WS-DAIOnt-RDF(S): Ontology Access Provision in Grids

Miguel Esteban Gutiérrez 1, Asunción Gómez-Pérez 2, Oscar Corcho 3, Óscar Muñoz García 4

1 Ontology Engineering Group, Universidad Politécnica de Madrid
Campus de Montegancedo s/n, 28660, Boadilla del Monte, Madrid - Spain
1mesteban@fi.upm.es
2asun@fi.upm.es
4omunoz@fi.upm.es

3 School of Computer Science, University of Manchester
Oxford Road, Manchester, UK
3ocorcho@cs.man.ac.uk

Abstract—This paper presents WS-DAIOnt-RDF(S), a system that provides an homogeneous access mechanism for using heterogeneous and distributed RDF(S) ontologies in Grid applications. These ontologies may be stored in different RDF(S) storage systems, geographically distributed across the Grid infrastructure. The paper describes in detail the design of the system, its rationale, the current implementation, and the ongoing standardisation effort.

I. INTRODUCTION

Nowadays, semantic technologies are employed and can be found computer related field, including grid computing. The next generation grid should virtualize the notion of distribution of computation, storage, and communication on unlimited resources with well defined computational semantics. A grid node may provide new services, functions or, even, new concepts unknown to clients. The semantics of such services is defined by means of ontologies [1], which are formal and explicit specification of shared conceptualisations.

Ontologies could be used in the grid for several purposes, such as describing policies, sharing information, services, and computing resources in virtual organizations, describing formal and non formal properties of grid resources and services, accessing data catalogues in a conceptual and multidimensional way, etc.

Right now, only few grid applications use ontologies, but the access to these ontologies is not integrated in the grid infrastructure. Therefore, if the ontology server providing such ontologies is down (or cannot provide them for any any reason), the ontology-based grid application cannot use them, so they will not work properly.

Within the grid computing area, the current grid architecture does not consider ontology use. On the other hand, there are no protocols nor standards available within the grid community for tackling this issue. Therefore, if the grid is to profit from the semantic technologies already available in other areas such as the Semantic Web [2], it is top priority to provide proven value mechanisms for accessing and managing ontologies in grid environments.

In this paper we discuss the issue of providing ontology access to grid environments, specifically, to the grid infrastructure defined by the Open Grid Services Architecture (a.k.a. OGSA) [3], and present an approach for accessing RDF(S) ontologies in the Grid, which is the first step for giving access to ontologies defined in other languages such as OWL.

The paper is organised as follows: first, we introduce those technical topics that should be known beforehand to fully understand the work here presented; second, we describe the general requirements for using ontologies in the grid; third, we provide an in-depth study of the problem encountered when providing ontology access within OGSA; fourth, we explain an RDF(S) ontology access mechanism, taking into account the requirements presented before. Fifth, we give details of the current implementation and describe how we plan to experiment with it; and finally, we draw the conclusions and propose future work.

II. RELATED TECHNOLOGY OVERVIEW

Before presenting the topic of the present work we think it is necessary to review briefly some of the technologies and topics which conform the basis over which the work relies on.

Therefore, we review some of the works concerning ontologies carried out in the context of the Semantic Web. Then, we outline the Open Grid Services Architecture, which is the de-facto grid architecture.

A. Ontologies and the Semantic Web

The Semantic Grid community [4] did not start from scratch when it adopted semantic technologies in general and ontology use in particular.

The Semantic Web develops languages and tools for building and using ontologies, which are the basis of semantic technologies. The W3C recommends to use three languages when implementing ontologies in the Semantic Web (RDF, RDFS, and OWL), whereas several ontology development tools (i.e., Protégé, WebODE, KAON) support the creation of ontologies in such languages. In addition, some tools
have been developed for querying ontologies and performing inferences with them.

Generally, these development tools are tightly coupled with the language in which the ontology is implemented. Such languages differ in their expressiveness (the kind of knowledge that can be represented with them) and in their inference mechanisms (the kind of reasoning they apply). For a detailed description and comparison of languages and tools we recommend [1].

However, the diversity of ontology languages and tools produces the translation problem, a problem that appears when an ontology developer wants to reuse an ontology (or part of it) with a tool or language different from that in which the ontology is available.

On the other hand, different APIs permit accessing ontologies implemented in a given language, each of them tightly coupled with the underlying language and with the tool operational model. Therefore, the ontology user (or the client application using the ontology) not only must know about the ontology internal structure, but also how to retrieve the ontology content according to the API of the concrete tool to be used for storing the ontologies.

For example, we can say that in the case of RDF(S) ontologies, several storage systems can be used for storing ontologies (Sesame, 3store, Jena, Jowri, OracleRDF, . . .), and each of them has its own primitives for accessing the container. Therefore, changing the storage used in an application would imply changing the business logic that performs the interaction with the RDF(S) storage.

Currently, the Semantic Web community cannot offer a standard mechanism or protocol for accessing ontologies implemented in a given ontology language in a storage-and-retrieval system independent fashion, which leads to severe interoperability problems, as can be seen in the RDF(S) example described above.

Therefore, to be able to use semantic technologies in the grid, we must face and solve these interoperability issues. Thus, if semantic technologies are to be used, it is necessary to provide means for accessing and using ontologies in the grid.

B. The OGSA Grid Infrastructure

In order to integrate an ontology access protocol in the current Grid, the Open Grid Services Architecture [3], the current de-facto reference grid architecture, must be examined. OGSA represents an evolution towards a service-oriented grid system architecture based on web services concepts and technologies. The OGSA specification is divided into two parts: the first focuses on the set of requirements that must be addressed by the architecture; the second is focused on the set of capabilities offered by the architecture, which altogether define OGSA and are deeply related to the requirements specified in the first part.

These requirements are: supporting dynamic and heterogeneous environments and guaranteeing the interoperability between them; providing resource sharing across organizations; offering optimised service and assuring their final quality; enabling job submission; accessing data sources and providing consistency, persistency, integration and location management; setting up a secure environment; automating administrative operations and consistent management of virtualized resources; and supplying a scalable and extensible framework which allows creating high-availability systems.

OGSA delivers its capabilities as services, which are grouped according to the requirements they are targeted to. The main groups are: infrastructure services, job management services, data access and integration services, resource management services, security services, self-management services and information services.

III. Ontology Usage Requirements for the Grid

After reviewing the requirements and functionalities provided by OGSA, we can add that the framework does not consider ontology usage among its requirements, therefore, infrastructure is provided.

The functionalities required for using ontologies in a grid environment, can be simply defined by listing the logical steps that a fictitious user of a virtual organization would follow to use an ontology1:

- **Discovery.** The first step is to discover which ontologies are available in the virtual organization. Therefore, it is crucial to provide the means for publishing ontologies in the virtual organization. To publish ontologies implies the usage and maintenance of a (ontology) registry, which would contain information about the ontologies available in the virtual organization and whose use would allow users to discover the ontologies.

  The information contained in the registry should permit describing different characteristics of the ontology, such as authoring, localisation, purpose, etc. The Ontology Metadata Vocabulary [5], [6] defines the vocabulary used when describing ontologies in the context of the Semantic Web. Therefore, the use of this vocabulary as the information set of the registry is the most sensible choice among the existing alternatives available for describing ontologies, i.e. the Dublin Core standard [7] or the SchemaWeb Directory metadata2.

  The basic functionalities that the registry should provide are the basic BREAD ones, that is, those that allow browsing, reading, editing, adding, and deleting the metadata stored in the registry.

- **Access.** After discovering and localising the ontology to be used, it is necessary to access to the contents of the ontology itself. Accessing the ontology contents means being able to create, retrieve, update and delete the set of modelling components that are used in the ontology.

  In the following sections we focus on ontology access, analysing first the possibilities for providing ontology access in

1 Notice that we are speaking at a logical level, so the steps do not have any technical prerequisite, as they are not bound to any technical infrastructure.

2 http://www.schemaweb.info/
OGSA, and then describing a concrete mechanism for RDF(S) ontology access.

IV. PROVIDING ONTOLOGY ACCESS IN OGSA

Up to now, the current OGSA architecture has not considered ontology access, and no protocols nor standards are available in the grid framework defined by OGSA for tackling this access issue.

An ontology can be seen as a structured set of data in which both the schema that defines the structure and the data organised according to the schema are of great interest. According to this view, the most sensible mechanism for providing ontology access services is through the existing infrastructure in OGSA for data access and integration.

In the OGSA framework, the data access services are defined by the Web Services Data Access and Integration Core specification [8], a.k.a. WS-DAI. This specification provides a data access framework, in which existing data access techniques supported by data resources are exposed to the user via a web service–based infrastructure.

The main components of the WS-DAI specification are the following: interfaces, which are the sets of messages and properties (XML structures) that describe how a user can interact with a data service; the data services, which are plain web services that implement any of the WS-DAI-based interfaces for providing access to data resources; and finally, the data resources, which are systems that consume and produce data.

The top level WS-DAI specification provides a basic and extensible framework for data service interfaces and for properties definition. With this framework we can define in a standard way the set of interfaces, messages and properties needed for giving access to specific data resources. However, WS-DAI does not describe the particular interactions with the data service (in terms of, for example, query languages).

The underlying WS-DAI realizations describe the specific operations needed to interact with specific data resources, such as relational databases, XML sources, etc. The upper level specification is irrelevant to the types of queries that pass through – they say nothing about the query language, result formats etc. The specific interfaces, messages, and properties used must be defined according to the WS-DAI framework. For example, the WS-DAIR realization provides a relational database access mechanism [9], the WS-DAIX realization provides a XML access mechanism [10], etc.

Therefore, the most suitable way of integrating such an ontology access mechanism in OGSA is by incorporating the mechanism to the WS-DAI family of data access specifications. And this is the approach followed for defining the access mechanism and on how the WS-DAI framework is used. Alternative approaches for designing highly interoperable ontology access mechanisms are discussed in [11].

The approaches identified in the previous work are the following: the vanilla approach, in which an individual WS-DAI realization of WS-DAI is created for providing specialized access to a concrete ontology language, focussing the realization on the particularities of that language. The two-layer realization approach, which proposes factoring the common operations of related ontology languages in an upper layer realization and providing the specific operations of each language in specialized lower layer realizations. The extensional approach, which explores the possibility of extending the WS-DAI framework with specific structural elements for defining ontology access mechanisms so that some kind of standardisation can be enforced. Finally, the abstract realization approach, which presents the idea of providing a means for defining the capabilities that might be offered via an ontology access service with the mechanisms in which they are provided, with a mechanism for publishing the specific capabilities supplied by an ontology access service that a client can discover and exploit.

In the next section we will present the requirements and objectives that have driven the design of the WS-DAIOnt-RDF(S) ontology access mechanism and its implementation in the RDF(S) Grid Access Bridge system.

V. ACCESSING RDF(S) ONTOLOGIES IN OGSA:

WS-DAIOnt-RDF(S)

In this section we present WS-DAIOnt-RDF(S), a WS-DAI-based RDF(S) ontology access mechanism for OGSA, which is being developed in the OntoGrid project.

Before describing the interface specification, we start with a brief introduction of the foundational issues underpinning the design of the access mechanism. We conclude this section by presenting two sample use cases that show how the identified interfaces would be used.

A. WS-DAIOnt-RDF(S) Foundations

RDF(S) is the union of RDF [12], [13] and RDF Schema [14]. The former is a formal data model from the W3C for describing web resources with machine readable data. This model is used for describing and interchanging metadata. The latter is a semantic extension of RDF that permits defining vocabularies in RDF.

Therefore, dealing with RDF(S) involves managing only data expressed according to the concrete syntax of the two languages. Thus, RDF(S) access services are particular types of data access services, that is, services that hold RDF(S) data and provide mechanisms for creating, retrieving, updating, and deleting this data. Nevertheless, the aim of the specification is not limited to the syntax of the RDF(S) language.

The objective of WS-DAIOnt-RDF(S) is to define an ontology access mechanism that provides access to RDF(S) sources

3This techniques may be defined with simple application APIs or with more elaborated domain-specific standards, such as SQL in the context of relational databases, XQuery in the case of XML documents, etc.

4Groundings of the WS-DAI specification.
at the conceptual level. Therefore, it does not only exploit the syntax, but the semantics of the RDF(S) model. To achieve this, we will use a set of data resources that mimic as much as possible the components defined in the RDF(S) model.

The data resources used are divided in two groups: the first provides data resources that represent aggregations of RDF(S) resources, whereas the second defines data resources that represent the concrete contents of particular RDF(S) resources. Fig. 1 shows how all of these data resources are related.

The first group defines the following data resources: repositories, which are sets of RDF(S) modelling components defined in the same namespace\(^6\); and selectors, which resemble RDF(S) storage systems and contain repositories. The second set of data resources maps the RDF(S) data model accurately, that is, each of the data resources represents an RDF(S) modelling component\(^7\).

**B. WS-DAIoNT-RDF(S) Interface Specification**

The interfaces are grouped in levels according to the granularity of the access operations they provide for interacting with an RDF(S) source. Each level is linked to the next through special factory interfaces that permit creating data services, that in their turn, implement access interfaces of lower levels. Fig. 2 illustrates how the WS-DAIoNT-RDF(S) interfaces have been combined in the RDF(S) Grid Access Bridge, which will be presented in the next section.

The first level provides the interfaces that allow managing the ontologies as a whole, working at the repository selector level:

- **RepositorySelectorAccess**: this interface provides functionalities for managing multiple repositories deployed in a virtual organization. It provides the means for: putting new repositories under its control; removing existing repositories from its control; and retrieving the whole contents of existing repositories, serialized in RDF/XML.
- **RepositorySelectorFactory**: this interface provides the means for pointing the user at the concrete data service that provides specialized access to a repository under the control of the repository selector. The specialized access is defined by the RepositoryAccess interface.

The second level provides interfaces which allow working with the contents of the ontology at a repository level:

- **RepositoryAccess**: this interface provides functionalities for creating, retrieving, updating and deleting components of the repository. The components managed are resources, classes, properties, statements, lists and containers. Additionally, querying functionalities are also provided to interact easily with a repository using traditional RDF(S) query languages, such as RQL, RDQL, etc. It is worth mentioning that the specification does not impose any restriction about the supported query languages and that the implementation is responsible for defining which query languages it supports.
- **RepositoryFactory**: this interface provides the means for pointing the user at the concrete data services that supply specialized access to a specific component of the repository, taking into account the nature of the component. The specialized access to each kind of component is defined by the interfaces described in the following level.

The third level provides interfaces that allow operating directly over the contents of an ontology according to the specificities of each RDF(S) modelling component.

- **ResourceAccess**: this interface provides the basic functionalities required for dealing with a resource in a generic way, and these are: defining the values of the built-in RDF(S) properties for an RDF resource (label, type, etc.), and retrieving and also deleting these values. The interface also provides functionalities for checking which properties can be defined (create a triple with subject the resource) according to the parents of the resource; which properties are already defined and which are their values.
- **ClassAccess**: this interface provides the specific functionalities needed for interacting with an RDF resource that happens to be a class. In this case, functionalities for managing the class hierarchy (retrieving and modifying the subclasses and superclasses) are also provided. Finally, functionalities for managing the resources that are instances of the class (retrieving, creating and removing) are also provided.

The interface also puts the means for checking the properties defined for the class (which have as domain the class).

- **PropertyAccess**: this interface offers the specific functionalities needed for interacting with an RDF resource that happens to be a property. These functionalities allow managing the domain and range of a given property and also its situation in a property hierarchy (which are the subproperties and the superproperties).

The interface also defines methods for managing the values of this property for a given resource, that is, creating, retrieving, and deleting the values.

- **StatementAccess**: this interface provides the specific func-

\(^6\)We can think of an ontology as a repository, because the components of the ontologies are usually defined in the same namespace.

\(^7\)Classes, properties, containers, lists, statements, etc.
tionalities required for interacting with an RDF resource that happens to be a statement (not to be confused with a triple). The interface permits managing the subject, predicate, and the object of a given statement.

- **ListAccess**: this interface provides the specific functionalities required for interacting with an RDF resource that happens to be a list, in fact, a specific section of a list. The basic functionalities provided by the interface are used for composing a list (creating first and rest nodes, and arranging them), and for checking if a resource “belongs” to the list.

- **ContainerAccess**: this interface provides the specific functionalities needed for interacting with an RDF resource that happens to be a container, basically for managing container members.

- **ContainerFactory**: this interface provides some extra functionalities for interacting with a container. Other functionalities are provided for pointing the user at other data services that supply specific ways of accessing the container (they implement the ContainerIteratorAccess or the AltAccess interfaces described below).

- **ContainerIteratorAccess**: this interface provides some extra functionalities for interacting with the container following the iterator pattern, that is, they are a method for retrieving next element in a sequence made up of all of the members of the container.

- **AltAccess**: this interface provides the specific functionalities required for interacting with an RDF container that happens to be an alt container. As there is a subtle difference between the intended semantics of this type of built-in container and the other two (seq and bag), we include specific functionalities for dealing with these particularities, which are the differentiation between default and alternative values.

Because of the nature of RDF(S) and the lack of cardinalities in its model, all the methods defined by the interfaces support multiple values for the same properties. Some of the interfaces even allow using sets of these values. Please refer to [15] for the full specification.

The interface specification has been presented to the DAIS WG of the OGF, and this specification is the basis of the **DAIS RDF(S) Access Initiative** for developing an standard RDF(S) access mechanism for OGSA, see [16]. Concretely, this specification is the reference used for developing the WS-DAI-RDF(S) Ontology specification [17].

C. Sample Use Cases

Before the implementation, we show how these interfaces are used. For this purpose, we use the RepositorySelector, Repository and Class data services showed in Fig. 2.

The first use case is the retrieval of the classes defined in a repository. This process is shown in Fig. 3.

The first step consists in loading the required repository using the RepositorySelector service (1). When receiving the GetRepositoryFactoryRequest, the data service creates a Repository data resource (2) and links it to the associated RDF(S) source (3). The data service returns the user an
Fig. 3. Guessing the classes defined in a repository

endpoint reference to the Repository data service through which we access the repository (4). This endpoint reference also includes the identity of the Repository resource.

The second step consists in asking directly the Repository service which are the classes that it contains (5). The Repository service uses the specified Repository resource to access the RDF(S) source and retrieves the classes defined in the source (6). Finally, the service returns the classes found (7).

The second use case consists in traversing a class taxonomy. Fig. 4 shows the steps of this use case.

Once we have loaded the desired repository (same steps as above), we have to load the class. We proceed in the same way as we did for the repository, but in this case using the Repository service. When the service receives the GetResourceFactoryRequest (5), it creates the required data resource (6) and links it to the associated resource in the repository (7). Then, the service returns an endpoint reference to the Class data service we will use for interacting with the class (8).

Using the endpoint reference, we then request the Class data service the classes related to the specified class (9). Then, the service uses the associated class data resource to find out which are the classes related to the class (10). Finally, the data service returns the related classes (11).

If we repeat steps (9) to (11), we are able to discover the parents and children of the class. If we then repeat the described process using one of the previously obtained classes, we are able to traverse the class taxonomy up and down.

VI. RDF(S) Grid Access Bridge, THE WS-DAI Ont-RDF(S) IMPLEMENTATION

The WS-DAI Ont-RDF(S) specification is being implemented in the RDF(S) Grid Access Bridge system (a.k.a. RGAB), and developed in the OntoGrid project.

In addition to supporting the WS-DAI Ont-RDF(S) specification, the objective of the RGAB system is to provide a grid-based ontology access mechanism for existing RDF(S) sources\(^8\), but not to provide a grid-based RDF(S) storage system. Therefore, the system is to act as a bridge between the user and the existing RDF(S) sources.

RGAB follows a two-tier architecture: the service tier and the storage tier. In the service tier the high level processes are implemented as services, and these services are integrated in an specific host grid infrastructure. On the other hand, the storage tier is in charge of directly interacting with the specific RDF(S) sources, hiding these to the service tier and providing an homogenous set on interfaces for their management.

This architecture enables a loosely coupled interaction between the services and the RDF(S) sources, and permits creating a plug-in architecture with plug-ins for supporting specific RDF(S) storage systems. This overall design enables to develop an storage-independent implementation, which allows using multiple backends simultaneously.

The grid infrastructure selected for developing WS-DAI Ont-RDF(S) is the Globus Toolkit 4, specifically the Java WS Core. With regard to the RDF(S) storage, we are currently developing plug-ins for Sesame and Jena, and we plan to support further RDF(S) storage systems, such as OracleRDF.

Fig. 5 illustrates the how the implementation is organized and the relationships with the interface specification. The links between sets of interfaces and concrete services mean that the same service implements all of the associated interfaces.

VII. EXPERIMENTATION

In order to validate the approach, we plan to perform several types of functional and non-functional testing, to evaluate the performance of the system, and to be able to determine if the operational model defined by the WS-DAI Ont-RDF(S) interfaces is usable or not. To do so, we will carry out different types of experiments, using a test suite of artificial RDF(S) sources.

The test suite contains RDF(S) sources which vary in (1) the type of elements defined inside, (2) the relationships defined

\(^8\)These sources can range from serialized RDF(S) files, to storage systems that provide RDF(S) storage capabilities, going through any other legacy system that exposes its data as RDF(S)
Using this test suite, we will carry out the following experiments:

- **Stability testing**: the objective is to evaluate whether the system is stable. To do so, we have to analyze how the size and structure of the repositories affect the performance of the system. This analysis will be done through a series of tests which exercise each functionality using different test repositories each time. We then measure the execution time of each functionality and determine the correlation degree between the execution time and the size and structure of the repositories.

- **Stress testing**: the objective is to find out which are the limits of the system. We determine this through specific tests that continuously access the repositories that vary in size and structure and then we measure the latency between successive accesses.

Other experiments used to evaluate the performance are the following:

- **Load testing**: the objective is to evaluate the behaviour of the system by simulating multiple users accessing the services concurrently.

- **Volume testing**: the objective is to check that the repositories can grow during the test time, and that they can be managed by the system without stopping it working or degrading its operation in any way. To check this, we will create specific tests that continuously add data to the repositories, and measure the latency between successive accesses.

A complete evaluation and testing plan for the RDF(S) Grid Access Bridge system, which also covers the software verification and validation of the system, is included in [18].

**VIII. CONCLUSIONS AND FURTHER WORK**

The current OGSA specification does not provide the means for using ontologies within the grid framework it defines. In order to enhance the current grid with semantic technologies, it is crucial to provide OGSA with an ontology usage infrastructure.

By making ontologies available like other resources in the OGSA framework, we leverage on the OGSA capabilities for other aspects of ontology management that are of major interest, such as security (authentication, authorization and accounting), replication, deployment, configuration and management that are provided “for free” by OGSA. These aspects are not yet taken into consideration in the Semantic Web community, but are fundamental if this type of technologies are to be used outside the academia.

The most sensible way of creating the ontology access mechanism is through the data access and infrastructure available in the OGSA framework. Thus, creating ontology access mechanisms as WS-DAI realizations would ensure the integration of the ontologies in the OGSA framework like...
other specific data resources, which would not be logically different from other data resources, but which, at low-level, would provide a specific access mechanism for them.

In this paper we roughly describe and OGSA compliant RDF(S) ontology access mechanism built on top of WS-DAI, namely WS-DAIOnt-RDF(S). The specification focuses on providing an access mechanism driven both by the model of RDF(S) and by the most common patterns used when working with concrete RDF(S) ontologies. Therefore, it is not tied to any technical commitments because of the existing RDF(S) storage systems.

Following this loosely coupled approach, the implementation presented also focuses on totally decoupling the system from the specific RDF(S) storages that can be managed, which enables the integration of heterogeneous RDF(S) storages within the same framework.

The work presented in this paper is still under development both in the context of the OntoGrid project and in the wider Grid community through the DAIS WG of the OGF.

ACKNOWLEDGEMENT

This work is supported by the EU FP6 OntoGrid project (FP6-511513) funded under the Grid-based Systems for solving complex problem. We would like to thank all the OntoGrid members for their help and feedback during the development of WS-DAIOnt-RDF(S), and the DAIS WG, specially Dr. Norman Paton, who has provided valuable feedback about the integration of the ontology access mechanism with the WS-DAI initiative.

REFERENCES