A COOPERATIVE AND DISTRIBUTED CONTENT MANAGEMENT SYSTEM

C. Noviello, M. Mango Furnari and P. Acampa
Istituto di Cibernetica E. Caianiello, Via Campi Flegrei, 34, Pozzuoli, Italy

Keywords: Cooperative Content Mangement System, Information Grid, Cultural Heritage.

Abstract: In this paper the authors address some methodological and technical issues on managing collections of digital documents published on the web by many different stakeholders. To cope with this kind of problems the notions of document, cooperative knowledge community and content knowledge authority are introduced. Then the architecture of a distributed and cooperative content management system is presented. A set of methodologies and tools for organizing the documents space around the notion of contents community were developed. Each content provider will publish a set of data model interpreters to collect, organize and publish through a set of cooperative content management system nodes glued together by a web semantic oriented middleware. These methodologies and software were deployed setting up a prototype to connect about 100 museums spread on the territory of Campania (Italy).

1 INTRODUCTION

Thanks to diffusion of low cost high-speed Internet connections, institutions and organizations face increasing demands to cooperate in sharing common knowledge. Content delivery and sharing information across the network is today possible using a variety of technologies, such as distributed databases, service oriented applications, and so forth. However, sharing content technology is only one aspect of content management in cooperative and distributed settlements. In fact, contents need to be created, managed, revised and finally published. Contents may also to be aggregated in collections, which in turn may be shared among stakeholders. Modern Content Management Systems (CMS) have a complete environment to support users in content production and publishing for the web. Thanks to a powerful, complete and user-friendly interface it is very simple to create, manage and store contents. However, most CMSs have poor support for cross systems interoperability and cooperation over the network. They mainly focus the attention to the users interaction and documents usage.

Prerequisite to share documents in machine understandable way is the adoption of de facto standards for content and metadata representations. Aside of this standards, documents must provide both user and machine oriented representations, and play an active rule in data sharing. In other words it would be of some interest to explore the possibility to map the document conceptual model on the object oriented programming model exploiting some of its methodological and technological features.

Furthermore, the current semantic oriented exploitation attempts are mainly oriented to cope with the conceptualization of a single information source, they use document semantic models as monolithic entities providing little support to specify, store and access them in a modular manner.

In this paper we address the problem of making distributed document collection repositories mutually interoperable.

The design methodologies described in this paper are based on the hypothesis that it is yet necessary to develop an adequate treatment for distributed and heterogeneous document model interpretation to promote information sharing on the semantic web. Appropriate infrastructures for representing and managing distributed document model interpreters have also to be developed. To pursue these goals we introduced the notion of *knowledge stakeholders community* that exchange modularized *document model interpretation* together documents using a *document middleware*. Experimental implementation and tools were developed to check the adequacy of the proposed methodologies; their deployment for the cultural heritage promotion arena is also described.

The rest of the paper is organized as follows: in

Noviello C., Mango Furnari M. and Acampa P. (2008).
A COOPERATIVE AND DISTRIBUTED CONTENT MANAGEMENT SYSTEM.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 140-147
DOI: 10.5220/0001878501400147

section 2 the reference scenario is described. In section 3 the architecture and the implementation of the proposed Distributed Contents Management System are given together a description of the document modular representation model structure is described. In section 4 the implemented test bed is described and the proposed architecture advantages are summarized.

2 KNOWLEDGE COMMUNITIES AND CONTENT MANAGEMENT SYSTEMS

A knowledge community is created aggregating a group of information stakeholders that want to share and reuse their information contents so to improve their offer to their customers. On doing that they want retain their operational autonomy and identity without reducing the cooperation opportunities. To pursue these goals they need an intermediate coordination organization (*knowledge community authority*) in charge to register, syndicate and guarantee the interoperability of their document repository schema. Furthermore, the stakeholder managers would also be interested to organize a set of related documents into documents collections, according to some relationships. These collections should be built starting from documents either directly managed or shared with other participating stakeholders.

From the managers' perspective each information system should allow him to make available the managed information both to the other participating content managers and to the knowledge community customers just after registering it into the content knowledge authority. No assumption about information schema and attributes names' schemata should be taken, so the community could easily grow up and evolve.

We developed a system architecture where *document*s play the role of elementary information building block. We assumed that at least three main kind of information compose a document, they are: a) the *content*, i.e., the information to which it refers; b) the *metadata*, i.e., the information about the contents; c) the *relations*, i.e., the collection of relationships that could be instantiated among the documents. We also assumed that contents could be themselves documents, and then we can map the conceptual knowledge graphs into the document network relationships. Furthermore, *document collection* can be defined as graph that maps some instances of document relations, so we can associate a document to each collection. This recursive definition could be instantiated an arbitrary number of times, although in practical cases the recursion depth is not greater than three, i.e., document, the recursion base case; the collections of documents; document repository, collections of collections; and a community i.e., a collection of document repository.

From software point of view a document together its components are represented as digital objects. The digital object plays the role of handle to document components that are encapsulated into digital objects. To deploy this recursive definition process for the knowledge entities we need software tools to aggregate, both locally and remotely, the knowledge sources.

We adopted a multi-tiers web architecture in which the application server plays the central role of business logic driver. Three main systems were identified: *Document Repository System* (*DRS*), that it is in charge to store and organize the documents; the *Document Access System* (*DAS*), to create friendly and flexible user interfaces to discover and access contents; and the *Contents Authority Management System* (*CAMS*), that store and manage the document model schema used by each participating nodes so to facilitate the DRS semantic oriented interoperability. These systems communicate among them exchanging XML encoded messages according to well-defined protocols.

One of the foreseen scenarios is sketched in Figure 1, where the community is built with three stakeholders that manage heterogeneous information.

The end–user will interact with the knowledge community through a conventional browser. DAS is in charge for document access from the document repositories community and for the composition process in order to delivery documents to the end–users. Processing steps may be represented as

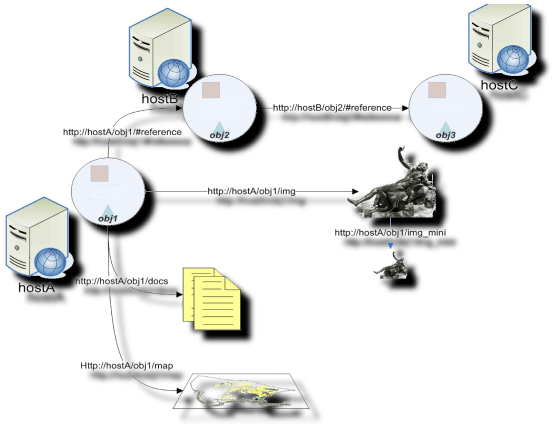

Figure 1: Heterogeneous data provider integration with Octapy based CMS.

sequence of transformations expressed, for example, using the eXtensible Stylesheet Language Transformation (Mangano, 2005). To assemble a document all its components and related ones must be fetched from the community of document repositories. In order to satisfy the partial instantiated document relationships the CAMS is invoked to filter, according to the document repository schema, the repositories to be involved on assembling the documents.

CAMS manages the document models expressed in RDF/OWL (McGuinness D. and van Harmelen F., 2003), since we don't constrains managers to adopt one common document model we associate to each document model an "interpreter" that allow to remap, for example, the values of metadata in some shared knowledge space. These interpreters are used by CMS nodes of the knowledge "community", and determines the pattern of interactions with other servers (Mango Furnari M. et al., 2003).

Once the subset of the knowledge community repository nodes is determined, each one is queried in order to give back the related documents to be sent to the DAS to complete the document assembling steps. Next, the documents are assembled into HTML page that will be 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 to meet future knowledge stakeholder needs; c) scalability without fundamental changes of resource name spaces.

Summarizing, to create a knowledge community we need software systems and middleware having a number of key features including:

- *Support to semantic web technologies:* it should push semantic web technologies into CMS, allowing interoperability with other systems. It should use document models suitable for semantic oriented interoperability.
- *Extensible metadata management:* the document models should contain metadata to be used to express any type of digital collections membership, parent-child or taxonomic relationships. Moreover, document repositories should be conformant with metadata exchange protocols, such as the Open Archive Initiative Protocol for Metadata Harvesting (OAI-PMH, 2001).
- *Multiple document representations:* for each managed document multiple representations it should be available that must be adequate for user and machine processing. Using the XML and RDF widespread standards, documents could be exchanged across different document repositories even if they didn't directly manage the model for the transferred documents.
- *Powerful component model:* it should have a flexible software component model in order to make easy the development of content-oriented extensions. Extensions that should be generic and not tightly coupled with the documents models.

In order to check the feasibility of the previously described architecture we designed and implemented the Octapy3 software platform, in which attention is payed to the necessary document repository functionalities so it could participate into a knowledge community. In the next section the Octapy3 design choices and its main components are described.

3 THE OCTAPY3 DISTRIBUTED AND COOPERATIVE CMS

The main Octapy3' functionalities are oriented to support: a) documents management and aggregation over the network; and b) an easy to use and fast process in order to define new document types. The way how to define document types has been one of the most intriguing aspect coped with on designing the software platform. In fact, we need to describe document and its components in such a way to be easily processed in order to produce software modules that manage them as digital objects. We approached this problem defining a document configuration language, called Octapy3 Content Markup Language (OCML), see Section 3.2. With this language we can define the document structure, the persistency, the presentation, and so on. These functionalities are built on top of a component model that could be easily extended. In Octapy3, contents play a central role since they are active parts of interoperability settlement among different systems. Contents aren't only implemented as data but are also active software components that expose a well-defined API, that allow implementing different kind of end–user and machine oriented documents representations.

Octapy3 has been designed to bring functionalities for content-based interoperability in order to create knowledge communities of distributed contents providers that share common knowledge. In Octapy3 the information sharing is achieved exchanging documents, each document has multiple representations, the default one is called Octapy eXchange Format (OXF), it is a serialization exposed by all digital object through the IOctapyContent interface. OXF exports both document contents and structures, i.e., the doc-

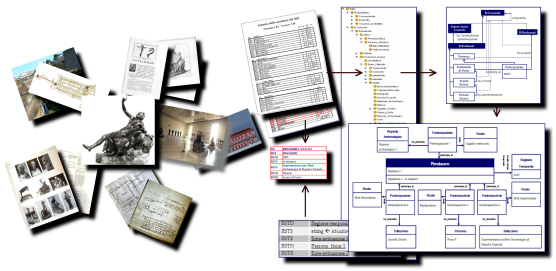

Figure 2: Heterogeneous data managed by Octapy based CMSs.

ument type definition. An OXF serialization allows to implement and manage remote contents considering them as local contents, even in the case the CMS doesn't have, locally available, the definition of the remote content type.

From an architecture point of view, Octapy3 have been organized around three main application levels:

- *Documents Definition Layer:* it contains software modules that add functionalities that simplify the definition of new document types. In Octapy3 the document definition is carried out during the configuration phase. To pursue this goal XML-based language Octapy Configuration Markup Language (OCML) was defined and used to specify both application and presentation logic.
- *Content Components Layer:* one of the main goals of Octapy3 is to abstract from the content structure introducing a clear separation among application layers. Starting from content description, Octapy3 generates specific software components, called "content component", that represent the managed contents. These components expose interfaces used, for example, to manage documents structure and documents relationships.
- *Distributed and Cooperative Layer:* Octapy is designed to build communities of cooperating knowledge node providers. Therefore, special attention has been paid to support standards for interoperability, such as XML for data representation, RDF/RDFS (Lassila O., 1998) for semantic interoperability, Dublin Core metadata set (DC, 1995), Open Archive Initiative protocol (OAI-PMH, 2001) for metadata exchange among hetereogeneous systems.

To achieve these goals we developed Octapy3 adopting the Component Architecture of the Zope3/Plone software platform (von Weitershausen P., 2007). On top of this architecture we first developed the "content components" to implement the document type definition process, and next the middleware to make location independent the document process management.

In Octapy3 documents are implemented as classes, more specifically the class OctapyContent implements the document living only in each document repository, i.e., locally. The classes RemoteObject and OctapyProxy are used to manage, on a given repository node, documents located in the rest of the knowledge community nodes.

In the rest of this section we will describe the main designed and implemented software components. We start shortly summarizing the components that implement the "content component". Next we focus on the implementation of the distributed and interoperability functionalities..

3.1 The Octapy3 Content Component

The Octapy3 platform is built using a Component Architecture model mechanism, in particular for documents that are the community knowledge building blocks. The Component Architecture make possible to abstract from the specific content schemata. We developed a special Octapy3 component, called "content components", that it is in charge to automatically generate software components starting from document type description written using the OCML language, see section 3.2 below. The software generation process is carried out using YODA (Yoda is Octapy Document Assembler) compiler that generates both the archetype (von Weitershausen P., 2007) code and the interfaces describing the document structure. This interface is an enumeration of attributes for each Content Type Description field and the corresponding archetype class that implements this interface, called "*content component*".

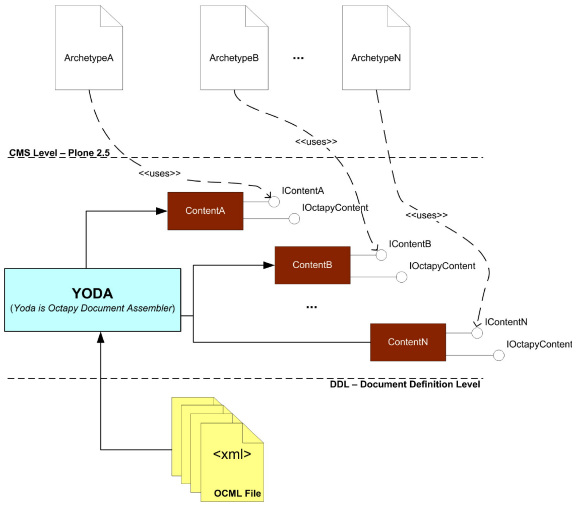

Figure 3: The Octapy component model.

A content component implements, see Figure 3, at least two interfaces: the content type definition and the IOctapyInterface. This interface is generic in the sense that it is common to all content components and exposes the method `getContentInteface()` and returns the interface describing the document content type, i.e. it allows searching for a specific data interface.

Octapy3 allows also handling and managing content type extensions in a generic way. An adapter, to the fixed interface IOctapyContent, could add the required functionalities to the Octapy3 system. For example, a presentation interfaces can be built using Browser Pages component that adapts the interface IOctapyContent and accesses the content interface via `getContentInterface()` method. Furthermore, it is also possible to write extension module for a given data schemata adapting only the generated content component interfaces.

3.2 The Octapy3 Configuration Mark-up Language

In order to extend, during the CMS configuration phase, the types of the managed contents we defined a specific document configuration language, called OCML. It allows: to define new document types; to choose documents storage methods for each document or some of its parts; to assign some sort of semantics to the document components associating specific interpreters; and to assign user presentation logic for each content types.

OCML is a command–oriented language whose syntax is inherited from XML. The commands are represented as set of XML tags, called directives, and are grouped in three main XML vocabulary identified by the XML namespaces: `data`, `storage` and `view`. A command interpreter may be associated to each directives in order to assign a specific operational semantic.

The `data` directives allow defining new content types and express the parent-child relationship among documents. A new content type is defined using `data:document` directives, and the `data:field` directives associate to them a structure. Fields can be logically grouped using `data:section` directives. For example, documents can be composed (nested) to form hierarchical structures instantiating the containment relation "*is-composed-by*" and the corresponding "*is-part-of*". A document could also contain other documents describing the containerish document type.

The `storage` directives associate a specific storage mechanism to a document or to some of its parts. For example, suppose that a structured document, called ArtisticObject was defined, and that among its fields there is one, called `image`, to which is associated a jpeg digital picture. Then the document may have database persistence for all its fields except for the `image` field whose persistence could be in a local filesystem.

The `view` directives associate one o more ways to present documents to the end–users. It's possible to define which widget to use in order to display the document field. Two directives can be used to cover this operation: `view:for`, used to specify the widget to be associated to a specific field; and `view:fordata` used to specify the widget to be associated to a given field type (e.g. Text, Image, etc).

OCML documents specification may also be split on multiple files using the special directive `<include src=''filename''/>`. More details on OCML can be found in (Acampa P. and Noviello C. , 2007).

3.3 The Metadata Attribute of Document Models

The content collection of fields of the application data layer have, in general, no associated meaning, since they are considered only containers for "values" and used to store fixed data. This information is, in general, interpreted as expressed by a pair of a *name* and uninterpreted *value*s. Nevertheless, in order to be semantic oriented an interpreted value must be assigned to a metadata name, i.e., it is necessary to make explicit both the domain from which the values are chosen and the valuation function (interpreter) used to assign a meaning. In a such semantic oriented scenario it is possible to define ArtisticObject content type whose fields `oss` and `title` are not only simply container for lines of text, but it is also possible to associate meanings that depend on the context. For example, the field oss may be interpreted as the description of an Artistic Object in one context and as picture caption in another context, with a different formatting and typesetting. This means that a document, or part of it, can have special interpretation that must be correctly handled by the software.

In OCML the `metadata` attribute of the directive `data:field` has been introduced, whose values can be used to specify which interpretation model has to be associated to a field. For example, the value: `<data:field name="oss " type="Text" metadata="{uiuse:description}"/>` instructs the presentation layer, where the field is used, that the field `oss` must be interpreted as the Artistic Object field `description`, and therefore it must be correspondently processed.

The values of metadata attribute are transparent to the data and other application layers: only com-

ponents that know how process it will use this information.

The interpretation model is implemented annotating the document component model attributes and methods with tags. The YODA generated content interfaces are annotated with the information provided by the `metadata` attribute using `setTaggedValue()` method. Extension modules can access this information through metadata attribute using the `getTaggedValue()` method.

It is important to underline that interpretation models are not only used to implement the presentation logic, but that may be also used whenever it is necessary to associate a special meaning to a field, some sort of *metadata cross-walk*. For example, the Octapy3 Dublin Core subsystem uses `metadata` attribute to map fields name to the DC metadata set, as shown in the following example:

```
...
  <data:field name="descrizione_breve"
              type="Text" metadata="{dc:title}"/>
  <data:field name="autore_scheda"
              type="Text" metadata="{dc:author}"/>
...
```

In the next section the Octapy3 test bed and the experience gained operating it are shortly described over a knowledge community of about 100 Octapy3 nodes.

3.4 Cooperation and Interoperability Components

To make effective the cooperation and to share documents across documents stakeholders, Octapy3 makes available different machines "understandable" document representations. Some of them are oriented to semantic web and are based on RDF/RDFS language and derivatives; others are oriented to facilitate the interoperability using standard protocol for metadata sharing, like the OAI-PMH protocol. Figure 4 shows the OAI Octapy subsystem components interaction, more information can be found in (Noviello, 2007).

In Octapy3 every containers implements the IOctapyContainer interface and all remote contents (documents and/or aggregates) implements the IOctapyProxy marker interface, i.e., those objects act as proxy for remote contents. Documents and their aggregates can be exported and/or aggregated in other CMS nodes using the RemoteContainer class. A RemoteObject special base class, called OctapyProxy, manages the content interface creation, starting from the information provided by OXF serialization. All the document interpretation models are properly added so presentation layer and other modules can correctly process them.

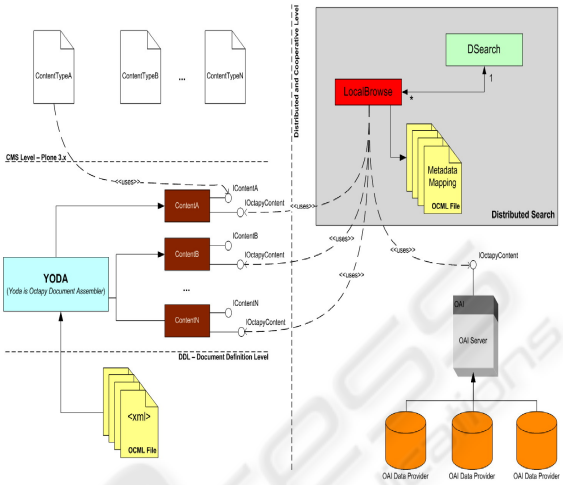

Figure 4: The integration of a OAI data provider in a Octapy based CMS.

4 THE MUSEO VIRTUALE TEST BED

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 both thematic combination of exhibits and their contextual information. To foster the museums cooperation software tools were developed to aggregate, both locally and remotely, the knowledge about cultural heritage goods. Using these tools the knowledge stakeholders could organize virtual exhibitions according to some physical or logical criteria either in the case either the information is directly managed or shared with other stakeholders.

The information provider[1] could also organize a set of related documents, as document collections, according to some relationships.

To assure the necessary museum manager operational autonomy, without reducing the cooperation opportunities, we deployed a cooperation schema as intermediate coordination organization that it is in charge to register, syndicate and guarantee the quality of document contents.

From technical point of view the main goal pursued with this test bed was to concretely verify the

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

possibility to create knowledge cooperating communities. Where each participating museums could exchange its own managed knowledge so to improve their institutional cooperation.

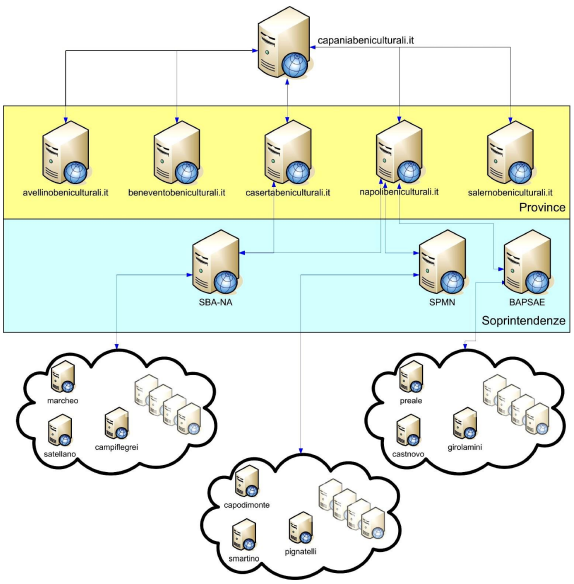

Figure 5: The organization of `campaniabeniculturali.it` knowledge community.

Figure 6: The content integration of the cultural heritage heterogeneous knowledge.

Currently more than 100 Octapy3 nodes, spread over the geographic region of Campania, are organized in knowledge clouds, each cloud covers the territories around the main cities of Campania, i.e. Benevento, Salerno, Caserta and Avellino. The cultural heritage knowledge offering is organized according thematic topics, such as the *first civilizations in Campania*, the *Roman civilization periods*, the *Gran Tour period* and so on.

From the museum managers' perspective each information system allows him to make available the managed artifacts' information through the Octapy3 node, where no assumption about fixed attributes names' schemata is taken, so the application builder can create new attributes, as needed just modifying the associated document interpretation model without changing the internal CMS schemata.

Since the software allows exchanging the contents through the OAI-PMH then the circuit itself is included in the Italian cultural heritage portal.

5 CONCLUSIONS

One of the most interesting technological aspects investigated and described in this paper was how to improve the document repositories systems flexibility to organize cultural heritage etherogeneous information spread in many autonomous organizations. To achieve this goal we developed, first of all, a document markup language to describe both document contents and then the corresponding software components devoted to manage the document life cycle. Next, we developed a middleware layer that make transparent the document physical location into a "knowledge community network" to easily share document metadata. On doing so, each museum document repositories may have different conceptual schemas, and no assumption about a fixed attribute names schemata are taken, so museum manager can create new attributes as needed just modifying the associated OCML document type description without change neither the internal software component nor database schemata.

The information provider could also organize a set of related document in document collections according to some relationships defined using the document container type with an arbitrary nesting level. With this feature the museum manager could define collection of cultural heritage description that contain inside other collections descriptions. Each digital document could belong to multiple collections and have multiple relationships with other documents. This nesting feature forms the document repository collection graph, and allow to deliver more than one logical view of a given digital documents asset.

Our work successfully showed that Octapy3 could be used as document repository backend for distributed CMSs, together the central role played by the cooperation middleware on deploying such kind of systems. With this type of CMS we could overcome some limitations suffered by the system described in (Aiello A. et al., 2005), (Aiello A. et al., 2006).

Where a "cultural heritage knowledge community infrastructure" was build around the use of an ontology to glue distributed document repositories. Repositories that didn't gave good performances when used in conjunction with a reasoning component. That was especially true on increasing both the number of documents and CMS nodes.

We have plans to extend Octapy3 to manage document schema and document exchange using RDFS/RDF representations to actively pursue some of the foreseen goals off semantic web technologies (Berners-Lee T., 1996), (Jena, 2004), (Horrocks I., 2002) in a distributed settlement. For example, we foresee the possibility to exploit pieces of well-known and supported ontology fragments, like ICOM-CIDOC (CIDOC, 1999). Fragment to be specialized for each different knowledge providers in order to contain the computational complexity on using ontologies in real cases.

ACKNOWLEDGEMENTS

Acknowledgments are expressed to all members of Advanced Information System research group team of Istituto di Cibernetica E. Caianiello that contributed to the Octapy3 paltform, for their help, and fruitful discussions. Also to 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, a special gratitude must be expressed since without their assistance the experimental activities and results would not exist

REFERENCES

Acampa P. and Noviello C. (2007). Specifica ocml, technical sheet octapy cms. Technical report, Istituto di CIbernetica.

Aiello A., M. Mango Furnari M., and Massarotti A. (2005). A distributed multimedia information system for cultural heritage identity preservation. *Transaction on Internet Research*, pages 11–17.

Aiello A., M. Mango Furnari M., Massarotti A., Caputo V., B., and Barone V. (2006). An experimental ontology server for an information grid environment. *Int. Journal on Parallel Programming*, pages 489–508.

Berners-Lee T. (1996). Www: Past, present, and future. *IEEE Computer*, 29:69–77.

CIDOC (1999). Icom/cidoc documentation standard group, revised definition of the cidoc conceptual reference model `http://cidoc.ics.forth.gr/`.

DC (1995). The dublin core metadata initiative `http://www.purl.org/dc/`.

Horrocks I., T. (2002). Querying the semantic web: a formal approach. In Horrocks, I. and Hendler J, editors, *The 1st International Semantic Web Conference - ISWC2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 177–191. Springer Verlag.

Jena (2004). Jena-a semantic web framework for java `http://www.hpl.hp.com/seweb/`. Technical report, HP Labs Semantic Web Research.

Lassila O., R. S. (1998). Resource description framework (rdf) model and syntax. Technical report, World Wide Consortium Working Draft.

Mangano, S. (2005). *XSLT Cookbook*. O'Reilly, 2nd edition.

Mango Furnari M., Aiello A., Caputo V., and Barone V. (2003). Ontology server protocol specification. Technical report, ICIB TR-12/03.

McGuinness D. and van Harmelen F. (2003). Owl web ontology language overview. Technical report, W3C.

Noviello, C. (2007). Il component-model di octapy 3, technical sheet octapy cms. Technical report, Istituto di Cibernetica.

OAI-PMH (2001). Open archives initiative.

von Weitershausen P. (2007). *Web Component Development with Zope 3*. Springer Verlag, New York, 2nd edition.