An Experimental Information Grid Environment for Cultural Heritage Knowledge Sharing

A. Aiello, M. Mango Furnari, and A. Massarotti

Istituto di Cibernetica E. Caianiello, Via Campi Flegrei, 34,
I-80078 – Pozzuoli, Italy
{a.aiello, mf, a.massarotti}@cib.na.cnr.it

Abstract. In this paper the authors address the problems of making existing distributed collection document repositories mutually interoperable at the semantic level. The authors argue that semantic web technologies offer a promising approach to facilitate homogeneous, semantic information retrieval based on heterogeneous document repositories on the web. From contents point of view, the distributed system is built as a collection of multimedia documents repository nodes glued together by an ontology server. A set of methodologies and tools for organizing the information 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 the Contents Community Authority on top of an ontology server. These methodologies were deployed setting up a prototype to connect about 20 museums in the city of Naples (Italy).

1 Introduction

In this paper the authors address the problem of making distributed document collection repositories mutually interoperable at semantic level. Furthermore, they argue that emerging semantic web technologies, more specifically the ontology one, offer a promising approach to facilitate semantic information retrieval based on heterogeneous document repositories distributed on the web. However, the current source ontologies exploitation attempts are oriented to cope with the conceptualization of single information source. Furthermore, most of existing tools treat ontologies as monolithic entities and provide little support for specifying, storing and accessing ontologies in a modular manner.

The authors' efforts described in this paper are based on the hypothesis that it is necessary to develop an adequate treatment of distributed ontologies in order to promote information sharing on the semantic web, and appropriate infrastructures for representing and managing distributed ontologies have also to be developed. To pursue these goals we defined a modularized ontology representation and developed an ontology server to deploy a knowledge repository community. An experimental implementation to verify the developed methodologies and tools within the cultural heritage promotion arena is also described.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 600–609, 2005.

The rest of the paper is organized as follows: In the first section the architecture and the implementation of the proposed Distributed Contents Management System are given together the Ontology Server architecture. In the second section a modular representation for the ontology structure is described. In the third section the implemented test bed is described. In last section the proposed architecture advantages are summarized and compared with other efforts.

2 The Distributed Contents Management System and Ontology Server Architecture

We chose the WWW paradigm as design criteria for a distributed contents management system, where the notion of *document* plays the role of elementary information and basic building block. Documents are represented as digital objects together with the associated metadata information, where the metadata are organized using domain ontology. Furthermore, we assumed the multi-tiers web architecture, with the application server playing the central role of business logic driver, where the main identified components are:

- *Document Repository System* (*DRS*). The DRS stores and organizes the documents together with the associated metadata.
- *Document Access System* (*DAS*). The DAS creates friendly and flexible user interfaces to discover and access the contents.
- *Contents Authority Management System* (*CAS*). The CAS stores and manages the ontologies used by each participating node to facilitate the DRS semantic interoperability.

All these systems communicate among them exchanging XML encoded messages over http, according to well-defined protocols that represent the XML communication bus core, see Figure 1.

The user will interact with the community of systems through a conventional browser; the DRS appears and behaves like a traditional web site. Documents must underway a text processing before to be displayed, and programmed according to a sequence of transformations expressed using the eXtensible Stylesheet Language Transformation (XSLT) [7]. The Document Access System manages this document composition process, whose business logic could be summarized as follows: the ontology client makes the first step by extracting the information from the data store and wrapping this information with XML tags. The extraction is done querying the ontology server. The second step involves the application of the appropriate stylesheet transformations to the XML data and thereby the creation of a corresponding HTML page. The foregoing step is carried out by the XSLT package included in the application server. The output of that transformation is the HTML page directly sent to the browser.

The advantages of the whole proposed architecture are: a) ease of deployment on Internet, high reliability and fault-tolerance, and efficient use of the network infrastructures; b) flexibility and generality as needed in order to evolve and

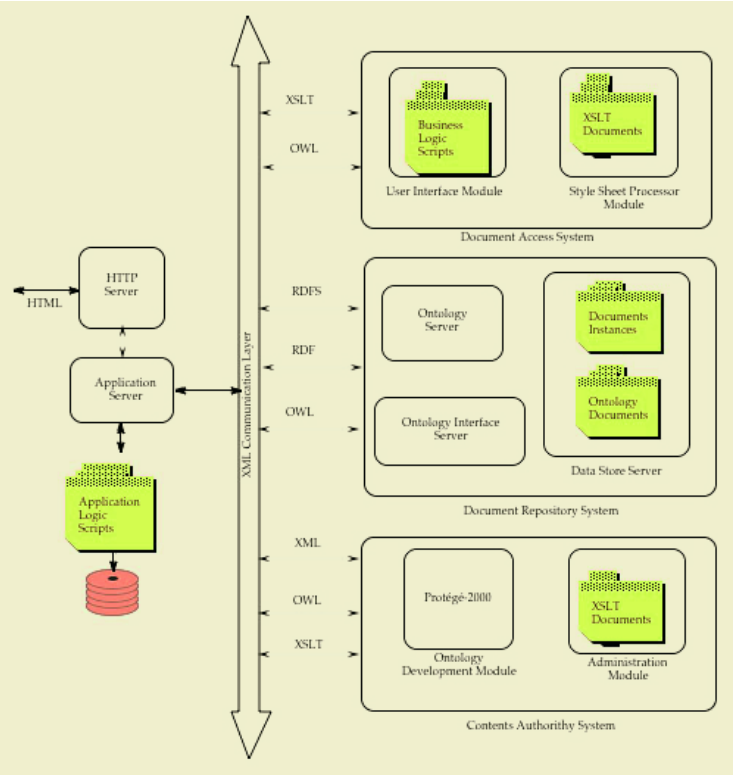

Fig. 1. Distributed Contents Management architecture

meet future needs; c) scalability without fundamental changes in the structure of the resource name spaces.

In the previous scenario the ontology server provides the basic semantic interoperability capabilities to build a Document Repositories Community. From the conceptual point of view the Ontology Server it is the most important type of servers since it manages the OWL/RDF [15] schema for the stored data, and determines the interactions with the other servers and/or modules, through the ontology exchange protocol [13].

The Ontology Server provides the information provider with the possibility of interacting with heterogeneous and distributed document repositories. Actually, it guarantees the necessary autonomy to the information provider in organizing their contents space. To achieve these goals the Ontology Server is equipped with the following modules:

– *Ontology Development Module.* The ontology development module is built around the Protégé-2000 [19] ontology editor, since its architecture is modular and extensible. We developed an extension for the OWL Protégé-2000 Plug-in in order to store the ontology directly on the Data Store Module using the client/server metaphor, see Figure 2.

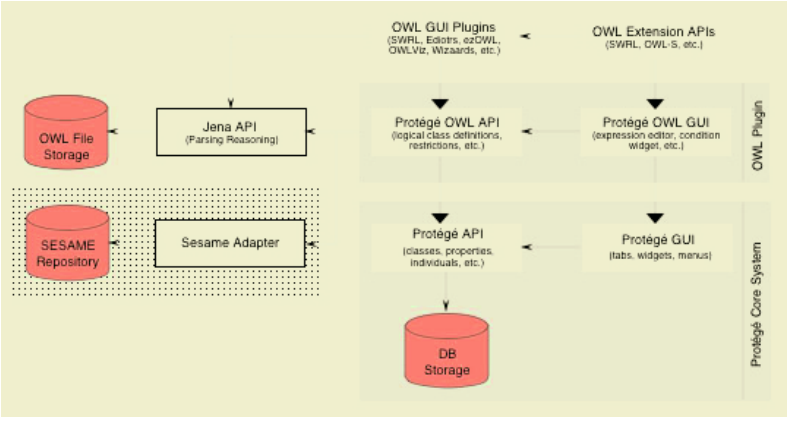

Fig. 2. The Plugin OWL and Tab architecture

- *Ontology Repository Module.* For the OWL/RDF data persistent storage we choose the Sesame package [3]. 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 [18], one of most widely used public domain database environment. 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.
- *Ontology Interface Module.* The Ontology Interface Module consists of a set of functionalities for walking through the ontology graph and the associated attributes. At runtime, 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 Module 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.

3 Ontology Modular Representation

The main purpose of building an ontology is to capture the semantics of the documents describing a given knowledge domain, especially the conceptual aspects and interrelations. We used OWL DL [17] to represent domain of concepts and relationships in a machines and humans understandable form. OWL DL is a rich ontology language with built-in semantics. It allows exploiting the well-defined semantics of description logics, where a reasonable upper bound is guaranteed for the complexity inconsistency, misclassifications, misunderstandings and deductions checking.

To cope with the interoperability problems related to exchange ontologies among cooperating information systems, we took into account the fact that interoperability worst case occurs when *useful* communication is restricted to transfer an ontology, as a whole such as happens with the currently serializing language and/or schema. By contrast, we may expect that transferring ontology in small *meaningful* chunks could significantly improve the knowledge system interoperability.

We defined a language for the XML serialization of the OWL DL ontologies, called *ezXML4OWL* [14]. The idea was to serialize an OWL mereology by mapping whole OWL ontologies to whole XML documents as well as parts belonging to the OWL mereology to the corresponding XML elements, all of them with the constraint that the relation *part-of* corresponds to the relation *XML-element-of*. Of course, it is not required that every XML-element occurring in ezXML4OWL have a correspondent in the OWL mereology. There are, indeed, auxiliary XML elements that have only serializing roles. Moreover, some redundancy was created to make our representation more understandable, and to make ezXML4OWL documents modular.

An ezXML4OWL document serializing an OWL DL ontology is composed of exactly the following three modules[1]: `<ontology>`, `<axioms>`, and `<facts>`. These modules are the top modules and like other modules and/or elements are recursively defined. The module `<ontology>` encodes metadata about the ontology, such as the name and the four OWL built-in ontology properties: `owl:imports`, `owl:priorVersion`, `owl:backwordCompatibleWith` and `owl:incompatibleWith`. The ontology's name and the associated properties are encoded according to the following skeleton:

```
<ontology name=ontologyID>
      <ontoProperty name= owl:priorVersion>
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:backwordCompatibleWith>
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:incompatibleWith>
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:imports>
```

[1] We use the term "module" for referring to ezXML4OWL elements that codify concepts belonging to the OWL mereology. Henceforth, the term "element" will refer to generic XML elements (not necessarily modules).

```
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
</ontology>
```

The module <facts> stores all the data gathered during the data entry phase. Since <facts> modules are about individuals, they might contain only modules of the type <individual>. Anyway, since each <individual> module would codify an instance of a class we defined a module, of the type <classIndividuals>, corresponding to the mereological entity "set of instances in the same class", and operating as a container for all <individual> codifying instances in the same class. <classIndividuals>. The fact module skeleton is:

```
<facts>
   <classIndividuals className>
      <individual name>
          <individualID_ValuedProperties>
               <property name >
                   <value name/>
                   ......
               </property>
               .....
               .....
          </individualID_ValuedProperties>
          <data_ValuedProperties>
                <property name >
                     <value name/>
                </property>
                .....
          </data_ValuedProperties>
      </individual>
      .....
</classIndividuals>\\
.....
.....
</facts>
```

We codify the different kind of ontology axioms in different modules, all being direct parts of the module <axioms>, where the module <classesLattice> explicitly describes the lattice formed by the classes associated to the ontology to be serialized. The lattice's structure is specified giving the direct subclasses of each class. The module <classesLattice> implements both the first two types of description prescribed by OWL: the class identifier (a URI reference) and the property rdfs:subClassOf. The module <classesSlots> codifies the classes and the related properties. There are modules for each type of OWL property.

Namely, the `owl:DatatypeProperty`'s are encoded in modules `<dataProp/>` and the `owl:ObjectProperty` are encoded in modules `<obProp>`. Attributes are also given to specify the range, the cardinality and the source (inherited or specific) of the properties, the remaining modules description and remarks can be found in [17].

The axioms module skeleton is:

```
<axioms>
   <classesLattice>
      <root name ‘‘classeID’’/>
      .....
      <root name =‘‘classeID’’/>
      <leaf name =‘‘classeID’’/>
      .....
      <leaf  name=‘‘classeID’’/>
      ....
      <sup name =‘‘classeID’’>
          <sub name =‘‘classeID’’/>
          .....
          <sub name =‘‘classeID’’/>
      </sup>
    </classesLattice>
    <classesSlots>
        <class name ‘‘classeID’’> ..... </class>
        ......
        <class name ‘‘classeID’’> ..... </class>
        .....
    </classesSlots>
    <subClassOf>          ..... </subClassOf>
    <enumeratedClasses>   ..... </enumeratedClasses>
    <equivalentClasses>   ..... </equivalentClasses>
    <disjointClasses>     ..... </disjointClasses>
    <objectProperties>    ..... </objectProperties>
    <datatypeProperties> ..... </datatypeProperties>
</axioms>
```

4 The Museo Virtuale di Napoli Testbed

The aim of any ordinary museum visitor is something quite different from trying to find certain objects. In physical exhibitions, the cognitive museum experience is often based on the thematic combination of exhibits and their contextual information. In order to figure out how much it would be complex to achieve these goals and which kind of technologies would be necessary, the research

project "Museo Virtuale di Napoli: Rete dei Musei Napoletani" (REMUNA)[2] is carried out at the Istituto di Cibernetica E. Caianiello. The collection of eighteen Neapolitan museums document repositories are used as case study. These repositories use different technologies, have different conceptual schemas and are physically located in different districts of Naples.

Each museum is equipped with multimedia information system and communication infrastructures. From the museum managers' perspective each information system allows him to make available the managed artifacts' information through the ReMuNa environment, just after registering them into the system. This information is encapsulated into a digital object that plays the role of a handle for the actual artifact information. No assumption about fixed attributes names' schemata is taken, so the application builder can create new attributes, as needed just modifying the associated ontology without changing the internal database schemata.

The information 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 adopted 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. This nesting features are represented by the document repository collection graph, and allows the system to deliver more than one logical view of a given digital documents asset.

To assure the necessary operational autonomy to the museum manager, without reducing the cooperation opportunities with other museum managers, we deployed this cooperation schema as an intermediate coordination organization that is in charge to register, syndicate and to guarantee the document contents quality, that we called Content Authority. The presence of the content authority could create a bottleneck; therefore the notion of delegation was introduced. In other words, the top authority could delegate another organization to operate as Cultural Heritage Contents Authority, on its behalf, for a more specific knowledge domain.

The domain ontology developed to exchange cultural heritage data has many common features with the CRM/CIDOC [9] have been developed. The designed cultural heritage ontology is empirical and descriptive one; it formalizes the semantics necessary to express stated observations about the world in the domain of museum documentation. It is composed of a class hierarchy, named classes interlinked by named properties. It follows object oriented design principle, the classes in the hierarchy inherit properties from their parents. Property inheritance means that both classes and properties can be optionally sub–typed for

[2] The project "Museo Virtuale di Napoli: Rete dei Musei Napoletani" is supported by Ministero dell'Università, Ricerca e Tecnologia, under contract C29/P12/M03, from here on denoted with ReMuNa.

[3] In this paper we assume that *museum manager* means the responsible, inside the museum organization, of the cultural heritage goods information.

specific applications, 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 yelds a number of significant benefits, for example the class hierarchy enables us to coherently integrate related information from different sources at varying levels of detail.

5 Conclusions

One of the most interesting technological aspects investigated was how to design document repositories systems that allow the museum manager to organize the cultural heritage heterogeneous information space spread in many autonomous organizations.

Ontology Exchange Protocol and tools were implemented to exploit the Multimedia Document Information System federation settlement. The ontology exchange protocol is very similar to the Dienst [12] collection service, where the main difference relies on 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. To a certain degree, our usage is similar to that of the CIMI project [4]. In fact, it has become increasingly evident that simple application-specific standard, such as Dublin Core (DC) [5], cannot satisfy the requirements of communities such as BIBLINK [2] and OAI [16] that need to combine metadata standards for simple resource discovery process.

Our work successfully showed that an RDF data store (Sesame) could be used as a backend document repository for a distributed Contents Management System (CMS), and the central role that the Ontology Server plays on deploying such kind of systems.

As the Semantic Web begins to fully take shape, this type of distributed CMS implementation will enable agents to understand what is actually being presented in distributed CMS, since all content within the system is modeled in machine understandable OWL/RDF.

Starting from these encouraging results we are planning to actively pursue some of the goals foreseen by the Semantic Web Initiative [1], [10], [11]. For example, to gain more semantic information we are exploiting pieces of well-known and supported ontologies, like ICOM-CIDOC [9].

Acknowledgment

Acknowledgments are expressed to all the people of Istituto di Cibernetica E. Caianiello that worked on the ReMuNa project, for their help, and fruitful discussions, and also to all the staff members of the Soprintendenza ai Beni Archeologici delle Province di Napoli e Caserta, Soprintendenza ai beni Artistici, Storici e Demo Antropologici della Provincia di Napoli, Soprintenda ai Beni Architettonici ed Ambientali della Provincia di Napoli, Archivio di Stato di Napoli, to

the people of 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. Berners-Lee, T., "WWW: Past, Present, and Future", IEEE Computer, **29**, (1996)
2. "The BIBLINK Core Application Profile", http://www.schemas-forum.org/registry/biblink/BC-schema.html
3. Broekstra J., Kampman A., van Harmelen F., "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)
4. "CIMI: Consortium of Museum Intelligence", http://www.cimi.org/
5. "The Dublin Core Metadata Initiative", http://www.purl.org/dc/
6. Davis J. and Lagoze C., "The Networked Computer Science Technical Report Library", Cornell CS TR96-1595
7. "Extensible Style Language for Transformation", http://www.w3c.org/Style/XSLT
8. Lassila O., Swick R., "Resource Description Framework (RDF) Model and Syntax", World Wide Consortium Working Draft
9. "ICOM/CIDOC Documentation Standard Group, Revised Definition of the CIDOC Conceptual Reference Model", 1999, http://cidoc.ics.forth.gr/
10. HP Labs Semantic Web Research, "Jena-A Semantic Web Framework for Java", 2004 http://www.hpl.hp.com/seweb/
11. Horrocks I., Tessaris S., "Querying the Semantic Web: a Formal Approach". The 1^{st} International Semantic Web Conference (ISWC2002), Sardinia, Italy, June 9-12, 2002
12. Lagoze C., Shaw E., Davis J.R. and Krafft D.B., "Dienst: Implementation Reference Manual", May 5, 1995.
13. Mango Furnari M., Aiello A., Caputo V. Barone V., "Ontology Server Protocol Specification", ICIB TR-12/03
14. Mango Furnari M., Aiello A., Massarotti A., "ezXML4OWL: an easy XML for OWL", ICIB TR-06/04.
15. McGuinness D., van Harmelen F. (eds)., "OWL Web Ontology Language Overview", 2003 http://www.w3.org/TR/2003/WD-owl-features-20030331/
16. "Open Archives Initiative", http://www.openarchives.org
17. "OWL Web Ontology Language Overview", http://www.w3.org/TR/2003/PR-owl-features-20031215/
18. http://www.postgresql.org/
19. http://protege.stanford.edu