may be interested in fixing errors, changing the model to follow some evolution, or extending the model by specialization and inheritance.

An Ontological Data Writer is a human or a SW indexing Resources with respect to the concepts of an Ontological Schema. Besides, an Ontological Data Reader is a human or, more frequently, a SW which exploits Ontological Data to access concrete resources in semantic querying [17]. Ontological data can also be formatted to be easily readable as well as a resource by human users [18].

In the example of Fig. 2 they will thus be interested in the instances of the FOAF ontology that refer to the resource "http://www.example.org/fototessera.jpg".

2.4 Use Cases in the Muddy Portal

In the specific context of the Muddy portal, the main goals of a Reader are browsing of pages derived from resources and semantic models, navigation of links due to relations in the ontological model, execution of semantic queries on the knowledge base (see Fig. 4).

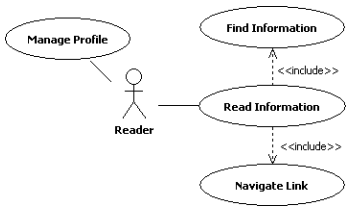

Fig. 4. Web portal Reader use cases

Besides, the Writer (see Fig. 5), extends the Reader capability to access information with the capability to contribute new knowledge in the form of a generic Ontological Schema or Data. Namely, the enabled user can contribute either the extensional or the intensional part of the ontological model for the web portal. Writers can also send/receive feedback and comments about the ontological model so as to encourage collaboration and cooperation among users. To contribute, writers will upload model files to ease the development of a portal prototype.

A further instrumental role is the Administrator, whose main responsibility is the assignment of readers and writers privileges.

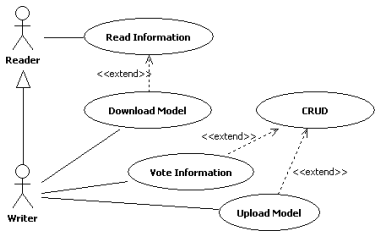

Fig. 5. Web portal Writer use cases

3 Architecture and Development Process

3.1 Architectural Components

The conceptual architecture of our semantic portal composes a variety of partecipants that can be effectively assembled using W3C supported specifications and components.

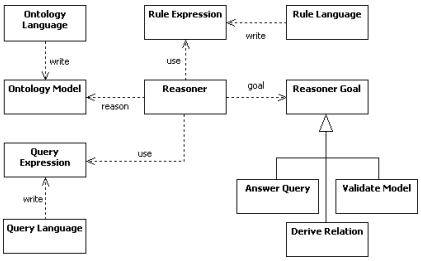

Fig. 6. Architectural components of the web portal

Ontology Model. The Ontology Model (see Fig. 6) is the main component of the architecture, with the main responsibility of providing representation of the domain model of the application. It is implemented by composition of the Ontology Schema and Ontology Data (see Fig. 1), both encoded using the W3C standard Ontology Web Language (OWL) [19]. In principle, the entire expressivity of OWL-full can be exploited, but successful termination of reasoning tasks included in portal services is guaranteed only under the assumption that the model is encompassed within OWL-DL reasoning.

In a logic perspective (see Fig. 7), the ontology model can be decomposed in three main parts: classes, properties and individuals. Classes in the Ontology Model are part of the Ontological Schema and play a role that can be compared to that of table definitions in a relational database. However, as opposed to relational tables, classes can be composed through delegation and inheritance. Properties are also part of the Ontological Schema, and they are used to define relations among classes. Individuals are realizations of concepts described by classes, and play a role that can be compared to that of records in a relational database.

It is worth noting that, in this architecture, the domain logic is modelled and specified using ontologies rather than UML class diagrams of the common practice of OO development. In a practical perspective, an UML model can be easily mapped to an ontology, provided that some major differences among class diagrams and ontologies are taken into account [20]: in ontological models, properties exist independently from classes to which they apply, and they can thus be organized in a hierarchy; a relation between two UML classes is encoded as a property, whose domain and range are the two classes themselves; in an ontological model, an individual can belong to multiple classes which can change during its lifecycle, as the type is a property itself of the individual; in ontological models, classes can be defined using set-theoretical constructs.

Fig. 8 reports the example of an ontological model: Book ed Author are classes associated through an oriented relationship capturing the property hasAuthor. Book, Author

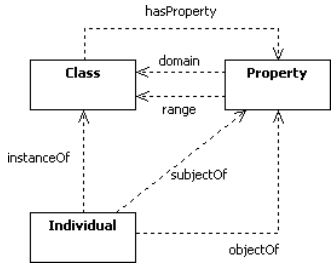

Fig. 7. Logical compontents of the ontology model

and hasAuthor comprise the intensional part of the ontological model. "Il Barone Rampante" is an instance of the class Book, while Italo Calvino is instance of the class Author. The two objects are related through an instance of the association hasAuthor. "Il Barone Rampante", Italo Calvino and their association comprise the extensional part of the ontological model.

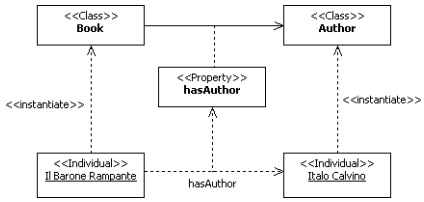

Fig. 8. An ontological model

Rules. To form the overall knowledge base, the ontological model is complemented with rules that extend the information explicitly represented with inference rules that let the system derive new knowledge. In general, a rule has the form of an implication $antecedent \Rightarrow consequent$ where both $antecedent$ and $consequent$ are conjunctions of atoms. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., $?x$). For instance, the rule asserting that an uncle is the brother of a parent would be written as: $Person(?x)\ and\ Person(?y)\ and\ Person(?z)$ $and\ parent(?x, ?y)\ and\ brother(?y, ?z) \Rightarrow uncle(?x, ?z)$. Note that $Person$ is a class in the ontological model while $parent$, $brother$ and $uncle$ are properties.

In our architecture, rules are represented using the W3C supported Rule Language SWRL. To circumvent limitations affecting open source reasoners, we internally represent the language using Jena rules [21].

Querying and Reasoning. Query on the knowlegde base are expressed using the W3C supported the SparQL. While the architecture supports the full expressivity of SPARQL, for the sake of usability, and in particular learnability, only a restricted fragment of the language is provided to the user. For instance, the query $SELECT?titleWHERE$

$\{< http://example.org/book/book1 >< http://purl.org/dc/elements/1.1/title >?title.\}$

retrieves the title (or, more precisely the value of property http://purl.org/dc/elements/1.1/title) of the individual identified by the URI http://example.org/book/book1. Also in SPARQL, variables are represented by prefixing them with a question mark.

SPARQL enables the expression of complex queries in simple and compact form. For instance, the following query retrieves the titles of books containing the term Rampante in their title:

$$PREFIX\ dc :< http://purl.org/dc/elements/1.1/ >$$
$$SELECT ?titleWHERE\{?xdc : title?titleFILTER\ regex(?title,"Rampante","i")\}$$

The API Jena is used to drive reasoning and retrieve information by decidingSPARQL queries on the model. The API is also used to validate the ontology model and derive new knowledge using OWL axioms and inference rules. In general, any reasoner represents a trade-off between power and efficiency in computing. In particular, in the case of our architecture termination of reasoning tasks is guaranteed only if the model and the rules are encompassed within the boundaries of OWL-DL and SWRL, respectively.

3.2 Participants in the Development Process

The proposed SW architecture supports separation of concerns among four different roles of Domain Expert, Ontology Expert, Stakeholder, IT Expert. These naturally fit in a realistic social context of development [22] and basically correspond to the roles identified in [4].

The Domain Expert knows about the domain of the portal and share partially formalized models among the community who belongs to. Domain Experts usually use specific tools to do their analysis and produce their research result. It is often the case that they don't know anything about ontologies and also they don't have opportunity (no time available) to learn about them.

The Ontology Expert is able to use sematic modelling tools and can describe knowledge contributed by Domain Experts with an Ontology Model. In this way the information, that was heterogeneous and sometimes also tacit or embedded, becomes formalized, explicit, homogeneous and consistent [23].

The Stakeholder is interested in the domain logic but he/she is not necessarily expert. For this role, it is useful to have an ontology model that can be read and studied and that can be used to navigate through Domain Experts documents.

Finally, the IT Expert has to develop software tools needed by other roles so to let them read and write resources and ontology models, see Fig. 1 and 3.

3.3 Salient Aspects of the Implementation

Layering. The source code of the web portal (see Fig. 9) is organized in three layers [12], [24]. As usual, in SW architecture, layering separates presentation, domain logic and persistence. In this case, layering also helps in filling the gap between ontological and object oriented perspectives [13] [25], which somehow reproduces the well known problem of Impedance Mismatch between Objects and relational databases [26].

Specifically: the presentation layer contains the logic to handle the interaction between the user and the software, relying on Java Server Faces (JSF) [27]; the domain

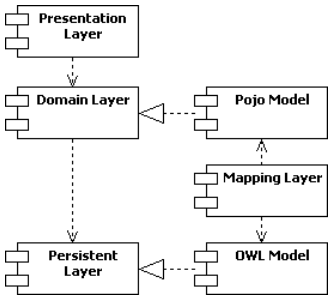

Fig. 9. Web portal layering

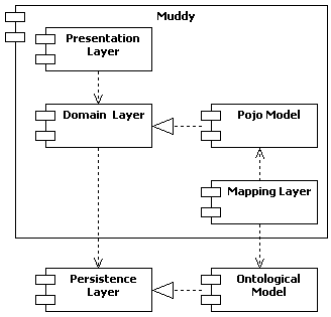

Fig. 10. Muddy Web portal layering

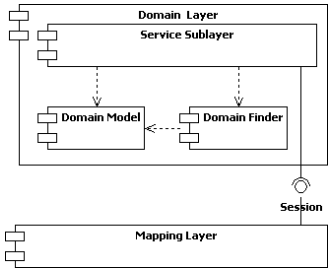

Fig. 11. Relation between domain logic and mapping layer

layer contains the application logic i.e. the domain model, implemented with Plain Old Java Objects (POJO); the persistent layer, is implemented as an ontology used to persist objects of the domain model.

Developers can refer to patterns such as Active Record or Mapper [24] and also use tools [28], [29] to let objects of the domain model correspond to individuals of the ontology model. For the web portal, a Mapping layer was used between domain and persistent layers because it allows better decoupling, easer testing and concurrent developing [30], [31].

The domain layer includes various components: the Domain Model (discussed later); the Domain Finder which includes classes with responsibility in querying the knowledge base to retrievethe objects of the Domain Model; the Service Sublayer which comprises a facade on Domain Model and DomainFinder for the purposes of higher layers of the application. It is interesting to note that the Domain Layer is able to access all the functions of the underlying Mapping Layer through the sole interface Session.

The Mapping layer manages the mapping between models and meta-models, elaborates complex relations i.e. reification, hides SPARQL embedded code and improves performances with methods like caching and proxying. Last but not least, only the mapping layer refers to the low level API i.e. Jena [21] so that is easer to change the used library [32] and that could be useful for a rapidly changing domain such as ontologies. To better understand its structure, notice that the Mapping Layer is comprised by three

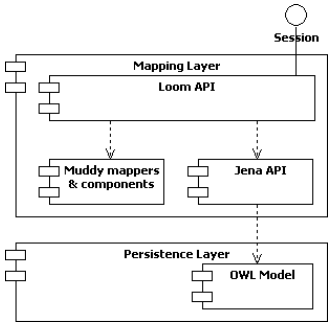

Fig. 12. Relation between mapping layer and persistence layer

major components: the Loom library, implemented as a part of our work, provides all basic functions that support agile implementation of a mapping layer; mappers and finders are specific to the application domain and they are implemented as sub-classes of abstract base classes included in the Loom Api; other additional libraries support low level access to ontological OWL models OWL (Jena API in the picture).

Domain Model. The POJO Model in the domain layer is composed by two Java packages which are derived from three ontological models.

The User package contains data about profiles, users and related information, and it is automatically derived from an ontology User model, so as to map ontology classes and properties to OO classes and attributes.

The Domain package has responsibility to manage information contributed by users and is derived from an ontological Domain model according to the architectural pattern of reflection [12]: all types in the ontological model are mapped into a single generic OO class (that would be a meta-level-class in the reflection pattern); this generic class has responsibility to manage relations among types defined by the users in the ontological model (that would comprise base-level-classes in the reflection pattern). This decouples the structure of the OO domain layer from the specific types defined in the Domain Ontology, thus enabling reuse of the OO layer for a variety of different ontological Domain models, defining different evolutions of a portal or different portals insisting

on different application domains. Also, this is the feature that permits the cooperative portal to accommodate contributions not only in the individuals of the extensional part, but also in the concepts of the intensional part of the knowledge-base.

Derivation of the Domain package is also affected by an additional ancillary ontology defining directives for the presentation of data in the page layout. The individuals of this ontology are used by the mapping layer to determine the presentation and filtering of concepts defined in the ontological Domain model. This accomplishes a responsibility which is much similar to that of "site view graphs" in the OntoWebber framework [10].

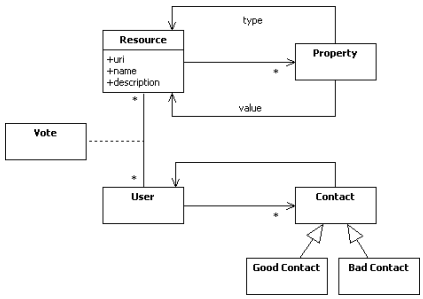

Fig. 13. The POJO Model of the web portal

Mapping Layer. Mapping between ontological and object-oriented models of the architecture was implemented following a pattern-oriented design [24], [12] aimed at building an extensible and reusable framework.

Mappers. The mapping layer includes a mapper class for each element of the OO domain model [24] (see Fig. 14). Each mapper class can read information form the ontological model to assign it to the object-oriented model and vice versa, and it is implemented as extension of the abstract DataMapper base class.

Mappers decouple the object-oriented Domain Layer from the ontological Persistence Layer, so that a client can ignore how objects are persisted in the ontology. For

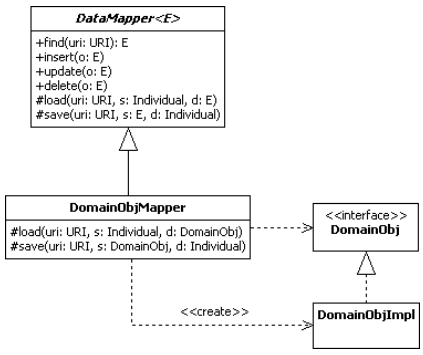

Fig. 14. Mappers

the sake of performance and and behavioral abstraction, DataMappers implement some specific patterns [24], [33]:

- **Identify Map (Cache):** mappers have a cache memory to speed up repeated access to the same object.
- **Proxy:** if possible, mappers substitute objects requested by the client with equivalent proxies. This delays the mapping operations until they are really needed.
- **LazyLoad:** mappers load objects of a list when they are used so the slow mapping operation is executed for useful objects only.
- **UnitOfWork:** unit of work class manages changes to objects that mappers have to persist.

Developers can use an instance of the class Session to access functions of the mapping framework (see Fig. 15).

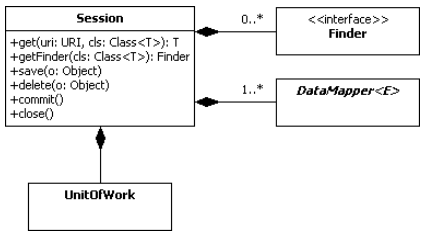

Fig. 15. The Session class

4 The Muddy Portal

Muddy is a web-based application implemented as an instance of requirements and architecture described so far. It allows reading and writing of concrete and conceptual information according to an ontological paradigm, providing the following user functions: navigate information following semantic links; execute semantic queries on the knowledge base; contribute new knowledge uploading model files; read/write feedback about the knowledge base.

As characterizing traits: users know the kind of technology employed to model data, as opposed to systems where a service is offered to users who ignore the underlying technology; users can share arbitrary ontological models, as opposed to applications where users interact with predefined conceptual models by creating, retrieving, updating and deleting individuals only.

4.1 The Portal Architecture

Fig. 16 depicts the architecture of the portal managed by the application.

Index is the first page of the portal with login and registration, giving access to the Home page and then, through Header page to the functions of the portal.

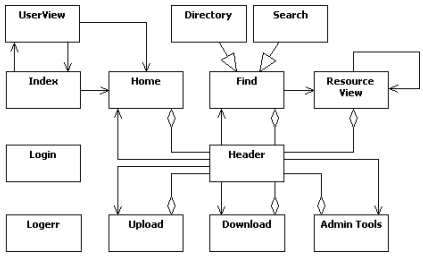

Fig. 16. Structure of the Muddy Portal

Users can be readers, writers and administrators and they are provided with different functions. A new user is always classified as reader and only administators can give users more privileges.

Find page is used to execute queryies on the knowledge base by users and it is specialized in Search and Directory pages. ResourceView page allows users to read information contained in the knowledge base. Upload and Download pages allow users to contribute new knowledge. Admintools page is for the administrator.

4.2 Find Pages

The Directory page (see Fig. 17) is used to execute pro-active search. The system shows to the user a list of categories that correspond to root classes of the ontological model managed by the portal. The user can select a category to get a page containing the list of instances of the category and the list of its direct subclasses. The user can navigate toward more specific classes or can inspect one of the instances found.

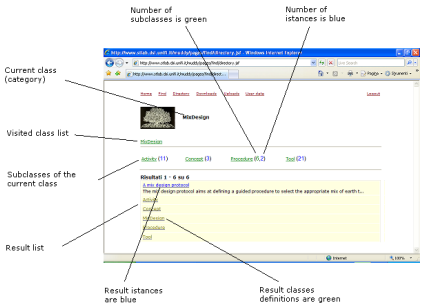

Fig. 17. The Directory search page

The Search page, see Fig. 18, implements an extension of full-text search methods. Users can specify one or more words that must be contained in desired resources. They can also specify the kind of relations that link desired resources to specified words. For instance, the expression ”neededTools = sieve” lets a user require all resources that has a ”sieve” among needed ”tools”.

This page tries to simplify the use of SPARQL to users.

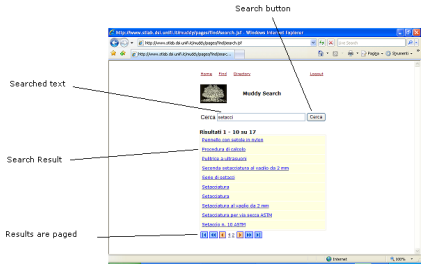

Fig. 18. The Search page

4.3 Resource View Page

This page shows information about a resource (see Fig. 19), and allows users to speed up navigation towards semantic related resources.

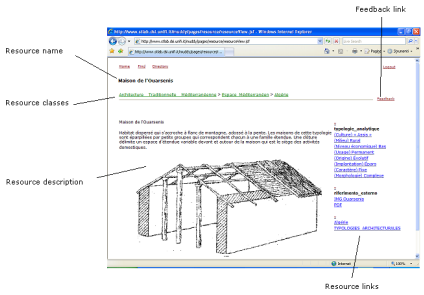

Fig. 19. The ResourceView page

The portal also allows users to give feedback about accessed resources which is used to calculate appreciation indexes about resources.

5 Conclusions and Ongoing Activity

We are further developing the portal and its underlying architecture, facing various interrelated issues, with less scientific relevance but crucial to tackle the transition phase towards the context of use:

- a usability cycle has been planned, to evaluate the capability of the portal to support the user in maintaining context in the navigation through the weblike structure of portal contents; in this perspective, the orientation towards change in the overall architecture, and in particular the concept of presentation ontology implemented in the mapper, provide major assets to face iterative refinement in design choices;
- preliminary performance profiling indicates that performance can be largely improved by the integration of a more elaborated RDF repository;

– functional extensions are being developed to implement the automated derivation of an editor of individuals based on the structure of ontology concepts and a tool for annotation of existing web resources with reference to the concepts of an ontological model.

References

1. Fayad, M., Cline, M.P.: Aspects of software adaptability. Communications of the ACM (1996)
2. ISO: Iso 9126 Software engineering – product quality (2004), http://www.iso.org/iso/en/ISOOnline.frontpage
3. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., Stojanovic, N., Studer, R., Stumme, G., Sure, Y., Tane, J., Volz, R., Zacharias, V.: Kaon tool suite (2007), http://kaon.semanticweb.org/frontpage
4. Tempich, C., Pinto, H.S., Sure, Y., Staab, S.: An argumentation ontology for distributed, loosely-controlled and evolving engineering processes of ontologies (diligent). In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 241–256. Springer, Heidelberg (2005)
5. Tempich, C., Pinto, H.S., Staab, S.: Ontology engineering revisited: an iterative case study with diligent. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 110–124. Springer, Heidelberg (2006)
6. Garzotto, F., Mainetti, L., Paolini, P.: Hypermedia design analysis and evaluation issues. incomm. of the ACM. Communications of the ACM (1995)
7. Lynch, P.J., Horton, S.: Web style guide: Basic design principles for creating web sites. Yale University Press (2002)
8. Schreiber, G., Amin, A., van Assem, M., de Boer, V., Hardman, L., Hildebrand, M., Hollink, L., Huang, Z., van Kersen, J., de Niet, M., Omelayenko, B., van Ossenbruggen, J., Siebes, R., Taekema, J., Wielemaker, J., Wielinga, B.J.: Multimedian e-culture demonstrator. In: International Semantic Web Conference, pp. 951–958 (2006)
9. Stojanovic, N., Maedche, A., Staab, S., Studer, R., Sure, Y.: Seal: a framework for developing semantic portals (2001)
10. Jin, Y., Decker, S., Wiederhold, G.: Ontowebber: Model-driven ontology-based web site management. In: 1st International Semantic Web Working Symposium. Stanford University, Stanford (2001)
11. Corcho, O., Lòpez-Cima, A., Gomez-Pérez, A.: A platform for the development of semantic web portals. In: ICWE 2006: Proceedings of the 6th international conference on Web engineering, pp. 145–152. ACM Press, New York (2006)
12. Schmidt, D.C., Rohnert, H., Stal, M., Schultz, D.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley & Sons, Inc., New York (2000)
13. Woodfield, S.N.: The impedance mismatch between conceptual models and implementation environments. In: Embley, D.W. (ed.) ER 1997. LNCS, vol. 1331. Springer, Heidelberg (1997)
14. Berners-Lee, T.: Semantic web roadmap (1998), http://www.w3.org/2001/sw/
15. ISO: Iso 9241-11 guidance on usability (1998), http://www.iso.org/iso/en/ISOOnline.frontpage
16. Eco, U.: Six walks in the fictional woods. Harvard University Press (1994)
17. Bonino, D., Corno, F., Farinetti, L.: Dose: a distributed open semantic elaboration platform. In: The 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003), Sacramento, California, November 3-5 (2003)

18. Dzbor, M., Motta, E., Domingue, J.B.: Opening up magpie via semantic services. In: Proc. of the 3rd Intl. Semantic Web Conference, Japan (November 2004)
19. W3C: Owl web ontology language (2004), http://www.w3.org/TR/owl-features/
20. Brockmans, S., Volz, R., Eberhart, A., Löffler, P.: Visual modeling of owl dl ontologies using uml. In: International Semantic Web Conference, pp. 198–213 (2004)
21. Company, H.P.D.: Jena a semantic web framework for java (2002), http://jena.sourceforge.net/
22. Cockburn, A.: The interaction of social issues and software architecture. Commun. ACM 39, 40–46 (1996)
23. Kryssanov, V.V., Abramov, V.A., Fukuda, Y., Konishi, K.: The meaning of manufacturing know -how. In: PROLAMAT 1998: Proceedings of the Tenth International IFIP WG5.2/WG5.3 Conference on Globalization of Manufacturing in the Digital Communications Era of the 21st Century, Deventer, The Netherlands, pp. 375–388. Kluwer, Dordrecht (1998)
24. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
25. Guizzardi, G., Falbo, R., Filho, J.: Using objects and patterns to implement domain ontologies. In: 15th Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil (2001)
26. Ambler, S.: Agile Database Techniques: Effective Strategies for the Agile Software Developer. John Wiley & Sons, Inc., New York (2003)
27. Mann, K.D.: JavaServer Faces in Action (In Action series). Manning Publications Co., Greenwich (2004)
28. Protégé.: Stanford-University (2006), http://protege.stanford.edu
29. Kalyanpur, A., Pastor, D.J., Battle, S., Padget, J.A.: Automatic mapping of owl ontologies into java. In: SEKE, pp. 98–103 (2004)
30. Heumann, J.: Generating test cases from use cases. The Rational Edge (2001)
31. Beck, K.: Test Driven Development: By Example. Addison-Wesley Professional, Reading (2002)
32. Aduna: Sesame: Rdf schema querying and storage (2007), http://www.openrdf.org/
33. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley Professional, Reading (1995)