International Journal of Parallel Programming, Vol. 34, No. 6, December 2006 (© 2006)
DOI: 10.1007/s10766-006-0022-1

An Experimental Ontology Server for an Information Grid Environment

A. Aiello,[1,2] M. Mango Furnari,[1] A. Massarotti,[1] S. Brandi,[1] V. Caputo,[1] and V. Barone[1]

Received May 27, 2005; accepted September 11, 2006

Semantic web and grid technologies offer a promising approach to facilitate semantic information retrieval based on heterogeneous document repositories. In this paper the authors describe the design and implementation of an Ontology Server (OS) component to be used in a distributed contents management grid system. Such a system could be used to build collection document repositories, mutually interoperable at the semantic level. From the contents point of view, the distributed system is built as a collection of multimedia documents repository nodes glued together by an OS. A set of methodologies and tools to organize the knowledge space around the notion of contents community is developed, where each content provider will publish a set of ontologies to collect metadata information organized and published through a knowledge community, built on top of the OS. These methodologies were deployed while setting up a prototype to connect about 20 museums in the city of Naples (Italy).

KEY WORDS: Semantic web; ontology; metadata; information grid.

1. INTRODUCTION

A dynamic computational environment is characterized by entity autonomy, heterogeneity, and distribution. It is an environment in which a priori agreements regarding engagement cannot be assumed. Hence, trading partnerships must be dynamically selected, negotiated, procured, and monitored. To achieve the flexible assembly of components and

[1]Istituto di Cibernetica E. Caianiello, Via Campi Flegrei, 34, I-80078, Pozzuoli, Italy.
[2]To whom correspondence should be addressed. E-mail: a.aiello@cib.na.cnr.it

resources requires not only a service-oriented model, but also information about functionality, availability, and interfaces of the various components. This information must have an agreed-upon interpretation that can be processed by machines. Thus, explicit assertion of knowledge and the explicit use of reasoning services should be required.

Knowledge is crucial for the flexible and dynamic middleware embodied by the *Open Grid Service Architecture* (OSGA).(12) For example, the dynamic discovery, formation, and dispatching of ad-hoc virtual organizations of resources require that Grid middleware be able to use and process knowledge about availability of services, their purposes, how they are discovered, invoked and evolve.

Knowledge and semantics may be implicitly or explicitly asserted and used. Computationally implicit knowledge is knowledge merely embedded in programs or tools in forms such as database schema or algorithms. Computationally explicit knowledge is knowledge for which some sort of formal knowledge representation technique exists that can be exposed to be discovered, processed and interpreted. Machines need formal, standardized *declarative* representations and reasoning schemes over those representations. Thus knowledge may be considered pervasive and ubiquitous, permeating the Grid.

Semantic web and knowledge-oriented Grid depend on making knowledge *explicit* so that rich semantics can be used in decision-making and in purposeful activity by computational entities provided with a machine-processable account of the meaning of other entities with which they interact. Informally specified knowledge and metadata are suitable only for human consumption, because humans can hope to make sense of knowledge in a wide variety of forms and contexts.

Metadata is one way to represent knowledge in a knowledge-oriented Grid system; it is intended to be machine processable and takes the form of declarative statements to be used when annotating content. Another representation possibility is given by the ontology approach, where an ontology is a conceptualization of a domain that provides a shared language for a community of knowledge (services) providers and consumers, be they machines or people.(15) Ontologies can serve as the conceptual backbone for every task in the knowledge management life cycle. As a formal specification, ontology is open to computational reasoning. Thus, metadata description using terms from an ontology can also be reasoned over to infer knowledge implied by, but not explicitly asserted in the knowledge base. They provide tools to structure and retrieve knowledge in a comprehensive way and are essential for knowledge search, exchange, and discovery over the Internet.

To put metadata and ontologies at work, methods, and tools to support their deployment are necessary. The core technologies proposed for semantic web and knowledge-oriented Grids have their roots in distributed systems and knowledge management systems, where ontologies and associated ontology reasoners are necessary to allow computational processes to fully interact. An adequate architectural view for semantic web and Grid applications is a component-based one, in which the various macro-components work together. However, the current ontologies exploitation attempts are oriented to cope with the conceptualization of single knowledge source. Furthermore, most of the existing tools consider ontologies as monolithic entities and provide little support for specifying, storing, and accessing ontologies in a modular manner.

In this paper the authors describe the design and implementation of an Ontology Server component to be used in a distributed contents management grid system. Such a system could be used to build collections of document repositories, mutually interoperable at the semantic level. The authors' efforts are based on the hypothesis that it is necessary to develop an adequate treatment for distributed ontologies, in order to promote knowledge sharing on the semantic web. As such, appropriate infrastructures to represent and manage distributed ontologies also need to be developed. To pursue these goals we developed an ontology server to deploy a knowledge repository community. An experimental test bed to verify the developed methodologies and tools was built within the cultural heritage promotion arena.

The rest of the paper is organized as follows: in Section 2 the Semantic Web and Information Grid scenario are described; Section 3 presents the ontology life cycle in detail. In Section 4 the adopted Ontology Server Architecture is described, while in Section 5 the implemented test bed and the experience gained with this deployment are summarized.

2. THE SEMANTIC WEB AND INFORMATION GRID SCENARIO

The intent of ontology middleware is to develop new capabilities to be constructed in a dynamically and transparent way from distributed services, reusing existing components, and knowledge resources. The aim is to assemble and coordinate these components in a flexible manner.

Although different knowledge management tasks are coupled, their interactions are not hardwired. Each component will deal with different tasks and can use different techniques and tools. Each of them could be updated while others are kept intact. Thus, the architecture should be robust in the sense that new techniques and tools can be adopted at any time. Knowledge could be added into the knowledge warehouse at any time. It should be necessary only to register the knowledge with the knowledge authority. After registration all of the services, such as publishing and inferencing, should be used to expose the new knowledge for use.

The various Semantic Web or Grid application components can be organized in software layers and placed into service-oriented relationships with one another. This service-oriented architectural view can be summarized as follow:

Basic services cover data and computational services such as networked access, resource allocation and scheduling; they use metadata associated with services and entities, but the semantic meaning of that metadata is implicit or missing;

Semantic knowledge services introduce explicit meaning and semantic description about a service that explicitly and declaratively assert their purposes and goals, not just the syntax of the data type or the signature of the function calls, so that computational entities can make decisions in light of that knowledge.

Knowledge services are the core services needed to manage knowledge in the semantic web or Grid, such as knowledge publication, annotation services, and inference engine services.

The minimal set of needed components consists of: annotation mechanisms, repositories for annotations and ontologies, query, and lifecycle management, as well as inference engines that are reliable and perform well. Knowledge publishing should allow users to register new distributed knowledge and service knowledge, which should be accessed and retrieved in the same way we browse the web.

In addition, tools are needed for acquiring and managing knowledge, such as an entrance point to an integrated knowledge management system, or the *knowledge authority* to provide a secure infrastructure for authentication and authorization, so that knowledge can be used and updated in a controlled way. These functionalities could be organized, for example, in the following set of services:

- *Annotation services* associate entities and their metadata together, in order to attach semantic content to those entities.
- *Ontology services* provide access to concepts in an underlying ontology data model. Services include extending the ontology, querying it and returning the parents or children of a concept, as well as determining how concepts and roles can be combined to create new legal composite concepts.
- *Inference engine* applies different kinds of reasoning over the same ontologies and the same metadata set.

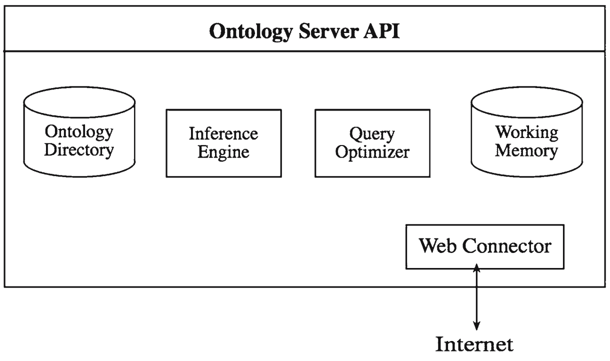


Fig. 1. "Thin client" ontology server configuration.

The deployment configurations and the corresponding functionalities for such an environment could range from the most lightweight deployment to very complex ones. Below, a few of these possible configurations are briefly described.

Thin-Client: In this configuration (see Fig. 1), the ontological knowledge is made available via Web based protocols. Such a system would be appropriate for use with applications that require to reference standard ontology knowledge .

Fat-Client: A "fat-client" configuration provides local persistence for ontological knowledge that the application generates (see Fig. 2). Depending on the configuration details, knowledge may be stored locally as files or in a database for increased performance and reliability. The local persistent storage can also provide local caching for ontological knowledge accessed via Web.

Client-Server: This configuration models a full-fledged ontology management system with a complex set of functionalities. To provide ontology sharing and evolution among a large number of clients, the ontology management functions would be moved to an ontology server (see Fig. 3). Not only does this arrangement provide for easier ontologies sharing and updating in a distributed environment, but it also allows for additional optimization and optimized indexing to support better knowledge query and retrieval. The ontology knowledge can also be preprocessed to speed up the handling of the most common queries. In addition, this configuration can be further extended to accommodate different sources of ontological knowledge.

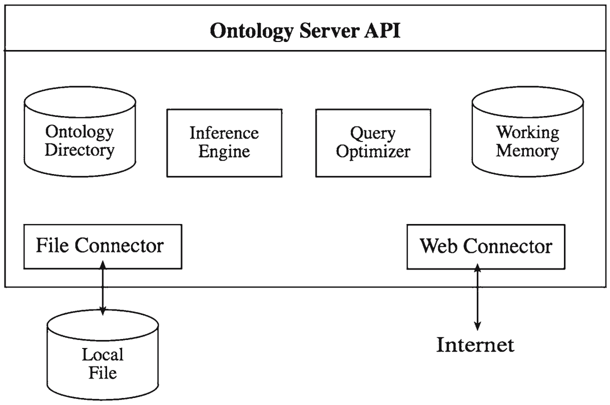


Fig. 2. “Fat client” ontology server configuration.

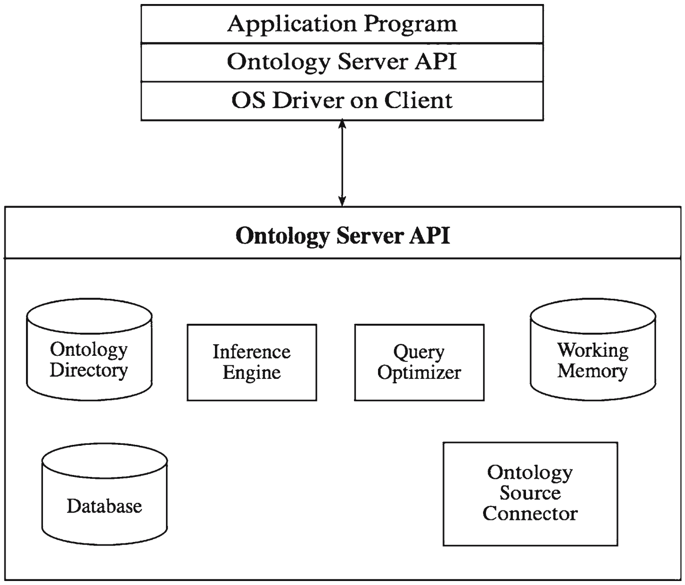


Fig. 3. Client/server ontology server configuration.

We had in mind this last scenario when designing and implementing the Ontology Server described in the rest of the paper and which we used in implementing a distributed content management system for the cultural heritage area, as described in Section 5.

3. THE ONTOLOGY LIFE CYCLE

Ontologies vary greatly in size, scope and semantics. They can range from generic upper-level ontologies (SUMO,(27) Cyc(21)) to domain-specific schema (NCI Cancer).(14) They can be created by knowledge representation experts or novice web users. They can be small ontologies containing a handful of concepts (FOAF(5)) or large ontologies containing thousands of terms and relationships (Galen(31), GO(13)). In such a diverse and heterogeneous knowledge space, ontology engineering assumes tremendous practical significance. Tools supporting it need to provide a seamless environment for browsing, searching, sharing, and authoring ontological data. To provide management support for the entire lifecycle of ontological knowledge, an ontology management system need to address a wide range of problems. Ontology lifecycle spans from creation to evolution.

The main ontology language requirements for the representation of a formal conceptualization of a domain are: a well defined syntax, formal semantics, an efficient support, and sufficient expressiveness power. In this context formal semantics describe the meaning of knowledge precisely. Here, precisely means that the semantics neither do refer to subjective intuitions, nor are they open to different interpretations by different people or machines.

The Resource Description Framework (RDF)(16) provides a foundation for representing and processing metadata. It has a graph-based data model and its key concepts are resources, properties, and statements, where a statement is a resource-property-value triple. The RDF allows incremental building of knowledge, its sharing, and reuse and furthermore, it is domain independent. The RDF Schema is a primitive ontology language. It provides a mechanism to describe specific domains and offers certain modelling primitives with fixed meaning. The key concepts of RDF Schema are classes, subclass relations, property, subproperty relations, as well as domain and range restrictions.

Ontology Web Language (OWL)(26) is the standard for ontologies proposed by W3C [1]. The OWL builds on top of RDF and RDF Schema, uses RDF's XML-based syntax and allows to describe the semantics of knowledge in a machine accessible way. Furthermore, formal semantics and reasoning support is provided through the mapping of OWL on logic, such as Predicate Logic and Description Logic.

Tool support should be available for all stages of the ontology lifecycle, whose main steps are:

[1]W3C is the acronym for *World Wide Web Consortium*, http://www.w3c.org

Creating: An ontology may be built from scratch, using a tool for editing and creating class structures, usually with an interface similar to a file system directory or bookmark folder interface. In the process of creating a new ontology it is also possible to reuse, in whole or in part, ontologies that have been previously developed.

Populating: Refers to the process of creating instances of the concepts in an ontology and linking them to external sources, such as web pages that are good sources of instance knowledge, or legacy sources of instances like product catalogues, white pages, database tables, etc.

Deploying: There are many ways to deploy an ontology once it has been created and populated. For example, the ontology provides a natural index of the instances described in it, and hence can be used as navigational aid while browsing those instances. Since OWL has capabilities for expressing axioms and constraints on the concepts in the ontology, powerful logical reasoning engines can be used to draw conclusions about instances in an ontology.

Validating, Evolving and Maintaining Ontologies: Like any other component of a complex system, ontologies will need to change as their conceptualization changes. Some changes might be simple responses to errors or omissions in the original ontology; others might be in response to a change in the environment. There are many ways in which an ontology can be validated in order to improve and evolve. The most effective ones are based on strict formal semantics of what the class structure means.

The ontology maintenance task may require merging ontologies from diverse sources. When this is the case tool support is important, for example to provide human-centered capabilities for searching through ontologies for similar concepts (usually by name), provisioning for merging the concepts, or performing more elaborate matching, based on common instances or patterns of related concepts.

A number of ontology development tools currently exist; notable among these are Protégé,(29) OilEd,(2) OntoEdit,(33) OntoLingua,(11) and WebODE.(1) Most of these tools provide an integrated environment to build ontologies, to check for errors and inconsistencies (using a reasoner), to browse multiple ontologies, and to share and reuse existing data by establishing mappings among different ontological entities.

4. THE ONTOLOGY SERVER ARCHITECTURE

Within the knowledge Grid scenario, the Ontology Server (OS) provides the basic semantic interoperability functionalities. In fact, it

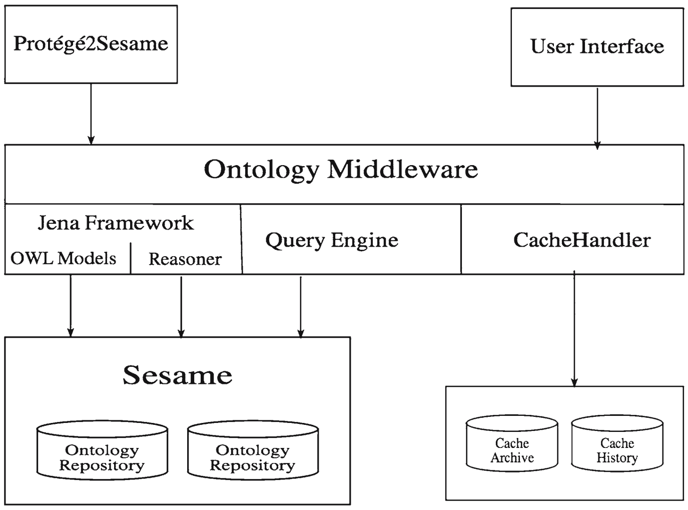


Fig. 4. Ontology server architecture.

provides the knowledge producer with the possibility of interacting with heterogeneous and distributed document repositories. It guarantees to the knowledge providers the necessary autonomy to organize the managed contents space.

From the conceptual point of view, the OS is one of the most important kinds of server since it manages the schema for the stored knowledge expressed using OWL/RDF[(24)] and determines the interactions with the other semantic web or Grid servers and components.

The main components of the OS architecture are organized into the Ontology Middleware, which coordinates the activities for the ontology development, the reasoner system and the user interface component, as shown in Fig. 4.

To achieve these goals, the OS was designed around the following layers:

Ontology Development: This layer covers the main minimal functionalities such as building the ontology models, validating, and/or maintaining the deployed ontologies.

Ontology Middleware: This layer implements the minimal functionalities such as the extraction of the ontological model knowledge, individuals insertion and extraction, reasoning on and about the ontology, querying the ontology model, etc.

Ontology Access: It consists of a set of modules that implement functionalities to walk through the ontology graph and the associated attributes. These functionalities could be used by a document access system to build the user interfaces, to browse the ontology structures, to implement an ontology driven search engine, and so forth. The ontology interface component can answer queries about the ontology class hierarchy and/or the class properties, giving back an RDF-XML document that could be transformed into HTML documents.

Ontology Repository: For the OWL/RDF data persistency we started with the Sesame package,[(6)] which we extended in order to expand its functionalities and improve the performance. We chose it because it is an open source, platform-independent, RDF Schema-based repository, equipped with an efficient querying facility written in Java. The low-level persistent storage is achieved using Postgresql.[(28)]

4.1. The Ontology Development Environment

To build an ontology model we rely on a group of domain experts equipped with the Ontology Development Environment (ODE). This environment covers the following main ontology lifecycle phases: building the ontology model, either from scratch or reusing parts of the previously developed ontologies, and validating and/or maintaining the deployed ontologies. Next, in order to insert the knowledge facts into the ontology repository we proceed as follows:

(a) insert the contents into buffered persistency area;
(b) validate and reason on and about the knowledge base;
(c) register the facts through the persistency module.

The identified ODE users are:

- the *editor*: as in the person in charge of defining the ontology model and inserting it into the ontology repository;
- the *reviewer*: as in the person in charge of validating and maintaining the ontology models;
- the *visitor*: as in the person and/or software components that can browse the ontology repository or walk the ontology classes/properties hierarchy.

All these operations are made available through the OS functionalities. Clearly, since they regard different kinds of users, different user authentication processes have been implemented.

We choose the Protégé-2000 ontology editor since it has an user friendly interface and a modular architecture that is extensible through a flexible programming interface. We developed an extension for the OWL Protégé-2000 Plug-in in order to store the ontology directly into the Ontology Repository using the client/server scheme, shown in Fig. 5.

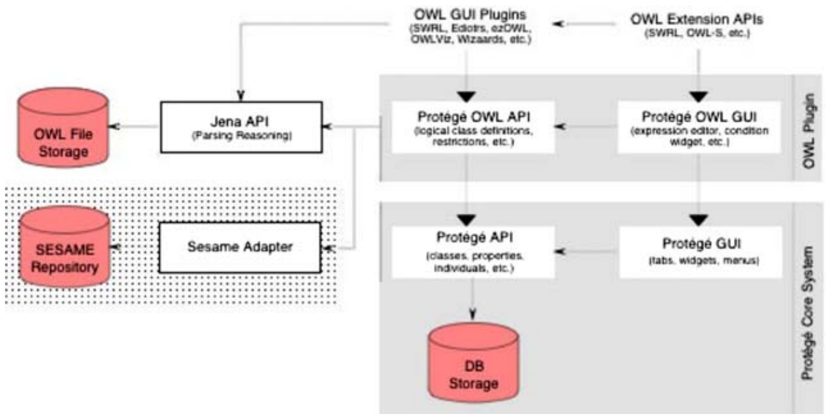


Fig. 5. The OWL plugin and *Protege2Sesame* tab architecture.

4.2. The Access Layer

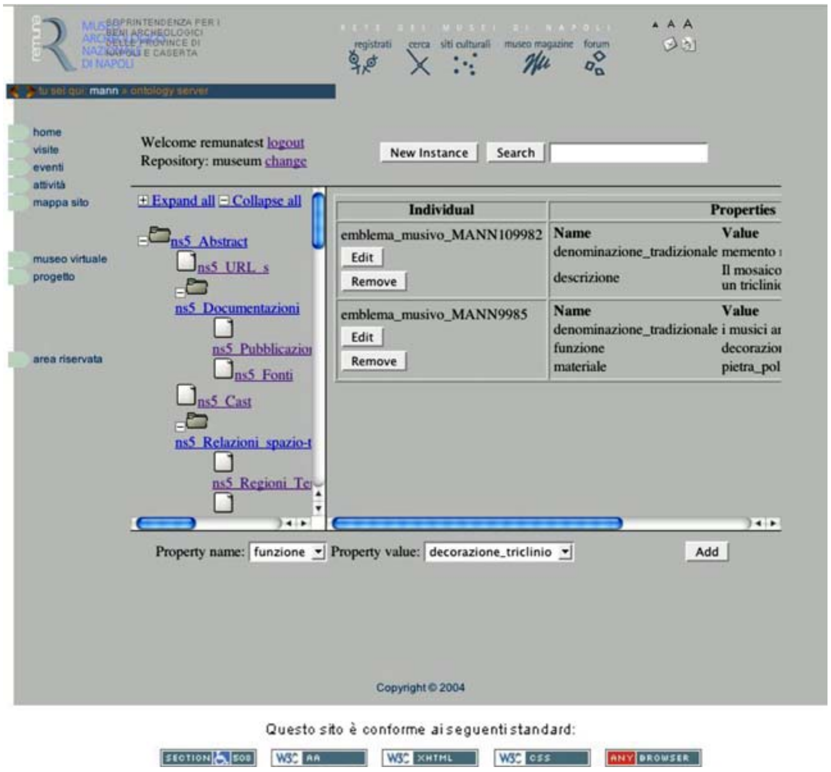
The access layer relies on the *OS Interface* and the *Query Engine*, both built on top of the persistency layer.

The OS Interface consists of a set of software components that implement functionalities for walking through the ontology graph and the associated attributes. The OS Interface can be queried about the ontology class hierarchy and/or the class properties, giving back an RDF document that could be transformed into HTML forms. To create, for example, an user interface to browse an ontology model it is necessary to create a set of dynamic forms, according to a classification schema, extracted from the corresponding ontology. Figure 6 shows an example of an ontology access session.

The *Query Engine* (*QE*) allows a generic user to query the OS without requiring any knowledge on the user's side about the Sesame querying language (SeRQL).(30) In order to achieve that we created a set of predefined query patterns, stored into a configuration file and we also built a *Query Composer Wizard* to assist the user in creating query patterns. A typical query scenario is: once a user has chosen an individual, through a free text search or by browsing the ontology, the QE proposes some predefined query patterns to the user. Patterns are modeled according to the considered class and/or property and to the effective presence into the repository.

4.3. The Ontology Server Middleware

The Ontology Middleware implements the minimal functionalities to manage the ontology lifecycle, such as the extraction of the ontological


SOPRINTENDENZA PER I BENI ARCHEOLOGICI DELLE PROVINCE DI NAPOLI E CASERTA

DI NAPOLI

tu sei qui: mann » ontology server

home
visite
eventi
attività
mappa sito
museo virtuale
progetto
area riservata

Welcome remunatest logout
Repository: museum change

New Instance Search

Expand all Collapse all

- ns5_Abstract
  - ns5_URL_s
  - ns5_Documentazioni
    - ns5_Pubblicazio
    - ns5_Fonti
  - ns5_Cast
  - ns5_Relazioni_spazio-t
    - ns5_Regioni_Te

Individual	Properties	
emblema_musivo_MANN109982	**Name**	**Value**
	denominazione_tradizionale	memento
	descrizione	Il mosaico un triclinic
emblema_musivo_MANN9985	**Name**	**Value**
	denominazione_tradizionale	i musici ai
	funzione	decorazio
	materiale	pietra_pol

Property name: funzione Property value: decorazione_triclinio Add

Copyright © 2004

Questo sito è conforme ai seguenti standard:

Fig. 6. An ontology access session example.

model knowledge (the ontology classes, and properties hierarchy), individuals insertion and extraction, reasoning on and about the ontology, querying the ontology model, etc. These functionalities are implemented through the following components:

Ontology Directory provides the metalevel knowledge about the available ontologies. By default, the ontology directory contains references to the top-level definitions of RDF, RDFSchema, XML-Schema, and similar knowledge. In addition, the ontology directory may contain deployment knowledge, giving additional information about the ontology knowledge sources. For example, for each ontology source the directory will need to store the URI, but may additionally store knowledge about the contents of the ontology source to aid the query optimization modules.

Inference Engine provides a mechanism for interpreting the semantics of an ontology language, represented as a set of language-specific rules. The rules are used to answer queries, when the requested fact is not immediately available and must be inferred from available facts. For example, if the application requests the `childrenOf` an individual, but the knowledge database only contains `parentOf` relations, the inference engine can use the inverse property of the `parentOf` relations to identify the correct response.

Query Optimizer Used when developing applications that connect large ontologies, since it would not be feasible to load the entire set of available knowledge into the knowledge database. In these cases, the system will query the ontology source for the appropriate knowledge on an as-needed basis. In addition, the query optimizer may be used to coordinate queries that span multiple sources.

Ontology Source Connectors provide a mechanism for reading, querying and writing ontology knowledge to persistent storage. The simplest connector is the file connector used to make persistent the knowledge into the local file system. There is also a connector for storing ontological knowledge into a database and remote servers.

4.4. The Persistency Layer

The Persistency Ontology layer is built on top of the Sesame(6) repository package, that in turn abstracts from the used database.

For the OWL/RDF data persistent storage, we choose the Sesame package. It is an open source, platform-independent, RDF Schema-based repository, provided with querying facility written in Java. The low-level persistent storage is achieved using Postgresql,(28) one of the most widely used public domain database environments. The Sesame environment offers three different levels of programming interfaces: the client API, for client-server programming, the server API and the lower level *Storage and Inference Layer* (SAIL) API, for the RDF repositories.

We also developed a buffered storage schema to account for the ontology workflow management. The first state of the workflow is associated with the ontology individuals insertion into the buffer, the second state is associated with the ontology validation process and the third state is associated with the storing process into the ontology repository. The buffer entry structure represents one or more classIndividuals, that in turn hold one or more class properties. Each individual is associated with the identifier of the user that inserted it and with the insertion date, which makes it possible to implement data consistency mechanisms.

5. THE MUSEO VIRTUALE DI NAPOLI TESTBED

In the context of the research project "Museo Virtuale di Napoli: Rete dei Musei Napoletani" (ReMuNa) [2] we built a community of semantic web-oriented Content Management System (CMS) for cultural heritage. One of the most interesting technological aspects that we investigated was how to design the ontology server component for the semantic web and knowledge grid applications. We used the ontology tools and server previously described to implement and exploit a CMS grid, where each system is used as a document repository that allows the museum manager to organize, as a whole, the cultural heritage and heterogeneous knowledge space spread in many autonomous organizations.

One of the most important constraints that we took into account was the fact that the aim of any ordinary museum visitor is something quite different from just trying to find certain objects in the web document space. In fact, in physical exhibitions the cognitive museum experience is often based on the thematic combination of exhibits and their contextual knowledge.

Furthermore, from the museum managers' perspective, each CMS should make available the managed artifacts' knowledge through the ReMuNa environment right after registering this information into the system. Knowledge is encapsulated into a digital object and no assumption about fixed attributes names' schemata is made, so that the application builder can create new attributes as needed, by just modifying the associated ontology without changing the internal database schemata.

Using the system that we developed, the knowledge provider [3] could also organize a set of related documents in document collections, according to some relationships defined on top of the associated ontology. The notion of "collection" is a recursive one, in the sense that a collection could contain other collections. Each digital document is allowed to belong to multiple collections and may have multiple relationships with other documents. These nesting features allowed us to deliver more than one logical view of a given digital documents asset.

Clearly, the deployment of the notion of collection strongly depends on the knowledge domain. Thus, it is necessary to guarantee an operational autonomy to the knowledge provider, without reducing the opportunities of cooperating with other knowledge providers. In other words, each

[2]This research project is supported by Ministero dell'Università, Ricerca e Tecnologia, under contract C29/P12/M03, from here on denoted withReMuNa, was carried out at the Istituto di Cibernetica "E. Caianiello".

[3]In this paper, we assume that *museum manager* means the people in charge of the cultural heritage knowledge about the goods, inside themuseum organization.

content provider will publish a set of ontologies to collect metadata information organized and published through a contents knowledge authority.

Summarizing, we have that, from a contents point of view, the distributed system is built as a collection of documents repository nodes glued together by an OS, where the *document* plays the role of elementary information and basic building block. The documents are represented as digital objects, together with the associated metadata information. The metadata is organized according to the associated domain ontologies where it takes values.

These methodologies were deployed and tested by setting up a prototype to connect about 20 museums in the city of Naples (Italy). Museums are equipped with multimedia knowledge systems and communication infrastructures. Those systems have different conceptual schemas and are physically located in different districts of Naples.

From the technological point of view, we adopted the multi-tiers web architecture, with the application server playing the central role of business logic driver. All the systems communicate among themselves by exchanging XML-encoded messages over http, according to well-defined protocols that represent the XML communication bus core (see Fig. 7). The user will interact with the community of the CMS through a conventional browser.

Each CMS grid node was designed around the following components:

- *Document Repository System* (*DRS*): which stores and organizes the documents together with the associated metadata, appearing and behaving like a traditional web site;
- *Document Access System* (*DAS*): which creates friendly and flexible user interfaces to discover and access the contents;
- *Contents Authority Management System* (*CAMS*): which stores and manages the ontologies used by each participating node to facilitate the DRS semantic interoperability.

The last software component is built on top of the Ontology Server described in Section 4.

To cope with the semantic interoperability issues, we designed a cultural heritage ontology that is empirical and descriptive. It formalizes the semantics necessary to express observations about the world in the domain of museum documentation, whose hierarchy is sketched in Fig. 8. It is composed of a hierarchy of classes, interlinked by named properties. It follows object-oriented design principle: the classes in the subsumption relation hierarchy inherit properties from their parents. Property inheritance means that both classes and properties can be optionally sub-typed for a

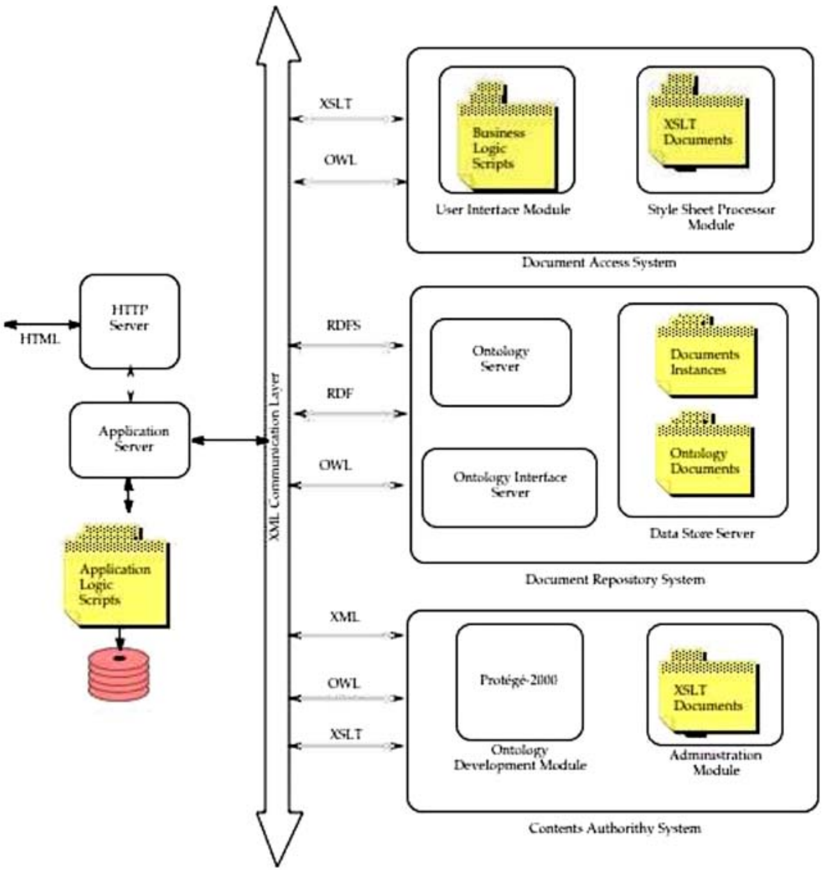
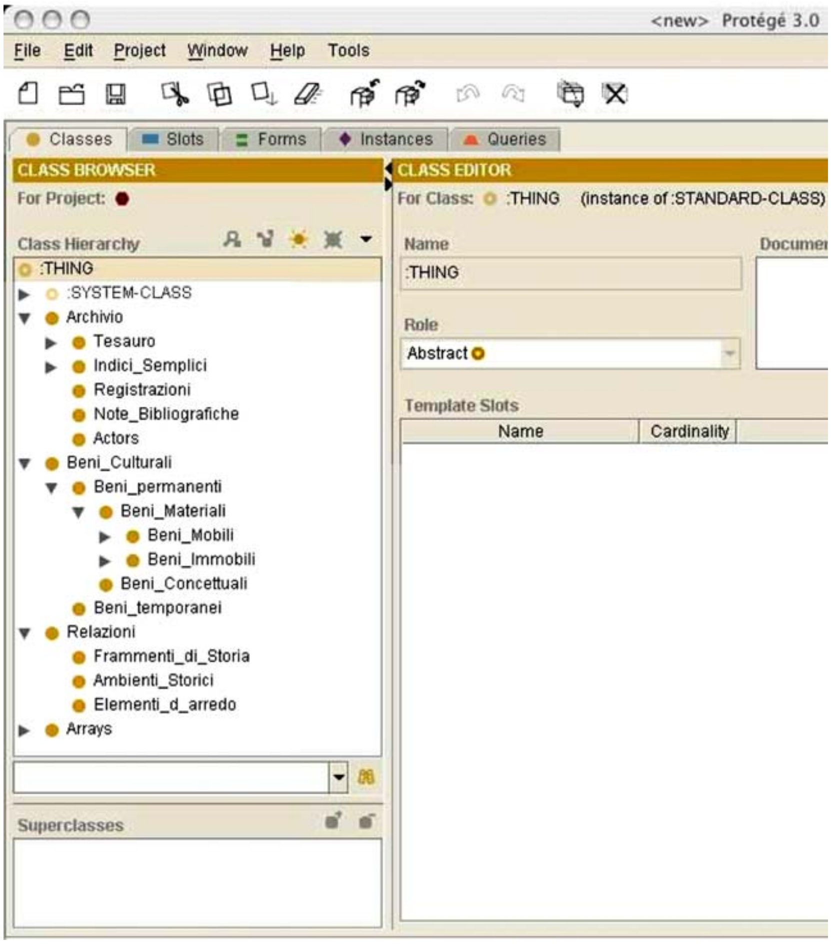


Fig. 7. Distributed contents management architecture.

specific domain, making the ontology highly extensible without reducing the overall semantic coherence and integrity.

The ontology is expressed according to the OWL semantic model. This choice yields a number of significant benefits; for example, the class hierarchy enables us to coherently integrate related knowledge from different sources at varying levels of detail. The developed domain ontology has many features in common with the CRM/CIDOC,(17) but it also covers the cultural heritage taxonomy aspects and the specific issues of an upper ontology.



For Class: :THING (instance of :STANDARD-CLASS)

Name: :THING

Role: Abstract

- :THING
  - :SYSTEM-CLASS
  - Archivio
    - Tesauro
    - Indici_Semplici
    - Registrazioni
    - Note_Bibliografiche
    - Actors
  - Beni_Culturali
    - Beni_permanenti
      - Beni_Materiali
        - Beni_Mobili
        - Beni_Immobili
      - Beni_Concettuali
    - Beni_temporanei
  - Relazioni
    - Frammenti_di_Storia
    - Ambienti_Storici
    - Elementi_d_arredo
  - Arrays

Fig. 8. Cultural heritage ontology - protégé screenshot

6. CONCLUSIONS

As the Semantic Web begins to fully take shape, the grid CMS implementation will enable agents to understand what is actually being processed, since all contents are modeled in machine understandable OWL/RDF. Our work successfully showed that an OWL/RDF data storage could be used as document repository backend for a grid CMS, where the ontology is used to cope with the semantic interoperability issues.

To a certain degree, our content metadata usage extends that of the CIMI project,[7] since we allow metadata to assume values into a specialized ontology and we do not consider metadata as syntactic entities. In fact, it has become increasingly evident that simple application-specific standards, such as Dublin Core (DC),[8] cannot satisfy the requirements of communities such as BIBLINK[4] and OAI.[25] As such, they are used to combine metadata standards for simple resource discovery process.

The exchange protocol used in our setting is similar to the Dienst[20] collection service, with the main difference lying in the fact that in our case the collections are entities built on top of a domain ontology describing the domain of the documents content and not predefined ones.

Starting from these encouraging results we are planning to actively pursue some of the goals foreseen by the Semantic Web Initiative.[3,18,19]

ACKNOWLEDGEMENTS

Acknowledgments are expressed to all the people of Istituto di Cibernetica E. Caianiello involved in the ReMuNa project, for their help, and fruitful discussions, and also to all the staff members of: Soprintendenza ai Beni Archeologici delle Province di Napoli e Caserta, Soprintendenza ai Beni Artistici, Storici e Demo Antropologici della Provincia di Napoli, Soprintendenza ai Beni Architettonici ed Ambientali della Provincia di Napoli, Archivio di Stato di Napoli, Direzione Musei of Comune di Napoli, and the Assessorato alla Cultura of Comune di Napoli. Without their assistance the ReMuNa project and activities would not exist.

REFERENCES

1. J. C. Arpirez, O. Corcho, A. Fernández–López, WebODE a Scalable Ontological Engineering Workbench, in *First International Conference on Knowledge Capture* (K–CAP 2001), Victoria, Canada, October (2001).
2. S. Bechhofer, I. Horrocks, C. Goble , R. Stevens, OilEd: Reason–able Ontology Editor for the Semantic Web, in *Proceeding of KI2001, Joint German/Austrian Conference on Artificial Intelligence*, Vol. 2174, Vienna, Springer Verlag, Berlin LNAI, pp. 396–408 (2001).
3. T. Berners-Lee, WWW: Past, Present, and Future, *IEEE Comput.*, **29** (1996).
4. The BIBLINK Core Application Profile, http://www.schemas-forum.org/registry/biblink/BC-schema.html
5. D. Brickley, L. Miller, The Friend of a Friend (FOAF) Vocabulary Specification: http://xmlns.com/foaf/0.1/.
6. J. Broekstra, A. Kampman, F. van Harmelen, Sesame: A generic architecture for storing a querying rdf and rdf Schema, in *The Semantic Web — ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pp. 54–68 (2002).
7. CIMI: Consortium of Museum Intelligence, http://www.cimi.org/
8. The Dublin Core Metadata Initiative, http://www.purl.org/dc/

9. J. Davis, C. Lagoze, The Networked Computer Science Technical Report Library, Cornell CS TR96-1595.
10. Extensible Style Language for Transformation, http://www.w3c.org/Style/XSLT
11. A. Farquilar, R. Fikes, I. Rice, Ontolingua Server: A Tool for Collaborative Ontology Construction, in *10th Knowledge Acquisition for Knowledge–Based System Workshop*, Banff, Canada (1996).
12. I. Foster, C. Kesselman, S. Tuecke, The Open Grid Service Architecture, in *The GRID 2: Blueprint for a New Computing Infrastructure*, I. Foster, C. Kesselman, (ed.), Morgan Kauffman, Los Altos, CA (2004).
13. The Gene Ontology Consortium. Creating the Gene Ontology Resource: Design and Implementation, *Genome Res.*, **11**, 1425–1433 (2001).
14. J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, B. Parsia, J. Oberthaler, The National Cancer Institue's Thesaurus and Ontology, *J Web Semantics*, **1**(1) (2003).
15. T. R. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, Presented at the Padua workshop on Formal Ontology (March 1993).
16. Lassila O., Swick R., Resource Description Framework (RDF) Model and Syntax, World Wide Consortium Working Draft.
17. ICOM/CIDOC Documentation Standard Group, Revised Definition of the CIDOC Conceptual Reference Model, (1999). http://cidoc.ics.forth.gr/
18. HP Labs Semantic Web Research, Jena-A Semantic Web Framework for Java, (2004). http://www.hpl.hp.com/semweb/
19. I. Horrocks, S. Tessaris, Querying the Semantic Web: a Formal Approach. in the 1st International Semantic Web Conference (ISWC2002), Sardinia, Italy, (2002).
20. C. Lagoze, E. Shaw, J.R. Davis, D.B. Krafft, Dienst: Implementation Reference Manual, May 5, (1995).
21. D. B. Lenat, Cyc: A large–Scale Investment in Knowledge Infrastructure, *Commun. ACM 38*, (11), (1995).
22. M. ango Furnari, A. Aiello, V. Caputo, V. Barone, Ontology Server Protocol Specification, ICIB TR-12/03.
23. M. Mango Furnari, A. Aiello, A. Massarotti, ezXML4OWL: an easy XML for OWL, ICIB TR-06/04.
24. D. McGuinness, F. van Harmelen, (eds)., OWL Web Ontology Language Overview, (2003). http://www.w3.org/TR/2003/WD-owl-features-20030331/
25. Open Archives Initiative, http://www.openarchives.org.
26. OWL Web Ontology Language Overview, http://www.w3.org/TR/2003/PR-owl-features-20031215/.
27. A. Pease, I. Niles, J. Li, Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications, *in the Working Notes of the AAAI–2002 Workshop on Ontologies and the Semantic Web*, Edmonton, Canada (2002).
28. http://www.postgresql.org/
29. http://protege.stanford.edu
30. http://www.openrdf.org/doc/sesame/users/ch06.html
31. A. L. Rector, A. Gangemi, E. Galeazzi, A. J. Glowinski, A. Rossi–Mori, The GALEN Model Schemata for Anatomy: Twoards a re–usable Application–Independent model of Medical concepts, in Published in P. Barahona, M. Veloso, J. Bryant (eds), *Proceedings of Medical Informatics in Europe MIE 94*, pp. 229–233, (1994).

32. http://www.openrdf.org/documentation.jsp.
33. Y. Sure, M. Erdmann, I. Angele, S. Staab, R. Studer, D. Wenke, OntoEdit: Collaborative Ontology Development for the Semantic Web, in *International Semantic Web Conference (ISWC02)*, Sardinia, Italy, LNCS 2342, pp. 221–235 (2002).