Accessing RDF(S) Data Resources in Service-based Grid Infrastructures

Miguel Esteban Gutiérrez¹, Said Mirza Pahlevi², Óscar Corcho¹ and Isao Kojima²

1. Universidad Politécnica de Madrids/n, 28660 Boadilla del Monte, Madrid, Spain
   {mesteban,ocorcho}@upm.es
2. National Institute of Advanced Industrial Science and Technology, Tsukuba, 305-856, Japan, {mirza.kojima}@ni.aist.go.jp

Summary

We describe the results of the RDF(S) activity within the OGF DAIS (Database Access and Integration Services) Working Group, whose objective is to develop standard service-based grid access mechanisms for data expressed in RDF and RDF Schema. We are producing 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 OGF23. We conclude by outlining the future work that will be made in the context of this standardisation process.

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 are those 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 standardised 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 like 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 Services), 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 include two main representation languages: RDF [10] 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[1] [12], and of 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 triples2, the IngentaConnect bibliographic metadata storage contains over 200 million RDF triples3, and the CombeChem application manages more than 80 million RDF triples in multiple databases4.

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 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.
2 UniProt Protein Database (http://dev.isb-sib.ch/projetcs/uniprot-rdf)
3 IngentaConnect (http://www.ingentaconnect.com)
4 CombeChem (http://www.combechem.org)
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 BOF 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\(^5\) 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]. Fig. 1 depicts the current DAIS WG Specification portfolio and show 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-grain operations for accessing RDF(S) data resources exploiting the semantics of the RDF Schema model and the latter relies on the usage of the SPARQL RDF query language [20] for expressing declaratively the data that is to be retrieved.

Fig. 1 The current Data Access and Integration Services portfolio and the new RDF(S) realization

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 workplan to conclude the development of the realization, and Section 5 draws conclusions.

\(^5\) 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.
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.

2.1. Grid Resource Matchmaking in Virtual Organisations

**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 middleware such as S-MDS [14], S-OGSA [29], GRIMOIRES [30], S-SRB [31], CaBIG data access services [32], etc.

**Goal.** Given 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.** Fig. 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 an 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].

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.

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 grain operations for operating over the metadata is worthwhile.

Use Case. Fig. 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).

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

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. 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 stored in a wide area distributed database (UCR database). There have been efforts to implement UCR databases using SPARQL-supported RDF databases\(^6\).

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

---

\(^6\) Nihon Unisys SSDB (http://dev.tyzoh.jp/trac/semi-structure-db)
Use Case. Fig.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 access mechanism to the heterogeneous RDF databases for distributed query processing. The indirect data access of the proposed specification (SPARQLAccess) can be used in the distributed query processing, as was done in OGSA-DAI-RDF⁷. Another attempt at 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 taken to mean any system that can act as a source or sink of RDF data, together with its associated management infrastructure that exhibits capabilities that are characteristic of RDF repositories, e.g. RDF and RDF Schema entailment support.

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 is to be retrieved, while the latter defines a set of fine-grain 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 provides the set of interfaces, operations and properties required for dealing with RDF(S) data resources according to the approach followed. The rest of the section presents these specifications in detail.

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

3.1.1 Foundations

The objective of the querying specification 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 update languages to be conveyed to the appropriate data resources, for instance RDF(S) data resources or data resources that supports RDF type queries. The query language supported is the W3C recommended SPARQL [20].

From an application’s point of view, the data in an RDF format is 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, as was shown in the example in section 2.3.

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 try to keep as much compatibility with the existing W3C standards while satisfying the core WS-DAI model. It is also important to provide the useful grid-specific functionalities specified in the core WS-DAI specification such as the indirect access, as a minimum extensions to the existing standards.

### 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 database access modes which the WS-DAI core model provides. 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 Fig. 5 a consumer uses the SPARQLExecute message to submit a RequestDocument in a format defined in [42]. The associated SPARQLQueryExecuteResponse message will contain a set of query result items. Message patterns and XML data structure of the interface is defined based on W3C standards described above.

![Fig. 5 Direct access using the SPARQLExecute interface](image-url)
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-DAIRDFS 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). A solution to the query is obtained by matching graph patterns to subgraphs of the target RDF graphs. However, as shown by the previous use cases, applications may require SPARQL to perform RDF schema based reasoning on the RDF data in order to semantically match resources. The lack of this important feature has forced most RDF database products to provide reasoning functions for SPARQL implemented in various ways\textsuperscript{8,9,10}.

In line with this, we are currently considering the provision of an 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.

3.1.2.2. Indirect Access Interfaces

Indirect access is another access pattern which 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 is not returned directly to the consumer. SPARQL is a similar language to SQL and can return a huge amount of data as a result of a query. Indirect 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\textsuperscript{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 is 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 (Prud'hommeaux, E. 2007). This problem will be a 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 useful 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 a 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 Fig. 6 presents an RDF(S) data service that implements theSPARQLFactory interface. The SPARQLExecuteFactory operation is used to make the results of a query available through, potentially, a separate data access service; for

\textsuperscript{8} Oracle Semantic Technology Center (http://www.oracle.com/technology/tech/semantic_technologies)

\textsuperscript{9} AllegroGraph (http://www.franz.com/products/allegrograph)

\textsuperscript{10} SPARQL Implementations (http://esw.w3.org/topic/SparqlImplementations)
example, a data access service which 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 is no longer made available through a service implementing direct data access interfaces, hence the service does not provide facilities for submitting SPARQL expressions.

![Diagram of indirect data access to the results of an SPARQL query](image)

**Fig. 6 Indirect data access to the results of an SPARQL query**

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 the 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 present the query results 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: **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.

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)\(^\text{11}\). Nevertheless, the vast majority of these systems are oriented towards 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 necessities 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, whilst 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.

\(^{11}\) http://esw.w3.org/topic/SemanticWebTools
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 Fig. 7 shows which are the data resources defined in the specification and the relationships existing between them using a UML notation.

![Data resource model](image)

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

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 to a particular RDF(S) resource that is an individual one 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 which is common to all these 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 focussed on specific parts of an RDF(S) source, in particular that associated to 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 to them\(^{12}\), where the key issue is that the *resources belong* to the *repository* (thus, *resource* data resource 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

\(^{12}\) Another way of understanding a repository data resource is as a container of RDF triples.
repositories are not owned by the repository collection but managed, so that these repositories can be managed by several implementations simultaneously.

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 to. 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.

Fig. 8 depicts the interfaces defined and also presents a possible composition of them into data access services. The dashed blue 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 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 in the data that is 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 in the data that is 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 the management of the components that set up the reification.
- **ListAccess**: provides access to particular RDF(S) resources that are RDF collections (List), focusing in 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.
- **ContainerAccess**: provides access to particular RDF(S) resources that are RDF(S) containers, focusing in the management of the members of the container, as well as, the structure of the container, regardless the its specific type\textsuperscript{13}.
- **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 \textit{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 is 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 patterns 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, ClassDescription, 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 user/developer point of view, the usefulness of the specification will depend on his requirements, and specially 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 fulfils them.

\textsuperscript{13} RDF(S) defines 3 types of predefined containers: \textit{seq}, \textit{bag}, and \textit{alt}.
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 in 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 Fig. 9).

Profile 0: Basic RDF support. This profile includes the minimum set of functionalities needed for dealing with RDF data. The functionalities of these 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 Profile 0, and extends them with 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 with 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.

4. Ongoing and Future Work

The WS-DAI-RDF(S) realization is still work 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, whilst 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 [47]. 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 which 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.

Further work is also to be done in the implementation side, as the 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 which is the basis of the WS-DAI-RDFS Query specification. The system is developed on top of OGSA-DAI [14] 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 [15], Jena [16] and Boca [17]. It includes the RDF Schema based reasoning functionalities, and implements a subset of the ontology primitives that are defined in early WS-DAI Ont-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 [18] to get user feedback, which will be of interest for the specification discussion. Fig. 10 shows some screenshots of the OGSA-DAI RDF user interface. The first two starting from the left hand side, show a SPARQL query interface and tquery results. The rest show the (OGSA-DAI) activity workflow editors.

![1) SPARQL query interface and the results 2) Activity workflow editor which supports reasoning.](image)

**Fig. 10 Screenshots of the OGSA-DAI RDF user interfaces**

The Ontology Engineering Group of the UPM [19] keeps working in the RDF(S) Grid Access Bridge prototype developed as part of the OntoGrid project [20] [48]. The system is the reference implementation of WS-DAI Ont-RDF(S), the RDF(S) ontology access specification that has been used as basis for the development of the WS-DAI RDF(S)

14 OGSA-DAI Homepage (http://www.ogsadai.org.uk/)
15 Sesame (http://www.openrdf.org)
18 AIST Database Grid Home (http://www.dbgrid.org/)
19 Ontology Engineering Group Home (http://www.oeg-upm.net)
20 Ontogrid Project Home (http://www.ontogrid.eu)
Ontology specification [49]. Current developments are devoted to covering the full WS-DAI-RDF(S) realization.

5. Conclusions

More and more interdisciplinary projects and activities across the world are creating enormous quantities of RDF(S) data, which has 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 mechanism, which shall enable the integration of RDF(S) data resource 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.

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.

Acknowledgments

This work is supported by several research and development projects. In Japan, our work is funded by AIST-SOA project to provide a semantic SOA infrastructure [50]. In Spain, this work is supported by the EU FP6 OntoGrid project (FP6-511513) funded under the Grid-based Systems for solving complex problems. We would like to thank all the participants of the DAIS-WG, especially Prof. Norman Paton, Dr. Dave Pearson, and Dr. Mario Antoniolletti, 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 Asunción Gómez-Pérez, Masahiko Kimoto, Steven Lynden and Akiyoshi Matono for valuable discussions and comments.

References


46. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley Professional Computing Series. 1994: Addison-Wesley Professional


