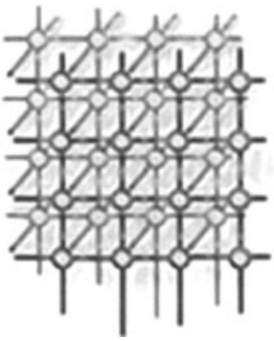
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **21**:1029–1051
Published online 11 February 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1409



Accessing RDF(S) data resources in service-based Grid infrastructures‡

Miguel Esteban Gutiérrez[1], Isao Kojima[2,*,†], Said Mirza Pahlevi[2], Óscar Corcho[1] and Asunción Gómez-Pérez[1]

[1]*Ontology Engineering Group, Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Boadilla del Monte, 28660, Madrid, Spain*
[2]*Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology Tsukuba, 305-8568, Japan*

SUMMARY

We describe the results of the RDF(S) activity within the Open Grid Forum (http://www.ogf.org) (OGF) Database Access and Integration Services (DAIS) Working Group (http://forge.gridforum.org/projects/dais-wg) whose objective is to develop standard service-based grid access mechanisms for data expressed in RDF and RDF Schema. We produce two specifications, focused on the provision of SPARQL querying capabilities for accessing RDF data and a set of RDF Schema ontology handling primitives for creating, retrieving, updating, and deleting RDF data. In this paper we present a set of use cases that justify this work and an overview of these specifications, which will enter in editorial process at OGF25. We conclude by outlining the future work that will be made in the context of this standardization process. Copyright © 2009 John Wiley & Sons, Ltd.

Received 15 March 2008; Revised 4 September 2008; Accepted 1 November 2008

KEY WORDS: database access and integration; OGSA; semantic web; RDF

*Correspondence to: Isao Kojima, Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology Tsukuba, 305-8568, Japan.
†E-mail: kojima@ni.aist.go.jp
‡Note: All URLs in this paper are as of 07/22/2008.

Contract/grant sponsor: AIST-SOA
Contract/grant sponsor: The Grant-in-Aid Scientific Research(A); contract/grant number: KAKENHI #20240010
Contract/grant sponsor: EU FP6 OntoGrid; contract/grant number: FP6-511513
Contract/grant sponsor: Marie Curie Reintegration Grant WS-DAIOnt-OWL: OWL Ontology Access in Grid Systems; contract/grant number: FP6-2004-Mobility-11, ref #046415

1. INTRODUCTION

1.1. Motivation and background

The next generation of grid technologies need to be able to virtualise the notion of distributed computation, storage, and communication over unlimited resources [1]. One of the major challenges to overcome is that related to the openness of the grid, that is, the fact that any grid node may provide, at any point in time, new services, functions, or, in general, new resources that are unknown *a priori* to its clients or other grid nodes. In order to incorporate these new elements into other applications or middleware, or to cooperate with them, not only do they have to be made available and accessible in a standardized way, but also visible [2]. Metadata usually plays an important role in this equation.

One usual means to represent this type of metadata is through the use of vocabularies that are defined, agreed, and shared by a community; hence, ensuring some degree of interoperability across the applications and/or middleware that exploits this metadata. Examples of resource description vocabularies are GLUE [3], the forthcoming Network Mark-up Language (NML) [4], the OGSA Reference Model [5], the DMTF Common Information Model (CIM) [6], etc. Through the use of these vocabularies, communities aim at tackling challenges such as resource discovery and selection (aka matchmaking), brokering, monitoring, accounting, etc.

These vocabularies have been traditionally expressed with XML Schema [7–9], which defines both the structure of resource descriptions and the set of datatypes needed for such structure. Hence, actual resource descriptions are XML documents that follow the corresponding schemata. This approach is enough in closed environments where the types of resources or the information that can be described are known *a priori*. However, in open environments where new elements can be incorporated dynamically this approach is too rigid: in general it proves to be difficult to extend existing vocabularies without changing the corresponding schemata and without having a negative effect over the systems that are able to consume and produce XML documents according to such schemata. For instance, imagine that in the CIM Model we need to refer to new *Services* different from the predefined *TimeService* (e.g. an authentication service). This modification would require extending the CIM schemata with a new XML element *CIM_AuthenticationService* together with the associated complex types required for defining such element according to CIM's schemata development guidelines (i.e. the base complex type *CIM_AuthenticationService_ Type*).

Besides, in many cases vocabularies contain taxonomies of terms (e.g. in GLUE there are several types of policies—*ManagementPolicy*, *AccessPolicy*, and *MappingPolicy*—and all of them inherit characteristics of the *Policy* element). However, in the actual XML renderings of these vocabularies, these taxonomies disappear and consequently systems using those descriptions are in charge of manually performing the necessary inferences when processing the information contained in the XML documents. For instance, if we need to retrieve information about policies in GLUE, our system needs to look for three different types of elements in the XML document—those related to management policies, access policies, and mapping policies—and this has to be hardcoded. Furthermore, if an element in a vocabulary needs to be extended (e.g. adding a new attribute to CIM *Service*s), in current XML Schema representations all its descendants have to be modified to include this new attribute.

The Resource Description Framework (RDF) is a set of recommendations of the World Wide Web Consortium for representing metadata that includes two main representation languages: RDF [10]

DOI: 10.1002/cpe

and RDF Schema [11], whose combination is usually known as RDF(S). This framework allows overcoming the aforementioned problems, namely the lack of flexibility and extensibility of resource description vocabularies and the lack of support for the representation and use of taxonomies within the vocabularies. In fact, there are already several ongoing efforts in the Grid community to embrace this framework (Grid Ontology[§] [12], and of Open Grid Forum (OGF) groups that are considering RDF(S): GLUE, NML, etc.).

On to other matters, RDF(S) is also being used to represent large amounts of data in a number of applications worldwide. For example, the UniProt Protein Database contains 262 million RDF triples[¶], the IngentaConnect bibliographic metadata storage contains over 200 million RDF triples[‖], and the CombeChem application manages more than 80 million RDF triples in multiple databases[**].

For all these reasons, there is a need to work on the provision of a standard, scalable, and robust access mechanism for distributed RDF(S) data resources. The key to realizing this goal is to provide a standard RDF(S) data access mechanism based on grid computing technologies, which is one of the main goals of our activity within the OGF DAIS working group.

1.2. Activity history and approach

The DAIS RDF(S) activity has its origins in several presentations made at the 3rd GGF Semantic Grid Workshop and the DAIS for RDF birds of a feather session held at GGF16, in February 2006 [13–15]. Most of these presentations showed that RDF(S) was commonly used in middleware and application development.

As a result, the Database Access and Integration Services Working Group (DAIS-WG) chartered an activity to develop an RDF(S) access specification as part of its data access specifications portfolio, composed at that time by the WS-DAI core specification [16], the WS-DAIR realization[††] for relational data access [17], and the WS-DAIX realization for XML data access [18]. The objective of this activity was to define a mechanism, WS-DAI-RDF(S), that provides a set of standard access interfaces to RDF(S) data resources, compliant with the principles and practices defined by the *Open Grid Services Architecture* (OGSA) [5], and with the guidelines of the data access and integration facilities defined by the WS-DAI core specification. This activity is part of the more general activity to develop OGSA's data architecture [19]. Figure 1 depicts the current DAIS WG Specification portfolio and shows how the new WS-DAI-RDF(S) realization fits within it.

The WS-DAI-RDF(S) realization distinguishes two types of access to RDF(S) data resources: programmatic and declarative. The former defines a set of fine-grained operations for accessing RDF(S) data resources exploiting the semantics of the RDF Schema model and the latter relies on

[§] In fact, the Grid Ontology described in [12] is expressed in a more expressive language, OWL, which is built on top of RDF(S), but this is out of the scope of this paper.
[¶] UniProt Protein Database (http://dev.isb-sib.ch/projects/uniprot-rdf/).
[‖] IngentaConnect (http://www.ingentaconnect.com).
[**] CombeChem (http://www.combechem.org).
[††] A realization is an extension of the WS-DAI core specification, aimed at providing an specific access mechanism for a particular type of data resource.

DOI: 10.1002/cpe

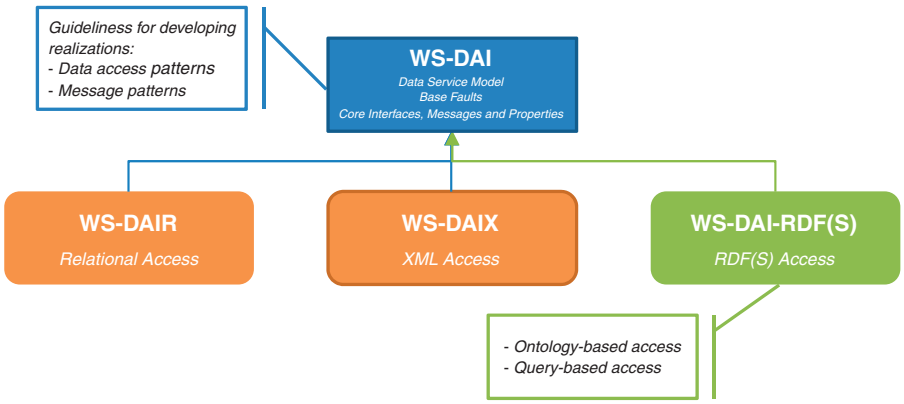


Figure 1. The current data access and integration services portfolio and the new RDF(S) realization.

the usage of the SPARQL RDF query language [20] for expressing declaratively the data that is to be retrieved.

As a result of the work performed in this activity, there are currently one informational document that includes background information and motivational scenarios, one glossary of terms document, and two specification documents, one for each of the types of access.

The remainder of the paper is organized as follows. Section 2 introduces a set of motivational scenarios which further justify the need for RDF(S) access mechanisms. Section 3 describes the internals of the WS-DAI-RDF(S) realization, covering both the declarative and programmatic specifications. Section 4 outlines our current work plan to conclude the development of the realization, and Section 5 draws conclusions.

2. MOTIVATIONAL SCENARIOS

In this section we present several scenarios that demonstrate the need and usefulness of RDF(S) to describe data and resource metadata and of the RDF(S) data access methods developed by this activity to manage the access to it. Other use cases that show the usefulness of RDF(S) data access protocols in different types of applications can be found in [21]. In this section, we have focused on grid-specific use cases.

The first scenario shows that RDF(S) can be used to enable resource matchmaking in a virtual organization, where RDF(S) is used to describe the resources offered by each organization, and how RDF(S) access methods (either programmatic or declarative) enable this task.

The second scenario shows how the resources in a virtual organization, such as the one formed in the aforementioned matchmaking scenario, can be monitored and annotated in order to maintain up-to-date metadata about them, so that future matchmaking tasks can be performed accurately.

Finally, the third scenario shows the importance of using a standard access method in a large scale distributed RDF database.

DOI: 10.1002/cpe

2.1. Grid resource matchmaking in virtual organizations

Motivation. A grid may include a large number of resources with various intrinsic capabilities distributed across different organizations. The explicit representation of resource metadata, with its adequate exploitation, plays an important role in facilitating effective grid resource discovery and selection, as shown in [22–28]. This is a key aspect considered in semantic grid information system architectures and middlewares such as S-MDS [14], S-OGSA [29], GRIMOIRES [30], S-SRB [31], CaBIG data access services [32], etc.

Goal. Given the repositories and services that store metadata from different types of resources, the goal of a matchmaker is to discover and select appropriate resources for a given task. This can be done by querying the available metadata –either using a high-level RDF query language such as SPARQL or using a specialized data access API– and ordering the matched resources based on specific ordering criteria, i.e. class subsumption relationships.

Requirement analysis. Each semantic grid information system may collect resource information from different sources of the grid, and maintain the resource metadata using their own proprietary mechanisms. Despite the differences, the metadata representation used by these systems is the same, that is, it is based on the RDF(S) model. Besides, the metadata could be created using the same RDF schema. In this scenario, it is also desirable to retrieve resource metadata from multiple available systems, so that the final user may obtain more complete information about resources, as the lack of information in one system might be overcome with the information of others.

Use case. Figure 2 shows the aforementioned matchmaking scenario implemented using the SPARQL query language. In this scenario, RDF(S) data sources are exposed through RDF(S) data access services, which support the WS-DAI-RDF(S) query-based access mechanism. A requester sends a resource request to the matchmaker, specifying the resource requirements as a SPARQL query (1). The matchmaker forwards the query to existing metadata information systems, which also support the same querying capabilities (2, 4, 6, 8). After receiving the query results (3, 5, 7, 9), the matchmaker merges the results and forwards them to the consumer (10). Similar work has been proposed and implemented in a semantic web environment [33].

2.2. Grid resource annotation and monitoring

Motivation. As previously mentioned, a grid can host a large number of resources with heterogeneous characteristics and capabilities distributed across different organizations, hosting various semantic grid applications (i.e. [34]), and architectures (see [29,35]) aimed at facilitating the discovery and selection of the resources available by using metadata of these resources. Providing the means for maintaining valid and up-to-date metadata is fundamental for carrying out accurate resource matchmaking in this scenario.

Goal. Given a set of agents that monitor available resources (which provide monitoring capabilities, i.e. INFOD [36] implemented for notifying changes in resource status) in a virtual organization, checking their characteristics, capabilities, and status; and given a set of repositories and services that store metadata and the vocabularies that provide the semantics for these metadata; the goal is to provide the means for creating the metadata using an adequate monitoring vocabulary, and for maintaining the metadata stored in the repositories.

DOI: 10.1002/cpe

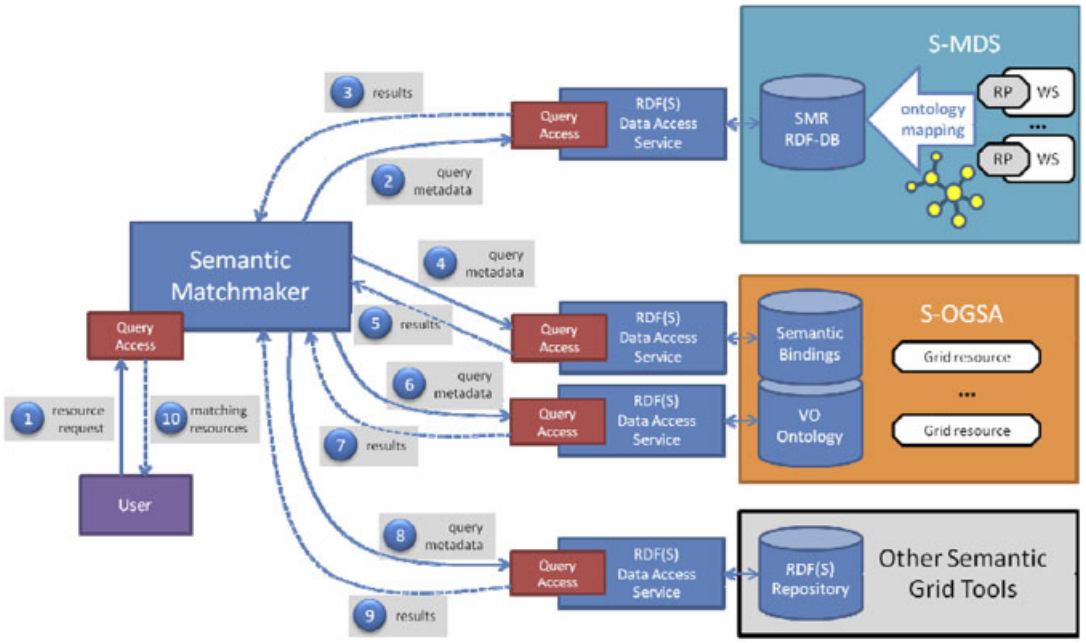


Figure 2. Grid resource matchmaking using WS-DAI-RDF(S) data access mechanisms.

Requirement analysis. The maintenance of the metadata implies browsing, updating, and deleting existing metadata already stored in the repositories. Therefore, it is necessary to have the means for both reading and writing the metadata. Furthermore, as both the annotation process and the metadata managed might be very complex and long, deleting and generating all the metadata about a resource every time a change happens may not be feasible. Thus, providing fine-grained operations for operating over the metadata is worthwhile.

Use case. Figure 3 schematically depicts this monitoring and annotation scenario. RDF(S) data access services provide a standard access method for the RDF(S) data resources (metadata repositories and vocabulary repositories). Monitoring agents connect to the resources' monitoring facilities (1). When a change in the resource is detected and notified to the agent (2), it browses the vocabulary repositories to check which elements are affected by the specific change (elements that are obsolete, elements that may be out-of-date, and new elements that may also have to be added) (3). After determining the set of changes that have to be made in the metadata repositories, the agent deletes the obsolete parts of the affected metadata (4), updates those parts that are out-of-date (5), and creates any new part that is required (6).

2.3. Distributed RDF storage for ubiquitous objects

Motivation. Ubiquitous code (ucode) [37] is a unique id in the form of 128-bit binary piece of data assigned to real-world objects for identification purposes among multiple computer systems.

DOI: 10.1002/cpe

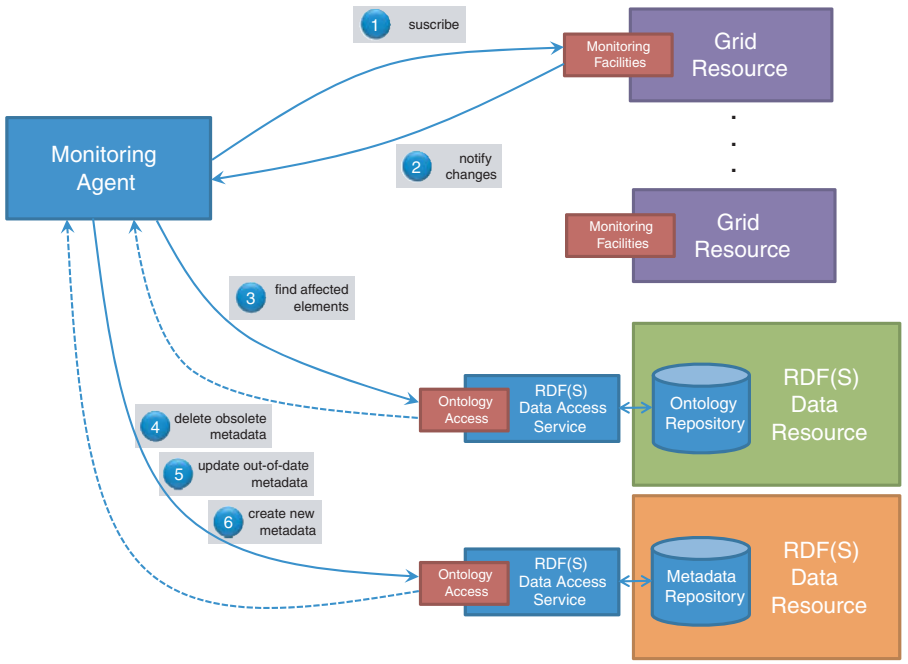


Figure 3. Grid resource monitoring and annotation using WS-DAI-RDF(S) data access mechanisms.

It is stored as a ucode tag attached to an identified object; this is often physically implemented as an RFID tag. A relation between objects (ucodes), which is called a ucode relation (UCR), is modeled as a triple, consisting of subject, predicate, and object. For example, *this apple* (subject ucode) *is produced by* (predicate relation ucode) *the JA Tsugaru-Minami Farm* (object ucode). The triples, which are usually large in number, are generally stored in a wide area distributed database (UCR database). There have been efforts to implement UCR databases using RDF databases which support SPARQL[‡‡].

Goal. Given such RDF databases, the goal is to provide a robust and scalable federated database that supports seamless access over the set of heterogeneous RDF databases.

Requirement analysis. UCR triples are stored in distributed UCR databases. Furthermore, each UCR database usually contains a large number of triples. Processing a user query usually involves accessing several databases, and the integration of the retrieved data. This may also involve large data transfers between database nodes for processing join queries. Thus, providing a query processing agent that provides a seamless and efficient access to the distributed database is crucial.

Use case. Figure 4 shows an overview of a grid-based distributed RDF database, which federates various (UCR) RDF databases. The service-based SPARQL query interfaces provide a uniform

[‡‡]Nihon Unisys SSDB (http://dev.tyzoh.jp/trac/semi-structured-db/).

DOI: 10.1002/cpe

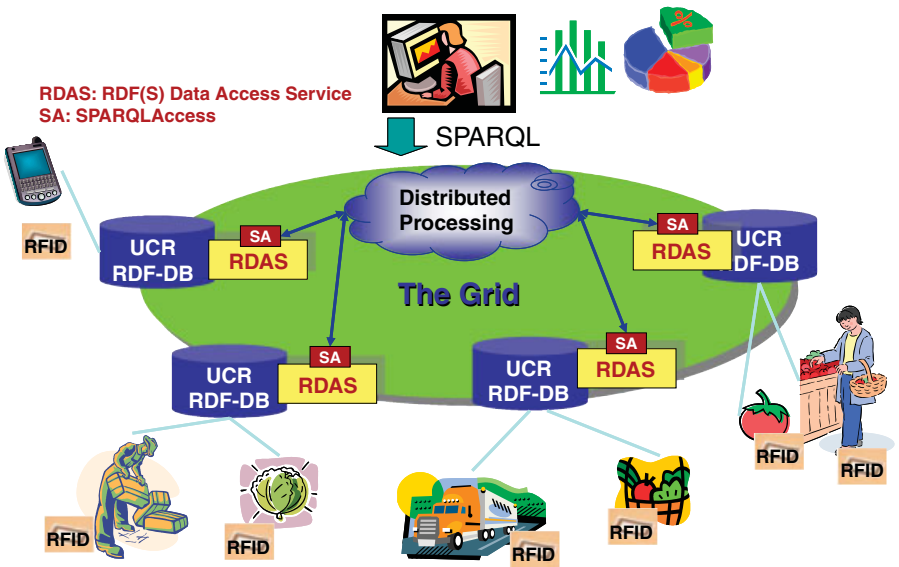


Figure 4. Large-scale distributed RDF database.

access mechanism to the heterogeneous RDF databases for distributed query processing. The indirect data access of the proposed specification (*SPARQLAccess*) is crucial to distributed query processing, as was the case in OGSA-DAI-RDF[§§]. Another attempt to distributed SPARQL query processing is described in [38].

3. THE WS-DAI RDF(S) REALIZATION

The WS-DAI-RDF(S) realization is aimed at providing specialized data access mechanisms for RDF(S) data resources. In this context, an RDF(S) data resource is a data source or sink that is based on the RDF data model, together with its associated management infrastructure that may exhibit RDF(S) model-based views.

As we presented before, this new realization distinguishes two types of access to RDF(S) data resources: declarative and programmatic. The former relies on the usage of the SPARQL RDF query language for expressing declaratively the data that are to be retrieved, while the latter defines a set of fine-grained operations for accessing RDF(S) data resources exploiting the semantics of the RDF Schema model.

These approaches have been implemented in two different specifications (WS-DAI-RDFS Querying [39] and WS-DAI-RDF(S) Ontology [40]). Each specification addresses a single approach, and

[§§] OGSA-DAI-RDF (http://wiki.dbgrid.org/index.php?OGSA-DAI-RDF).

DOI: 10.1002/cpe

provides the set of interfaces, operations, and properties required for dealing with RDF(S) data resources according to the approach followed. The remainder of the section presents these specifications in detail.

3.1. Accessing RDF(S) using a query-based approach

3.1.1. Foundations

One challenge of data management in a distributed environment is that RDF data are created in a bottom-up manner. That is, each application creates its own RDF data and stores the data in their own storage space, resulting in RDF data being stored in a distributed environment and managed locally, as was shown in the example in Section 2.3. In such an environment it is important to facilitate efficient data integration by providing common interfaces.

The objective of the querying specification here is to provide a set-oriented declarative access method to get the data set in which the user is interested. Our specification does not specify its own access language for RDF(S) data resources. Instead, it acts as a channel for RDF queries and updates languages to be conveyed to the appropriate data resources, for instance RDF(S) data resources or data resources that support RDF-type queries. The query language supported is the W3C recommended SPARQL [20].

In addition to the SPARQL query language, the W3C has recommended the following related standard specifications to access remote/distributed RDF data using SPARQL:

- SPARQL Query Results XML Format [41] is an XML format for the variable binding and boolean result format provided by the SPARQL query language.
- SPARQL Protocol for RDF [42] is a protocol for conveying SPARQL queries from query clients to SPARQL query processors.

The approach taken here in the WS-DAI-RDF(S) Querying specification is to keep as much compatibility with the existing W3C standards while satisfying the WS-DAI core specification. As described later, the WS-DAI-RDF(S) Querying specification fully supports the SPARQL query language and its associated result format [41]. In addition, the core WS-DAI provides useful grid-specific functionalities such as indirect access. These functionalities are not supported by any W3C standard and supporting them as a minimum extension to the existing W3C standards is very important.

3.1.2. Interfaces

3.1.2.1. Direct access interfaces. Direct access to a data access service allows the results of a request to be delivered to the consumer directly in the response message. This is one of two data access modes, provided by the WS-DAI core model. To cater for this mode of operation the following interface is defined for accessing an RDF(S) data resource using SPARQL:

- SPARQLAccess allows the evaluation of SPARQL queries across a collection of RDF graphs. This interface supports SPARQL requests to be made to an RDF(S) data resource.

In the example shown in Figure 5, a consumer uses the SPARQLExecute message to submit a RequestDocument in a format defined in [42]. The associated SPARQLQueryExecuteResponse

DOI: 10.1002/cpe

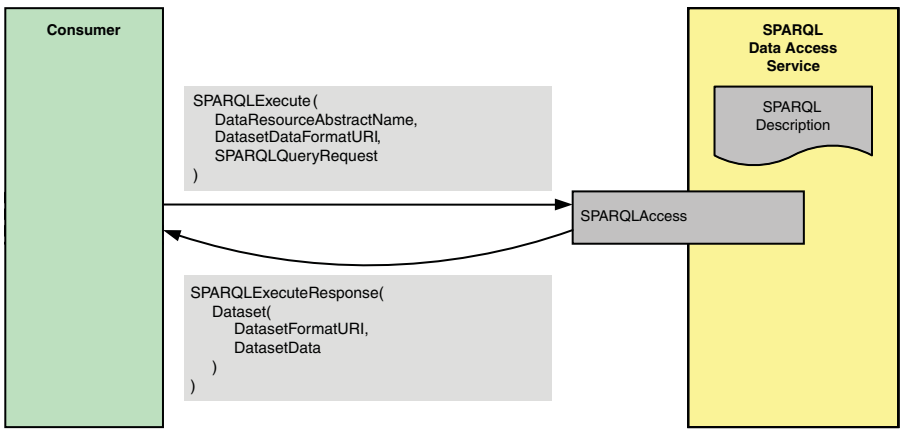


Figure 5. Direct access using the SPARQLAccess interface.

message will contain a set of query result items. Message patterns and XML data structure of the interface are defined based on W3C standards described above.

It is worth mentioning that SPARQL provides only retrieval functionality without update capabilities. In our querying specification, we do not define any specific update language and leave this problem open to future discussion in W3C. However, our WS-DAI-RDF(S) querying framework could be extended to encompass any new or emerging RDF query/update standards by employing the patterns established in the WS-DAI core specification.

Another important point is that SPARQL is a data-oriented query language that queries information held in the RDF data model or RDF graph (i.e. it does not support RDF schema-based reasoning). For example, SPARQL without reasoning could not get all subclasses of a class since it does not support the transitive closure of class–subclass relationships. The semantics given by RDF Schema are not supported since the solution to the query is obtained by just matching graph patterns to subgraphs of the target RDF graphs. The lack of this important feature has forced most RDF database products to provide reasoning functions for SPARQL implemented in various ways[¶¶].

In line with this, we are currently considering the provision of a SPARQL query interface which supports reasoning. These reasoning capabilities would be provided as optional features, so that maximum compatibility with the existing SPARQL related W3C standards is maintained. Ongoing discussion on how to provide the reasoning functions is presented in Section 5.

3.1.2.2. Indirect access interfaces. Indirect access is another access pattern that the WS-DAI core model supports. This allows data, usually the result of a query, to be accessed by means of a new service-managed data resource, and thus the data are not returned directly to the consumer. SPARQL is similar to SQL and can return a huge amount of data as a result of a query. Indirect

[¶¶] Oracle Semantic Technology Center (http://www.oracle.com/technology/tech/semantic_technologies), AllegroGraph (http://agraph.franz.com/allegrograph), SPARQL Implementations (http://esw.w3.org/topic/SparqlImplementations).

DOI: 10.1002/cpe

access can thus be very useful when it is anticipated that the size of a query result will be large. It is also useful to keep a data snapshot as in the application scenario given in our motivational document [43].

Another important reason to provide indirect access is to support the distributed query processing for SPARQL. As in the ubiquitous use case in Section 2.1, the RDF data are spread over a network and processing a query over many RDF databases requires to support distributed SPARQL query processing. Similar problem is discussed as the Federated SPARQL problem[‖‖]. This problem will be the focus for future research, however, we need to have the basic access mechanisms to build a distributed query processing system across RDF databases. Using indirect access is important as it supports the creation of intermediate results during the distributed query processing process. It is also useful as the basis for supporting third-party data transfer which will be needed for performing distributed 'join' operations within a SPARQL query.

In DAIS indirect access is supported through the use of the factory pattern. To cater for this mode of operation the following interface has been defined:

- SPARQLFactory provides access to the results of a SPARQL query.

The example in Figure 6 presents an RDF(S) data service that implements the SPARQLFactory interface.

The SPARQLExecuteFactory operation is used to make the results of a query available through, potentially, a separate data access service; for example, a data access service that implements the SPARQLItemsSet interface. In this example the SPARQLItemsSet could be stored in a database or decoupled from the database, but the important distinction is that the data are no longer made available through a service implementing direct data access interfaces; hence, the service does not have to provide facilities for submitting SPARQL expressions.

To support access to the data resources resulting from the use of the factory pattern, additional interfaces and properties are defined, in particular:

- SPARQLItemsSetDescription provides properties of a set of SPARQL query result items.

ResultsSetAccess and TripleSetsAccess. SPARQL has four query forms: CONSTRUCT, DESCRIBE, SELECT, and ASK. The first and second forms return an RDF graph as query result; the former returns an RDF graph constructed by substituting variables in query patterns, while the latter returns an RDF graph that describes the resources found. The resulting RDF graph can be presented by using RDF/XML [44] or N3 (Notation 3) format.

In contrast to these two forms, the results of the other two are not RDF graphs: the third returns all, or a subset of, the variables bound in a query pattern match; the fourth returns a boolean indicating whether there is a match for a query pattern. To represent the latter query results, i.e. bindings in an XML format, the W3C provides the SPARQL Result Set XML Format specification [41].

In line with the aforementioned particularities of the query results, we define two interfaces that provide specialized access to these query results. The interfaces are to be implemented by those data access services whose EPR are returned to the user by the SPARQLExecuteFactory operations.

[‖‖]Federated SPARQL, (http://www.w3.org/2007/05/SPARQLfed/).

DOI: 10.1002/cpe

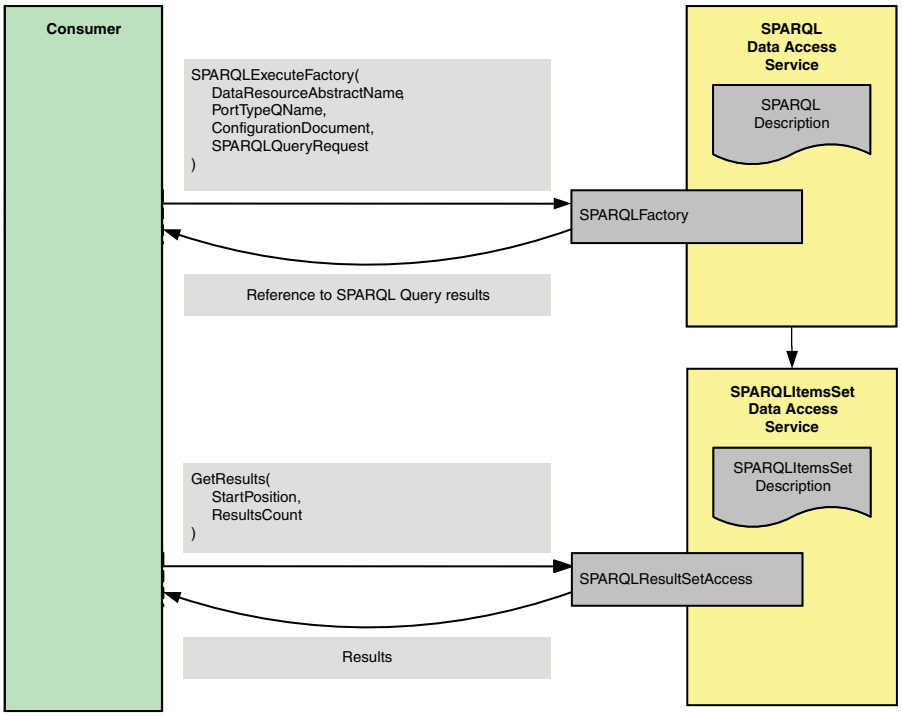


Figure 6. Indirect data access to the results of a SPARQL query.

- SPARQLResultsSetAccess provides access to a set of query results, which are the result of a SPARQL SELECT/ASK query.
- SPARQLTriplesSetAccess provides access to a set of triples, which are the result of a SPARQL CONSTRUCT/DESCRIBE query.

A consumer uses the GetResults (or GetTriples) message to retrieve a number of results from the items set. It submits a RequestData containing the StartPosition and ResultCount parameters. The associated GetResultsResponse (or GetTriplesresponse) message will contain the requested results in a serialized form.

GraphCollectionAccess. The specification extends the base interfaces and corresponding properties defined in the WS-DAI core specification to provide access to RDF(S) data resources consisting of a collection of RDF graphs. To cater for this representation, the following property and interface are defined:

- GraphCollectionDescription provides properties of an RDF(S) Collection that a data access service may represent.
- GraphCollectionAccess provides access to RDF graphs in a collection.

They provide basic access methods for the set of RDF triples, i.e. an RDF graph.

DOI: 10.1002/cpe

3.2. Accessing RDF(S) using an ontology-based approach

The aforementioned approach is aimed at accessing RDF(S) sources using the SPARQL query language. Unlike to SQL, the SPARQL query language does not provide the means for specifying RDF(S) data creation, deletion, or update. Thus, it is mandatory to provide further RDF(S) access mechanisms that allow tackling these aspects when dealing with RDF(S) sources, and complement the query specification.

Up to now, the Semantic Web community has been the most active in the development of systems for using and exploiting RDF(S), and has developed a plethora of RDF(S) triple store, third-party libraries, and tools, for working with RDF(S)***. Nevertheless, the vast majority of these systems are oriented toward the usage of RDF(S) for solving specific problems. Thus, the RDF(S) access mechanisms provided by these systems are biased according to the specific needs they had when working with RDF(S) data sources. As a result, no agreed API for accessing RDF(S) is available yet.

The objective of the WS-DAI-RDF(S) Ontology access specification is to provide an integral access mechanism for RDF(S) sources that goes beyond the retrieval capabilities offered by the querying specification, while providing a simple but complete set of functionalities that abstract the most general necessities a user may have when working with RDF(S) data sources.

To achieve this objective, the specification proposes a model-based access mechanism for accessing RDF(S) sources at the conceptual level, that is, an access mechanism that revolves around the concepts and semantics defined by the RDF(S) model. Thus, the specification details a set of ontology handling primitives for dealing with such model, hiding the syntactic aspects of RDF(S) and transparently exploiting its semantics.

In order to integrate this in the grid, the WS-DAI-RDF(S) Ontology specification defines a collection of data access interfaces for RDF(S) data resources, which extends those specified in the WS-DAI core specification [16]. These new interfaces provide a set of model-based operations for accessing RDF(S) data resources at different granularities.

3.2.1. Data resources

The WS-DAI-RDF(S) Ontology specification differentiates several types of RDF(S) data resources, each of them provided for allowing addressing and managing RDF(S) sources at different granularity levels. The diagram depicted in Figure 7 shows which are the data resources defined in the specification and the relationships existing between them using UML notation.

The data resources can be classified in two groups:

(a) *Placeholders of built-in RDF(S) classes* (*Resource*, *Class*, *Property*, *Statement*, *Container*, and *List* data resources): These data resources provide class-oriented views to an RDF(S) resource, that is, the views focus on the specific data that can be associated with a particular RDF(S) resource that is an individual of the main RDF(S) built-in classes, as defined in [45].

These data resources are organized hierarchically, having the *Resource* data resource as the more general type of resource (defining the minimum data that are common to all these

***http://esw.w3.org/topic/SemanticWebTools.

DOI: 10.1002/cpe

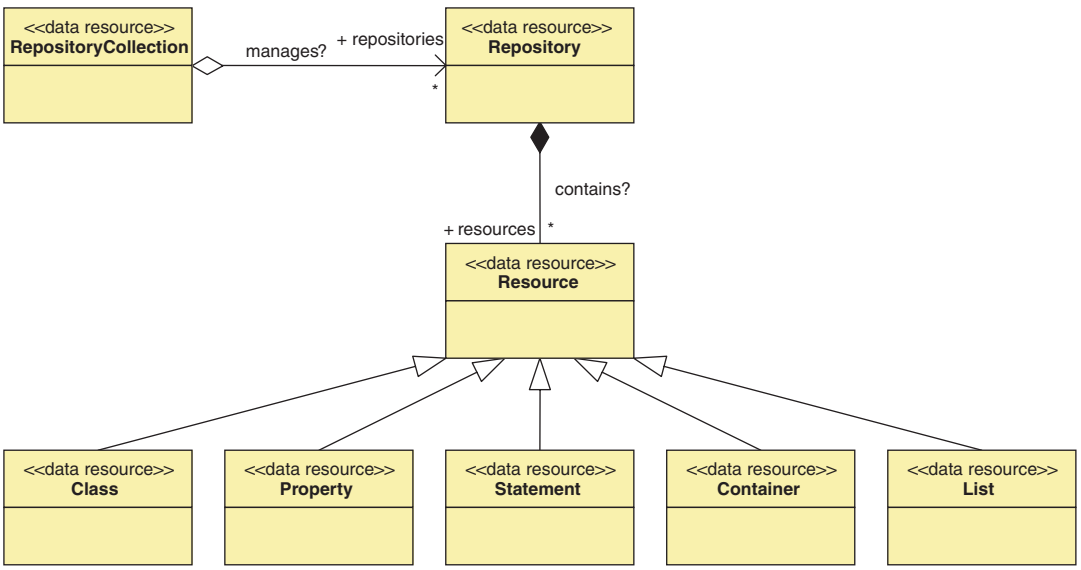


Figure 7. WS-DAI-RDF(S) Ontology Data resource model.

data resources), and the other data resources specialize this view, taking into account the specificities of the particular class they represent.

The specialized data resources are classified in two groups: *schema data resources*, which contains placeholders of elements that define the schema of the ontology (*Class* and *Property* data resources) and *additional data resources*, which are focused on other specific types of resources that can be defined in RDF(S) (reified statements, RDF collections, and containers).

(b) *Convenience abstractions* (*RepositoryCollection* and *Repository* data resources): The previously mentioned data resources provide fine grain views focused on specific parts of an RDF(S) source, in particular those associated with a given resource. But RDF(S) sources can contain more than a resource; therefore, it is necessary to raise the granularity at which RDF(S) sources are viewed and can be managed.

In this way, a *repository* data resource represents an aggregation of resources and the data associated with them[†††], where the key issue is that the *resources belong* to the *repository* (thus, *resource* data resources are derived data resources from *repository* data resources).

Similarly, a *repository collection* data resource is an aggregation of *repositories*, the repositories managed by the implementation. Nevertheless, in this case the repositories are not

[†††] Another way of understanding a repository data resource is as a container of RDF triples.

DOI: 10.1002/cpe

owned by the *repository collection* but managed, so that these repositories can be potentially managed by several implementations simultaneously.

How these resources can be used will depend on the specific application. For instance, *Repository Collection* and *Repository* data resources could be used in the first scenario described in Section 2. These data resources would hold the metadata as well as the vocabularies needed for carrying out the semantic matchmaking. Using these type of resources the application has a complete view of the data involved in the semantic matchmaking process. Nevertheless, a constrained view of the complete data set might be better, as is the case of the annotation and monitoring scenario (see Section 2.2). In this scenario we require both the complete and the partial view of the data set. The former is required for determining the changes that have to be done—using *RepositoryCollection* and *Repository* data resources, and also using *Class* and *Property* data resources. The latter is preferred for carrying out the changes, targeting the affected RDF resources –using *Resource* data resources.

3.2.2. Interfaces

In order to interact with the data resources described above, several interfaces are provided in the WS-DAI-RDF(S) Ontology specification. These interfaces are organized following two principles:

(a) *Operation granularity.* It is necessary to differentiate and separate data access capabilities and interfaces according to the granularity of the operations, and to the type of data resources they are associated with. Three interface levels are identified: *collection level interfaces* provide the means for managing RDF(S) sources as a whole, and deal with *RepositoryCollection* data resources; *repository level interfaces* contain operations for accessing to particular RDF(S) sources, by interacting with *Repository* data resources; and *resource level interfaces* deal with RDF(S) resource data resource placeholders, and provide the most specialized operations required for dealing with the specific data managed by these data resources.

Each level is linked to the next through factory interfaces that allow the creation of derived data resources, and forward further access to these data resources through other data services, which implement specialized access interfaces of lower levels.

Figure 8 depicts the interfaces defined and also presents a possible composition of them into data access services. The dashed lines represent the intended navigability from factory interfaces to access interfaces.

(b) *Operation complexity.* The interfaces are also grouped according to the complexity of the operations they provide. Thus, *primitive interfaces* include operations that provide basic straightforward data creation, retrieval, and removal for a given data resource. On the other hand, *utility interfaces* include complex operations for a given data resource, which provide added value functionalities that enhance the access capabilities for a data resource.

3.2.2.1. Direct access interfaces. Direct access to a data access service allows the results of a request to be delivered to the consumer directly in the response message. To cater for this mode of

DOI: 10.1002/cpe

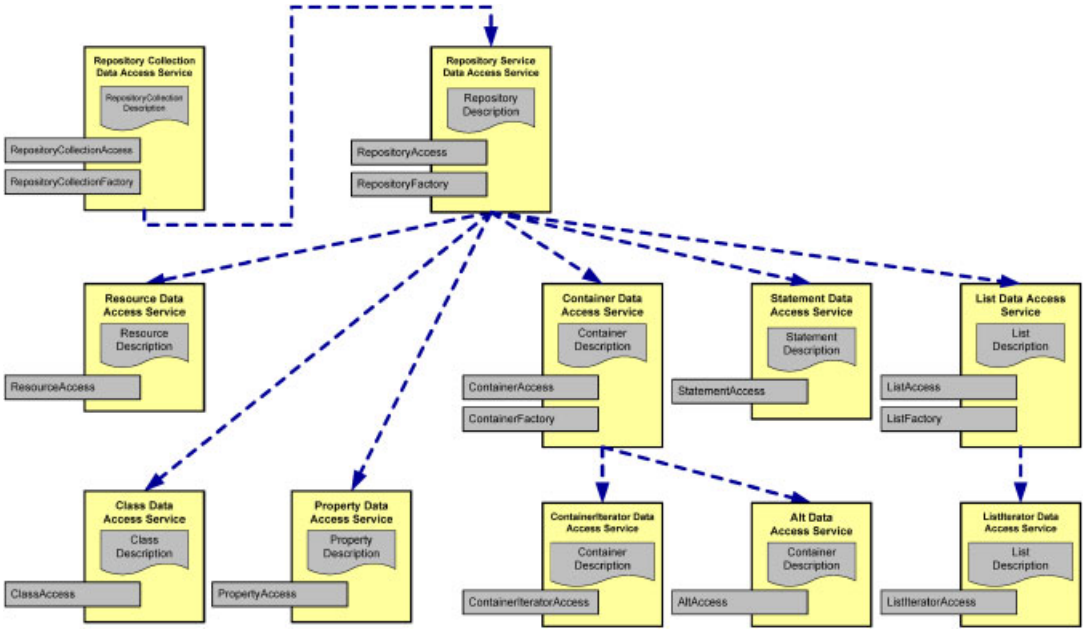


Figure 8. Data access services and interfaces.

operation the following interfaces are defined for accessing RDF(S) data resources:

- *RepositoryCollectionAccess* provides access to repositories of a collection.
- *RepositoryAccess* provides access to the inside of repositories, offering functionalities for managing the repository at RDF(S) resource level.
- *ResourceAccess* provides access to a particular RDF(S) resource, centered in those aspects common to every resource: property value management, resource description, etc.
- *ClassAccess* provides access to particular RDF(S) resources that are RDF(S) class, focusing on the data that are specific to RDF(S) classes: class hierarchy traversal, instance retrieval, etc.
- *PropertyAccess* provides access to particular RDF(S) resources that are RDF(S) properties, focusing on the data that are specific to RDF(S) properties: range and domain management, property hierarchy traversal, etc.
- *StatementAccess* provides access to particular RDF(S) resources that are RDF(S) statements—reified triples, not the triples themselves—focusing on the management of the components that set up the reification.
- *ListAccess* provides access to particular RDF(S) resources that are RDF collections (List), focusing on the management of the members of the collection, as well as, the structure of the collection.
- *ListIteratorAccess* provides access to RDF collections following the iterator pattern [46], allowing an easy retrieval of the members of the collection without requiring the identification of the position in it.

DOI: 10.1002/cpe

- *ContainerAccess* provides access to particular RDF(S) resources that are RDF(S) containers, focusing on the management of the members of the container, as well as the structure of the container, regardless of its specific type[‡‡‡].
- *ContainerIteratorAccess* provides access to RDF(S) containers following the iterator pattern [46], allowing an easy retrieval of the members of the container without requiring the identification of the position in it.
- *AltAccess* provides access to particular RDF(S) containers that are of the particular *alt* type.

3.2.2.2. Indirect access interfaces. Indirect access is supported through the use of the factory pattern. This allows data, usually a particular view of the whole data set, to be accessed by way of a new service-managed data resource, and thus the data are not returned directly to the consumer. To cater for this mode of operation the specification provides the following interfaces:

- *RepositoryCollectionFactory* provides access to repositories of a collection.
- *RepositoryFactory* provides access to the inside of repositories.
- *ListFactory* provides access to the contents of an RDF collection.
- *ContainerFactory* provides access to the contents of a container.

The usage of the factory pattern provides a basic navigation mechanism that lets the user browse RDF(S) data resources with different granularities and exploiting the particular semantics of the RDF(S) data represented by concrete RDF(S) data resources.

3.2.2.3. Description interfaces. In addition to these interfaces, multiple description interfaces are provided (*RepositoryCollectionDescription*, *RepositoryDescription*, *ResourceDescription*, *Class Description*, *PropertyDescription*, *StatementDescription*, *ListDescription*, and *ContainerDescription*), which extend the properties enumerated in the WS-DAI core specification to provide information about the relationships of the RDF(S) data services and resources with the RDF(S) data to which they provide access.

3.2.3. Profiles

The WS-DAI-RDF(S) Ontology specification is aimed at providing a means for managing RDF(S) data resources in an integral fashion, offering mechanisms for creating, retrieving, updating, and deleting contents. The specification defines these mechanisms following the RDF(S) model and semantics, providing to the user with different granularity levels for using available data resources with the required detail. To achieve this, the specification provides multiple different data resources and interfaces. As a result, the full specification is bigger than that of other realizations, such as the relational [17] or the XML one [18].

From a consumer/service provider point of view, the usefulness of the specification will depend on his requirements, and especially on the necessities he has when dealing with the RDF(S) data sources, that is, what he needs to do, and how he expects to do it. Thus, depending on his needs, he shall only use the subset of the specification that completely fulfills them.

[‡‡‡]RDF(S) defines three types of predefined containers: *seq*, *bag*, and *alt*.

DOI: 10.1002/cpe

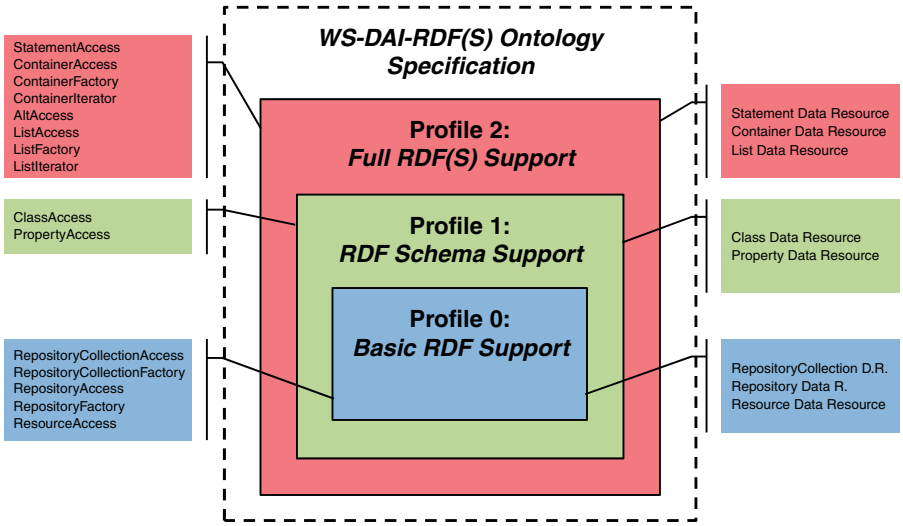


Figure 9. WS-DAI-RDF(S) Ontology profiles.

Consequently, in order to facilitate the adoption and implementation of the specification from the community, it is necessary to allow using what is needed, without enforcing the adoption of the elements of minor interest.

To achieve this we divide the specification into three different *profiles*, each one including an increasing number of functionalities that shall enable the user to deal with RDF(S) data resources with finer grain of detail, while ensuring interoperability among implementations (see the relationships between profiles in Figure 9).

Profile 0: *Basic RDF support.* This profile includes the minimum set of functionalities needed for dealing with RDF data. The functionalities of this profile are those defined in the following interfaces: RepositoryCollectionAccess, RepositoryCollectionFactory, RepositoryAccess, RepositoryFactory (limited to Resource EPR retrieval), and ResourceAccess.

Profile 1: *RDF Schema support.* This profile includes the functionalities described in the Profile 0, and extends them to those required for dealing with RDF vocabularies, as defined in the following interfaces: RepositoryFactory (augmented to retrieve Class and Property EPRs), ClassAccess, and PropertyAccess.

Profile 2: *Full RDF(S) support.* This profile includes the functionalities described in Profile 1, and extends them to those required for dealing with the rest of the built-in RDF vocabulary (containers, RDF collections and reifications). These functionalities are defined in the following interfaces: RepositoryFactory (augmented to support the rest of the data resource EPRs), StatementAccess, ListAccess, ListFactory, ListIteratorAccess, ContainerAccess, ContainerFactory, ContainerIteratorAccess, ContainerFactory.

Going back to the motivational scenarios, and taking into account the discussion about which data resources could be used in some of them, we can see that the Profile 0 is enough for the first scenario, as it provides the base means for dealing with RDF data resources. The second scenario

DOI: 10.1002/cpe

requires dealing with RDF vocabularies for determining the scope of the changes that have to be carried out. Thus, the scenario would benefit from using a Profile 1 compliant implementation, as it would provide the means for further exploiting the semantics of RDF vocabularies.

4. IMPLEMENTATIONS

OGF requires two interoperable implementations of any specification before any Grid Working Draft becomes a recommendation. Up to now both the National Institute of Advanced Industrial Science and Technology of Japan (AIST) and the Universidad Politécnica de Madrid (UPM) in Spain have provided implementations of these specifications. AIST is leading the development of OGSA-DAI-RDF [15], the preliminary work that is the basis of the WS-DAI-RDFS Query specification. The system is developed on top of OGSA-DAI[§§§] that will provide implementations of the relational and XML specifications of WS-DAI. OGSA-DAI RDF provides a multi-platform environment for different RDF(S) data resources such as Sesame[¶¶¶], Jena[‖‖‖], and Boca[****]. It includes the RDF Schema-based reasoning functionalities, and implements a subset of the ontology primitives that are defined in early WS-DAIOnt-RDF(S) specifications explained below. Although the current version of OGSA-DAI RDF does not fully comply with the current querying specification, we have released the software for public use[††††] to get user feedback, which will be of interest for the specification discussion. Figure 10 shows some screenshots of the OGSA-DAI RDF user interface.

The Ontology Engineering Group of the UPM[‡‡‡‡] keeps working in the RDF(S) Grid Access Bridge prototype developed as part of the OntoGrid project[§§§§] [47]. The system is the reference implementation of WS-DAIOnt-RDF(S), the RDF(S) ontology access specification that has been used as basis for the development of the WS-DAI RDF(S) Ontology specification [48]. Current developments are devoted to covering the full WS-DAI-RDF(S) realization.

We applied our current implementation to S-MDS [34] with Query-based matchmaking [33] as in the scenario of Section 2.1. As the feedback from this application, we learned the following lessons.

(1) Service-based, platform-independent access method is very useful to access distributed RDF databases, which are constructed by using different software products.
(2) As the current software is based on popular grid tools such as Globus and OGSA-DAI, it is very easy to interface RDF databases with existing grid middleware, such as MDS and GRAM. This implies that WS-DAI specification implementation will be also easy to interact with other OGSA-based specification implementations.
(3) There was a strong need to support OWL language also with the OWL reasoners. Providing the extensibility to support other semantic web languages will be the issue. This will be discussed in Section 5.

[§§§] OGSA-DAI Homepage (http://www.ogsadai.org.uk/).
[¶¶¶] Sesame (http://www.openrdf.org).
[‖‖‖] Jena Semantic Web Framework (http://jena.sourceforge.net).
[****] IBM Semantic Layered Research Platform (http://ibm-slrp.sourceforge.net).
[††††] AIST Database Grid Home (http://www.dbgrid.org/).
[‡‡‡‡] Ontology Engineering Group Home (http://www.oeg-upm.net).
[§§§§] OntoGrid Project Home (http://www.ontogrid.eu).

DOI: 10.1002/cpe

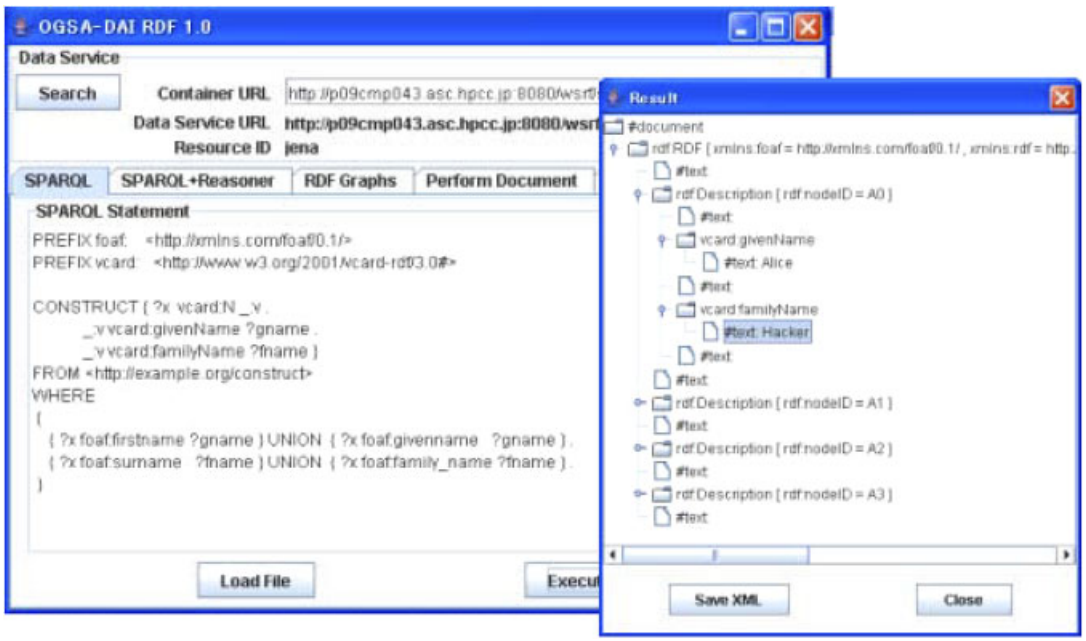


Figure 10. Screenshots of the OGSA-DAI RDF: SPARQL Query interface and the result.

5. ONGOING AND FUTURE WORKS

The WS-DAI-RDF(S) realization work is still in progress. Both the Query and Ontology specifications are still being aligned, so that they both provide a unified view of RDF(S) data resources, while providing different means for accessing to them.

In order to carry out this alignment, we are developing a glossary in which the terminology used in the specifications is defined [49]. Once the glossary is finished, the current versions of the specifications—see [39,40]—will be updated to reflect the agreed terminology. In addition, the current informational document that introduces the RDF(S) activity [43] will be updated as well, including more motivational scenarios and the terminology defined in the glossary. These documents will then all be submitted to the OGF Editors to enter the standardization pipeline.

Future works on the specification include how to support the reasoning within the querying specification. Possible extension to the current specification will be one of the following approaches currently in discussion.

(1) Define separate SPARQLQueryStatement (withReasoner) interface that supports reasoning function.
(2) Extend the current SPARQLQueryStatement interface to support reasoning option.
(3) Define the configurable description of the resource to support RDF(S) reasoning.

DOI: 10.1002/cpe

These in-discussion candidates have their own pros and cons, especially to support the future extensibility for other semantic language.

Though the work that has been carried out to date is focused on access to RDF(S), this work can be extended to other higher languages in the Semantic Web stack (such as OWL), as the definition of the specifications has been done at conceptual level, defining operations that comply with the RDF(S) model semantics and data resources that organically reflect the structure of the RDF(S) model. Thus, supporting a new language atop RDF(S) would just require focusing on the representation primitives that change the base RDF(S) semantics and data model, and creating new operations and data resources as required, and adjusting the semantics of the related operations as needed, extending the configuration mechanisms defined in the specifications.

6. CONCLUSION

In this paper we have introduced the WS-DAI-RDF(S) realization, including an overview of the Query and Ontology specifications. We have also presented several scenarios that show how these specific parts of the realization could be used. More and more interdisciplinary projects and activities across the world are creating enormous quantities of RDF(S) data, which have to be made available in a robust, scalable, and effective manner so that researchers and third-party users can exploit this increasing amount of data. The WS-DAI-RDF(S) realization, which is being developed in the OGF DAIS-WG, is aimed at developing the necessary robust scalable standard RDF(S) access mechanisms, which shall enable the integration of RDF(S) data resources in service-based grid platforms. This mechanism together with the rest of mechanisms and capabilities provided by the grid infrastructure represent the best way for solving the aforementioned issue.

ACKNOWLEDGEMENTS

This work is supported by several research and development projects. In Japan, our work has been funded by AIST-SOA project to provide a semantic SOA infrastructure [50] The Grant-in-Aid Scientific Research(A) (KAKENHI #20240010) also supported this work in part. In Spain, this work is supported by the EU FP6 OntoGrid project (FP6-511513) funded under the Grid-based Systems for solving complex problems. Part of this work has also been supported by the Marie Curie Reintegration Grant WS-DAIOnt-OWL: OWL Ontology Access in Grid Systems (FP6-2004-Mobility-11, ref #046415). We would like to thank all the participants of the DAIS-WG, especially Prof. Norman Paton, Dr. Dave Pearson, and Dr. Mario Antonioletti, who have helped in chartering this activity, and have provided valuable feedback about the integration of both specifications with the WS-DAI initiative. We would also like to thank Masahiko Kimoto, Steven Lynden, Akiyoshi Matono, and Haruyuki Kawabe for valuable discussions and comments.

REFERENCES

1. Foster I, Kesselman C. *The Grid 2*: *Blueprint for a New Computing Infrastructure*. Morgan Kauffman: Los Altos, CA, 2003; 748.
2. Jeffery K, De Roure D. Future for European Grids: GRIDs and service oriented knowledge utilities. Vision and research directions 2010 and beyond. *Next Generation GRIDS Expert Group Report*, European Commission, Brussels, January 2006.

DOI: 10.1002/cpe

3. Andreozzi S, Burke S, Ehm F, Field L, Galang G, Konya B, Litmaath M, Millar P, Navarro JP. *GLUE Specification v. 2.0*, GLUE Working Group, Open Grid Forum, 28 February 2008.
4. Grosso P, Patil A, Primet P, Ceyden A. *Network Topology Descriptions in Optical Hybrid Networks*, NML Working Group, Open Grid Forum, 10 January 2008.
5. Foster I, Berry D, Djaoui A, Grimshaw A, Horn B, Kishimoto H, Maciel F, Savva A, Siebenlist F, Subramanian R, Treadwell J, Von Reich JJ. *The Open Grid Services Architecture*, *Version 1.5*, Foster I, Kishimoto H, Savva A (eds.). OGSA-WG, Global Grid Forum, 24 July 2006; GFD.80.
6. *Common Information Model* (*CIM*) *Infrastructure Specification*. Distributed Management Task Force, Inc. (DMTF), 4 October 2005; DSP0004.
7. Biron PK, Malhotra A. *XML Schema Part 2*: *Datatypes Second Edition*, W3C XML Schema Working Group Documents, World Wide Web Consortium, 28 October 2004.
8. Fallside DC, Walmsley P. *XML Schema Part 0*: *Primer Second Edition*, W3C XML Schema Working Group Documents, World Wide Web Consortium, 2004.
9. Thompson HS, Beech D, Maloney M, Mendelsohn N. *XML Schema Part 1*: *Structures Second Edition*, W3C XML Schema Working Group Documents, World Wide Web Consortium, 28 October 2004.
10. Carroll JJ, Klyne G. Resource description framework (RDF): Concepts and abstract syntax. *RDF Core Working Group Documents*, McBride B (ed.), World Wide Web Consortium, 10 February 2004.
11. Brickley D, Guha RV. RDF vocabulary description language 1.0: RDF schema. *RDF Core Working Group Documents*, McBride B (ed.), World Wide Web Consortium, 10 February 2004.
12. Parkin M, van der Burghe S, Corcho Ó, Snelling D, Brooke J. The knowledge of the grid: A grid ontology. *The Cracow Grid Workshop 2006*, Cracow, Poland, Bubak M, Turala M, Wiatr K (eds.). 2006; 111–118.
13. Esteban Gutiérrez M, Gómez-Pérez A, Muñoz García Ó, Terrazas BV. Ontology access in grid environments with WS-DAIOnt and the RDF(S) realization. *Third GGF Semantic Grid Workshop*, Athens, Greece, 2006.
14. Said MP, Kojima I. S-MDS: A semantic information service for advanced resource discovery and monitoring in WS-resource framework. *Third GGF Semantic Grid Workshop*, Athens, Greece, 2006.
15. Kojima I. Design and implementation of OGSA-DAI-RDF. *Third GGF Semantic Grid Workshop*, Athens, Greece, 2006.
16. Antonioletti M, Atkinson M, Krause A, Malaika S, Laws S, Pearson D, Paton NW, Riccardi G. Web Services Data Access and Integration—The Core (WS-DAI) Specification, Version 1.0. *DAIS Working Group*, Open Grid Forum, 2006.
17. Antonioletti M, Collins B, Krause A, Laws S, Magowan J, Malaika S, Paton NW. Web Services Data Access and Integration—The Relational Realisation (WS-DAIR) Specification, 1.0. *DAIS Working Group*, Open Grid Forum, 2006.
18. Antonioletti M, Hastings S, Krause A, Langella S, Lynden S, Laws S, Malaika S, Paton NW. Web Services Data Access and Integration—The XML Realization (WS-DAIX) Specification, 1.0. *DAIS Working Group*, Open Grid Forum, 2006.
19. Berry D, Luniewski A, Antonioletti M. OGSA Data Architecture. *OGSA Data WG*, Open Grid Forum, 14 November 2007.
20. Prud'hommeaux E, Seaborne A. SPARQL Query Language for RDF. *RDF Data Access Working Group Documents*, W3C, 12 November 2007.
21. Clark KG. RDF Data Access Use Cases and Requirements. *RDF Data Access Working Group Documents*, Word Wide Web Consortium, 25 March 2005.
22. Tangmunarunkit H, Decker S, Kesselman C. Ontology-based resource matching in the grid—The grid meets the semantic web. *Second International Web Conference*, *ISWC2003*, Sanibel Island, FL, U.S.A., Fensel D, Sycara KP, Mylopoulos J (eds.). Springer: Berlin/Heidelberg, 2003; 706–721.
23. Brooke J, Fellows D, Garwood K, Goble C. Semantic matching of grid resource descriptions. *Second European AcrossGrids Conference*, *AxGrids 2004*, Nicosia, Cyprus, Dikaiakos MD (ed.). Springer: Berlin/Heidelberg, 2004; 240–249. DOI: 10.1007/b99982.
24. Raman R, Livny M, Solomon M. Matchmaking: Distributed resource management for high throughput computing. *Seventh IEEE International Symposium on High Performance Distributed Computing* (*HPDC7*), Chicago, IL, U.S.A. IEEE Press: 1998; 140–146.
25. Chuang L, Lingyun Y, Foster I, Angulo D. Design and evaluation of a resource selection framework for grid applications. *Eleventh IEEE Symposium on High-Performance Distributed Computing* (*HDPC-11*), Edinburgh, Scotland. IEEE Press: 2002; 63–72.
26. Raman R, Livny M, Solomon M. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing* 1999; **2**(2):129–138. ISSN: 1386-7857.
27. Chapin SJ, Katramatos D, Karpovich J, Grimshaw A. Resource management in legion. *Future Generation Computer Systems* 1999; **5**(5–6):583–594.
28. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Warren S, Tuecke S. A resource management architecture for metacomputing systems. *Fourth Workshop on Job Scheduling Strategies for Parallel Processing* (*in Conjunction with IPPS/SPDP'98*), Orlando, FL, U.S.A., Goos G, Hartamanis J, van Leeuwen J (eds.). Springer: Berlin/Heidelberg, 30 March 1998; 62–82. DOI: 10.1007/BFb0053977.

29. Alper P, Bechhofer S, Corcho Ó, Goble C, Kotsiopoulos I, Missier P. An overview of S-OGSA: A reference semantic grid architecture. *Journal of Web Semantics* 2006; **4**(2):102–115.
30. Miles S, Papay J, Payne T, Decker K, Moreau L. Towards a protocol for the attachment of semantic descriptions to grid services. *Second European AcrossGrids Conference*, *AxGrids 2004*, Nicosia, Cyprus, Dikaiakos MD (ed.). Springer: Berlin/Heidelberg, 28–30 January 2004; 230–239. DOI: 10.1007/b99982.
31. Jeffrey SJ, Hunter J. A semantic search engine for the storage resource broker. *Third GGF Semantic Grid Workshop*, Athens, Greece, 2006.
32. Saltz J, Oster S, Hastings S, Langella S, Kurc T, Sanchez W, Kher M, Manisundaram A, Shanbhag K, Covitz P. caGrid: Design and implementation of the core architecture of the cancer biomedical informatics grid. *Bioinformatics—Computer Applications in the Biosciences* 2006; **22**(15):1910–1916.
33. Said MP, Kojima I. SPARQL-based set-matching for semantic grid resource selection. *Second Advances in Semantics for Web Services 2007 Workshop* (*Semantics4ws'07*), Brisbane, Australia, 2007.
34. Said MP, Kojima I. Towards automatic service discovery and monitoring in WS-resource framework. *First International Conference on Semantics*, *Knowledge and Grids*, Beijing, China. IEEE Computer Society: 2005; 106–106.
35. Goble C, De Roure D, Shadbolt NR, Fernandes AAA. Enhancing services and applications with knowledge and semantics. *The Grid 2*: *Blueprint for a New Computing Infrastructure*, Foster I, Kesselman C (eds.). Morgan Kauffman: Los Altos, CA, 2003.
36. Davey S, Dialani V, Fehling R, Fisher S, Gawlick D, Kantarjiev C, Madsen C, Malaika S, Mishra S, Shankar M. Information dissemination in the Grid environment—Base specifications, *INFOD Working Group*, Open Grid Forum, 2007.
37. T-Engine Forum. *Ubiquitous ID Architecture* (*in Japanese*), T-Engine Forum, November 2006. UID-CO00002-0.00.24.
38. Prud'hommeaux E. Federated SPARQL, W3C, 2007.
39. Kojima I, Pahlevi SM. Web Services Data Access and Integration—The RDF(S) Realization (WS-DAIRDFS) RDF(S) Querying Specification, Version 0.1. *DAIS Working Group*, Open Grid Forum, December 2006.
40. Esteban Gutiérrez M, Gómez-Pérez A. Web services data access and integration—The RDF(S) realization (WS-DAI-RDF(S)) RDF(S) ontology access specification, Version 0.8. *DAIS Working Group*, Open Grid Forum, 21 February 2008.
41. Beckett D, Broekstra J. SPARQL Query Results XML Format. *RDF Data Access Working Group Document*, W3C, 12 November 2007.
42. Grant Klark K, Feigenbaum L, Torres E. SPARQL Protocol for RDF, W3C, 12 November 2007.
43. Esteban Gutiérrez M, Gómez-Pérez A, Kojima I, Pahlevi SM. DAIS for RDF(S) Realization—Background and motivational scenarios. *DAIS Working Group*, Open Grid Forum, September 2006.
44. Beckett D. RDF/XML syntax specification (Revised). *RDF Core Working Group Documents*, McBride B (ed.), World Wide Web Consortium, 2004.
45. Hayes P. RDF semantics. *RDF Core Working Group Documents*. McBride B (ed.), World Wide Web Consortium, 2004.
46. Gamma E, Helm R, Johnson R, Vlissides JM. *Design Patterns*: *Elements of Reusable Object-Oriented Software* (*Addison-Wesley Professional Computing Series*). Addison-Wesley Professional: Reading, MA, 1994.
47. Esteban Gutiérrez M, Gómez-Pérez A, Muñoz García Ó. *Deliverable 3.2v2. Deployment of Ontology Services*. OntoGrid Project, 2007.
48. Esteban Gutiérrez M, Gómez-Pérez A, Corcho O, Muñoz García Ó. WS-DAIOnt-RDF(S): Ontology access provision in grids. *Eighth IEEE/ACM International Conference on Grid Computing* (*Grid 2007*), Austin, TX, U.S.A., Fahringer T *et al.* (eds.). IEEE Computer Society: 2007; 89–96.
49. Esteban Gutiérrez M, Muñoz García Ó. Data access and integration services—RDF(S) access glossary of terms. *DAIS Working Group*, Open Grid Forum, October 2007.
50. Sekiguchi S. AIST SOA for building service oriented e-Infrastructure. *Sixth IEEE International Symposium on Cluster Computing and the Grid* (*CCGRID'06*), Singapore, Turner SJ, Sung Lee B, Cai W (eds.). IEEE Computer Society: 2006; 4. Available from: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2006.15.