Implementation of a Service-Based Grid Middleware for Accessing RDF Databases

Isao Kojima and Masahiro Kimoto

Information Technology Research Institute,
National Insutitute of Advanced Industrial Science and Technology(AIST)
Umezono1-1-4, Tsukuba, Ibaraki 305 Japan
{isao.kojima,m-kimoto}@aist.go.jp

Abstract. This paper presents the design and implementation of a service-based RDF database middleware suite called DAI-RDF, an extension of the OGSA-DAI middlewarewhich currently supports relational and XML databases. DAI-RDF provides various RDF data processing activities including SPARQL query language, ontological primitives, and reasoning functions. Since DAI-RDF is based on service-based grid architecture, it is easy to use to support large-scale distributed semantic grid applications. Performance evaluation, including evaluation of the reasoning functions included, shows the usefulness of the system.

Keywords: Service based Grid, RDF, OGSA, SPARQL, Ontology.

1 Introduction

One goal of Grid computing[1] is to provide useful computational infrastructure which enables highly parallel applications to handle large amounts of distributed data. Along with the growth of the grid application area, the need for Semantic Web technologies, such as ontology handling and inference capability, began to emerge. Semantic Grid [2] is intended to extend the Grid by giving information and services well-defined meanings, better enabling computers and people to work in cooperation. In this activity, there are several important research issues, such as, how to support large-scale semantic web applications using scalable Grid infrastructure, and how to enhance Grid applications with Semantic Web technology.

In Semantic Web applications, RDF (Resource Description Framework) [3] is the essential data structure needed to support semantics. This means that semantic data access infrastructure in the Grid environment should support RDF databases. However, up until now there has been no implementation of grid middleware, and no standard access specification for accessing RDF data. For example, popular grid-based database access middleware suites, such as OGSA-DAI [4] and AMGA [5] were not able to handle RDF data directly.

On the other hand, many RDF databases [6][7][8] support remote access based on the http protocol, and there is also a set of W3C standard specifications for accessing RDF data [9][10] using the SPARQL[11] query language. These RDF database systems with W3C specifications are, however, not sufficient for supporting distributed applications in the Grid environment. For instance, many Grid applications

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 866–876, 2009.
© Springer-Verlag Berlin Heidelberg 2009

utilize a function called "third party transfer" which allows servers to transfer the data to yet another, different (third-party) site. Support of this function is very useful, particularly in cooperative data processing of applications among distributed servers. This function also requires its own security function, called GSI (Grid Security Infrastructure) [1] which supports the delegation of security credentials. However, these functionalities are not supported by existing W3C specifications or by relevant middleware which handles RDF data. Supporting GSI is also important in being able to combine database processing services with various grid tools such as Globus MDS ((resource) Management and Discovery System) [12].

The final issue is that the W3C standard specifications are only for the SPARQL language, and there is no standard access method for handling ontologies, although many existing RDF systems provide similar ontology handling APIs [6][7]. SPARQL is a graph matching language and there are no semantics capabilities, such as those required for RDF(S) and OWL. We need to have ontological access capability if we want to handle the semantics of RDF(S).

Based on these motivations, the authors are working on a set of access protocol specifications for RDF databases called WS-DAI-RDF(S) [13], which can support SPARQL and ontological primitives. The activity is ongoing in the DAIS-WG (Database Access and Integration Service Working Group) in the OGF (Open Grid Forum). The authors are also working on a reference implementation of the middleware called DAI-RDF, which is presented in this paper. An early prototype called OGSA DAI-RDF was presented in [14] and this paper presents the current implementation, which satisfies the in-discussion WS-DAI specification for RDF [15].

The structure of the paper is as follows. Section 2 presents an overview of the approach of DAI-RDF. The architecture and the structure of the middleware suite are discussed in Section 3. Performance evaluation results are shown in Section 4. Section 5 describes the current status and the future direction of the software.

2 The DAI-RDF Approach

In order to overcome the problems touched on above, the approach in DAI-RDF is to extend the OGSA-DAI [4] database middleware suite to support RDF databases. OGSA-DAI is a database access middleware suite which is in development at OMII (Open Middleware Infrastructure Institute)-UK. It provides web service-based database access and data processing functionalities based on the OGSA architecture, which is defined in OGF. Currently, OGSA-DAI supports two types of database system. The first type is relational databases, such as Oracle, MySQL and PostgreSQL. The second type is XML databases, such as eXist and Xindice.

Based on OGSA-DAI, we have achieved the implementation of the following functional features.

A Secure Distributed Processing Framework Utilizinging GSI: As described in the previous section, supporting third-party transfer is very useful for achieving distributed query processing, which allows the exchange of intermediate results between database servers. In this case, the authentication between one server and another server (a third party site) might cause a security problem that can not be solved by the simple authentication mechanism of the client. GSI provides a safe

authentication infrastructure based on the delegation of credentials, and OGSA-DAI supports GSI. GSI support is also used to combine database processing with other grid services, such as computational services and data transfer services, under one single authentication mechanism.

This function has also proved useful in our S-MDS application [16, 17], which combines grid tools and RDF data processing.

Easy-to-Use Programming for Service-Based Distributed Database Processing through Support of the OGSA-DAI Activity Framework: In the case of distributed database processing, it is often necessary to perform a series of data processing tasks on one site. An example could be as follows,

1) Access the database server using the SPARQL query language
2) Join the results with other results which came from other SPARQL servers.
3) Compress or convert the resulting join.
4) Send the result to another server with HTTP (or another protocol).

It is obvious that this sequence of operations can be achieved with a web service-based workflow language. However, it is not efficient to perform this set of operations on a single server using HTTP-based inter-service protocols. OGSA-DAI provides a function unit called Activity, and several Activities can be combined into a Workflow. A Workflow is a unit of the request to the database service. In this case, each of above four operations can be accomplished as Activities, and a Workflow, which is composed of a set of Activities, is sent to the server as one request. The Activity Framework provides an easy-to-use distributed RDF database programming environment with little overhead, even when combining several separate data operations.

It should be noted that all existing OGSA-DAI activities, such as data compression, data conversion, and data transfer activities, can be combined with our new RDF processing activities. These activities, including third party transfer, will also be useful as the implementation base for distributed RDF query processing problems [18, 19], and for a federated SPARQL [20, 21].

Interoperability for Accessing Various Databases by Implementation of OGF Standard WS-DAI: In the DAIS-WG of OGF, there is an access standard called WS-DAI (Web Service Database Access and Integration), which provides service-based remote database access. The core WS-DAI specification [22] provides a model independent access pattern, and two specific implementations, WS-DAIR [23] and WS-DAIX [24] are defined to access relational databases and XML databases respectively. The main feature of the WS-DAI specification is to provide indirect result transfer when the size of the result is very large. Currently, there are ongoing attempts to test the interoperability between OGSA-DAI and AMGA.

Based on the same approach, the authors are also working on another WS-DAI specification for accessing RDF data resources (WS-DAI RDF(S)) [13]. In this specification, the SPARQL query language and ontological primitives are both supported. Our DAI RDF is also intended to be a reference implementation of WS-DAI RDF(S), able to access different RDF products with a single access protocol.

Moreover, it will be easy to interoperate with various other database products if we support RDF databases within the same WS-DAI framework.

In summary, extending OGSA-DAI to support RDF is very useful for achieving RDF database infrastructure over the Grid. Our work is unique since there are no other activities aimed at extending OGSA-DAI to support RDF databases, especially in synchronization with activities to set standard specifications to access an RDF database under the WS-DAI of OGF.

3 The Architecture and Implementation of DAI-RDF

Here, we present the design approach for the architecture and the actual implementation of DAI-RDF.

3.1 Overview of the System

Fig.1. shows the architectural overview of DAI-RDF. Basically, we added the *RDF data resource type* to the existing OGSA-DAI data resource types. The original data resource types consist of relational and XML resource types.

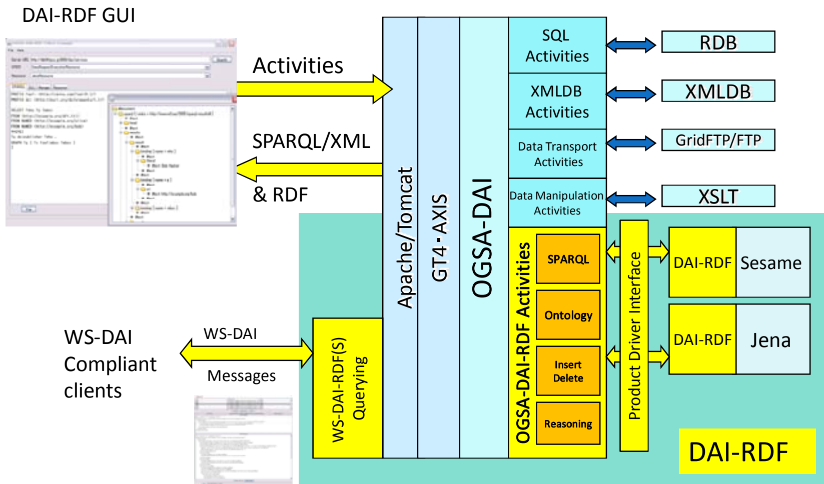

Fig. 1. The structure of the DAI-RDF system

For RDF data resource types, we have implemented a set of Activities as categorized into the following types;

1) **SPARQL Activity** (*SPARQLQueryActivity*)**:** Accesses a data resource using the W3C SPARQL query language.
2) **Reasoning Activity** (*ontologyReasonerActivity*)**:** Performs the reasoning function on a set of RDF triples (RDF Graph) based on certain semantics schemes, like OWL or RDFSchema.

3) **Graph Management** (*GraphManagementActivity*) **and Triple Management** (*TripleManagementActivity*) **Acitivities:** Provides basic I/Os for RDF Graphs and Triples. This enables insertion, deletion, and modification of the target RDF graphs and/or triples. (Currently, we are not able to handle updates for the inferred graphs)
4) **Ontology Handling Activities:** Provides a basic set of ontological primitives. Since the ontological specifications in WS-DAI-RDF(S) [13] are still under discussion, our implementation is based on an early version of WS-DAIOnt [25] and the current WS-DAI-RDF(S) Ontology (Profile0).

We also provided basic interfaces for the still-under-discussion OGF standard WS-DAI RDF(S) Querying [15] on top of DAI-RDF (except for the factory pattern).

Since W3C has already set the standards related to SPARQL [11], we implemented these interfaces so as to have as much compatibility with these standards. For instance, the query is sent with a format based on the W3C *SPARQL Protocol for RDF* specification [9]. The result bindings of a SELECT query are also based on the W3C *SPARQL Query Results XML Format* [10]. For application programming, each Activity also supports a java API in order to provide a java programming environment, as shown in Fig.2.

```
// Get DRER.
                DataRequestExecutionResource drer =
                       server.getDataRequestExecutionResource(drerID);
// Build the individual activities making the workflow and connect them up.
//SPARQL Query Acitivity.
                   SPARQLQuery query = new SPARQLQuery();
                   query.setResourceID(dataResourceID);
                   query.addExpression("   SELECT ? Title WHERE{    ………   }           ");
                    ………..
                  query.addOutputType(xmlUtils.getOutputType());

//Request Status.
                     DeliverToRequestStatus deliver = new DeliverToRequestStatus();
                     deliver.connectInput(loadTuples.getDataOutput());

// Workflow Creation and construction by Adding activities
                     PipelineWorkflow workflow = new PipelineWorkflow();
                            …..
                     workflow.add(query);
                     workflow.add(deliver);
// Execute.
              try{
                          RequestResource requestResource = drer.execute(pipeline,
                                                 RequestExecutionType.SYNCHRONOUS);
                }
```

Fig. 2. A portion of a sample of java code for DAI-RDF

As in Fig.2, a unit of OGSA-DAI execution is called a workflow. In this workflow example, the SPARQLQuery Activity (query) and the DeliverToRequestStatus Activity (deliver) make up a workflow (workflow.add). These activities can be executed in parallel if there is no relationship between the activities.

3.2 Implementation Issues and the Solutions

Here, we present the implementation issues encountered in developing DAI-RDF, and their solutions.

1) **Provide a Product Driver Interface**

In RDF database systems, there are no common access protocols such as the JDBC interface for relational databases. Although the SPARQL protocol supports an HTTP-based access protocol, there is no common interface for ontology handling and graph management functions. This means that RDF database middleware must handle every variation in software products. This might lead to a lack of extensibility for the future support of new RDF databases. Also, DAI-RDF is an extension of OGSA-DAI middleware solutions, so that all DAI-RDF components must be modified whenever OGSA-DAI is changed. In order to solve these issues, we designed a platform-independent driver interface as shown in Fig.1. For instance, if you want to set up a Jena database, you need to specify only the Jena driver class when configuring the RDF data resource. Currently we support Jena, Sesame, and Boca by implementing corresponding database drivers. This architecture provides the extensibility for supporting other RDF database products. By using this structure, we will not need to modify the driver even when OGSA-DAI middleware is upgraded.

2) **Implementation of a Reasoning Activity**

Reasoning is supported in various ways in existing software products. For instance, the following types of reasoning support are provided by SPARQL.

1. *Creation of an inferred graph separately*. Reasoner creates yet another graph and a query with reasoning is performed on the created graph. An example is Jena.
2. *Supply of a Reasoning Switch:* An option is provided to choose whether the query processing needs a reasoning function or not. If the reasoning switch is ON, the SPARQL query processing is done with reasoning. Several systems such as AllegroGraph also provide this option.
3. *Configuration of the RDF resource with a reasoning option.* In Sesame2, a user can configure the resource with reasoning. In this case, the query is always carried out with reasoning if the resource is configured to do so.

The problem here is what model should be used if we want to support all three product types in a single framework. It must be said that it is very difficult to integrate them into a single model. In DAI-RDF, we have adopted a model based on 1. which splits the reasoning function with the SPARQL interface. The reason for the split is that, the reasoning function is currently outside of the focus of SPARQL and related W3C standards. In this approach, the inferred graph is created explicitly in a separate reasoning activity. However, this might cause a performance problem, and we will evaluate this approach in Section.4.

3) **Support of Two I/O types for all RDF Activities (Streamed data and URLs)**

The main target of a SPARQL query is generally a set of graphs (default graphs and named graphs) specified with URLs. On the other hand, Activity output and input is generally a piped stream of data. Thus, all DAI-RDF activities support these two types of I/O parameter patterns, as shown in Fig.3. For instance, any graph in the input set of graphs of a SPARQLQuery Activity can be a set of URLs or a set of the output of other Activity output streams. This achieves flexibility in database processing workflow programming. For Stream, we supports several binary data formats (such as the Jena model and the Sesame repository), along with the standard data formats (XML and N3).

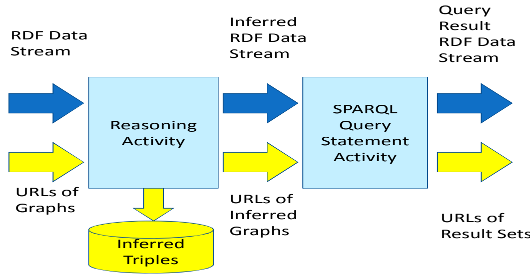

Fig. 3. Piped Input/Output and URLs for connecting Activities

```
// Get DRER.
                                              ,,,,,,,
//  Reasoner
        GetReasoner reasoner = new GetReasoner();
        reasoner.setResourceID(dataResourceID);
        DAIRDFXMLUtilities xmlUtils = new DAIRDFXMLUtilities(reasonerConfigFile);
          ArrayList<String[]> schemaGraphStreamData
                = xmlUtils.getStreamDataList("schema-graph-stream-data");
        ArrayList<String[]>instanceGraphStreamData
                = xmlUtils.getStreamDataList("instance-graph-stream-data");
        ArrayList<String> instanceGraphUri
                = xmlUtils.getGraphUriList("instance-graph-uri");
        ArrayList<ObtainFromHTTP> deliver = new ArrayList<ObtainFromHTTP>();
                                               ,,,,,,,,,
        reasoner.addInstanceGraphsAsArray(instanceGraphURIArray);
        reasoner.addReasoningDataOutputType(xmlUtils.getSimpleTagElement
                ("data-output-type"));
        reasoner.addOntologyReasoningType
                (xmlUtils.getSimpleTagElement("ontology-reaonser-type"));
                                                                 ….. 
 // SPARQL query.
        SPARQLQuery query = new SPARQLQuery();
        query.setResourceID(dataResourceID);
        query.addExpression(xmlUtils.getSimpleTagElement("expression"));
        query.connectReasoner(reasoner.getDataOutput());
        query.addOutputType(xmlUtils.getSimpleTagElement("sparql-output-type"));
                                                            …
// Workflow.
          ,,,,,,
        pipeline.add(reasoner);
        pipeline.add(query);
        pipeline.add(deliverToRequestStatus);
```

Fig. 4. A portion of a java program which represents the workflow .diagram of Fig.3

Fig.4. shows the portion of java code which connects the Reasoner and SPARQL activities with data streams. Note that these workflows can perform cooperative operations between distributed services, and the user can program a distributed database processing task by constructing a set of workflows.

4 Performance Evaluation

We evaluated[1] the performance of DAI-RDF. The first item we analyzed was the response time of SPARQL query processing. The data used was Wordnet [26] and the

[1] Opteron2.0Ghz x 2way, 6GB memory, 500GBdisks, SUSE Linux9. Java1.5 Heapsize512MB.

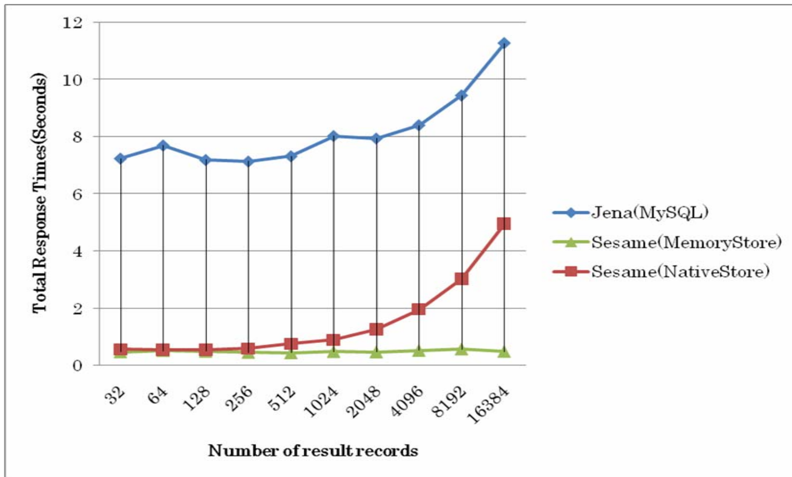

Fig. 5. Query response time per the number of result records returned to the client

Table 1. Detailed time analysis of SPARQL query processing time (seconds)

Number of results	*Total response time (same as in Fig.5-Jena)*	*SPARQLQuery activity processing time*			*OGSA-DAI overhead[28]*
		DAI-RDF middleware overhead	*Query Execution time (Jena)*	*XML data construction time (from model)*	
32	*7.2474*	*0.0462*	*0.0014*	*6.4983*	*0.7012*
64	*7.7051*	*0.0240*	*0.0012*	*7.0637*	*0.6962*
128	*7.1442*	*0.0236*	*0.0017*	*6.9647*	*0.2090*
256	*7.1894*	*0.0220*	*0.0019*	*6.6118*	*0.5893*
512	*7.3276*	*0.0268*	*0.0024*	*6.6345*	*0.6626*
1024	*8.0345*	*0.0695*	*0.0015*	*6.6079*	*1.3555*
2048	*7.9485*	*0.0227*	*0.0014*	*6.9375*	*0.9867*
4096	*8.4111*	*0.0240*	*0.0020*	*6.1200*	*2.2651*
8192	*9.4505*	*0.0256*	*0.0015*	*6.7810*	*2.6422*
16383	*11.2763*	*0.0218*	*0.0016*	*6.9483*	*4.3045*

result is shown in Fig.5 and Table 1. Fig.5 shows the response time using the Sesame and Jena software. As shown in Fig.5, the result depends on the performance of the underlying software implementation and the size of the resulting data.

Table 1 shows a detailed analysis of the case in Fig.5 (Jena with MYSQL). As this result clearly shows, the most time consuming factor is constructing the resulting XML data structure from the Jena model. Currently it is done by triple-by-triple processing using the Jena API. Since the Sesame implementation can output XML directly, the time difference in Fig.5 is caused mostly by this step. As in 3.2, DAI-RDF supports a binary format (the Jena model or the Sesame Repository) for intermediate data transfer so that this step is only used in the final step of the data processing task. In general, the overhead of DAI-RDF is not a performance bottleneck factor.

Table 2. Time analysis of Reasoning with SPARQL query processing time (seconds)

Server Implementation	Total Response Time	Reasoner (RDFS)		SPARQLQuery Activity		
		DAI RDF overhead	Reasoning execution time	DAI-RDF overhead	Query execution time	XML data composition
Jena (mysql)	1.0435	0.051	0.397	0.0553	0.0059	0.049
Sesame (MemoryStore)	1.6494	0.063	0.763	0.0540	0.0537	
Sesame (NativeStore)	1.6053	0.116	0.773	0.0448	0.0442	

Table 3. The results of LUBM benchmarks (seconds)

Q1	Q2	Q3	Q4	Q5	Q6	Q7
4.0097	timeout	6.946	4.848	16.8041	6.3892	timeout

Q8	Q9	Q10	Q11	Q12	Q13	Q14
109.5456	timeout	6.1108	3.4489	3.1776	8.9399	6.0145

Table 2 shows the performance of reasoning/inference processing using Sesame and Jena. The data is in OWL format from [16] and the reasoning is done with RDF(S). The result shows that combining two activities does not cause the performance problem described in Section 3.2. The reason is the implementation of OGSA-DAI in which activities are linked together as a single program (especially in DAI3.0)

Finally, Table 3 shows the result of using the LUBM benchmark [29] to show the scalabitliy of the middleware for large datasets and applications. Compared with [29], this result shows the reasonable response times obtained, so that the DAI-RDF middleware suite can be considered useful for large scale applications. In summary, we believe that our middleware suite provides reasonable performance for constructing distributed RDF applications.

5 Conclusions

The current version of DAI-RDF is now available at http://www.dbgrid.org/. The platforms supported are DAI2.2 and DAI3.1. The WS-DAI RDF(S) interface is not fully supported since we found that JAXB [30] could not create the appropriate EndPointReference which the WS-DAI Core [22] requires. The following software was used. 1) The Semantic Service Registry of AIST Semantic SOA [31] was used for tracking our internal application. The repository is used for finding service metadata which is written in OWL-S and RDF. 2) Metadata Storage of the S-MDS extensions [17] was also used. Ongoing development includes distributed SPARQL query processing based on an extension of our other product, OGSA-DQP/XML[32].

References

1. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure, 2nd edn. The Elsevier Series in Grid Computing. Morgan Kauffman, San Francisco (2003)
2. Semantic Grid Community Portal, http://www.semanticgrid.org
3. Resource Description Framework, http://www.w3.org/RDF/
4. OGSA-DAI Project, http://www.ogsadai.org.uk/
5. ARDA Metadata Catalog Project, http://amga.web.cern.ch/amga/
6. Jena Semantic Web Framework, http://jena.sourceforge.net/
7. Sesame :RDF Schema Querying and Storage, http://www.openrdf.org/
8. IBM Semantic Layered Research Platform (Boca), http://ibm-slrp.sourceforge.net/
9. Clark, K.: SPARQL Protocol for RDF, W3C Recommendation (January 15, 2008), http://www.w3.org/TR/rdf-sparql-protocol//2006/CR-rdf-sparql-protocol-20060406/
10. Beckett, D.: SPARQL Query Results XML Format, W3C Recommendation (January 15, 2008), http://www.w3.org/TR/rdf-sparql-XMLres/
11. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF, W3C Recommendation (January 15, 2008), http://www.w3.org/TR/rdf-sparql-query/
12. Globus Monitoring and Discovery System, http://www.globus.org/mds/
13. Esteban, M., Kojima, I., Mirza, S., Corcho, O., Gomez, A.: Accessing RDF(S) data resources in service-based Grid infrastructures. Concurrency and Computation: Practice and Experience 21(8), 1029–1051 (2009)
14. Kojima, I.: Design and Implementation of OGSA-DAI-RDF. In: 3rd GGF Semantic Grid Workshop, Athens, Greece (2006)
15. Kojima., I., Said, M.P.: Web Services Data Access and Integration – The RDF(S). Realization (WS-DAIRDF(S)) Querying, Open Grid Form (2009), http://forge.gridforum.org/sf/go/doc14074?nav=1
16. Said, M.P., Kojima, I.: S-MDS: Semantic Monitoring and Discovery System for the Grid. Journal of Grid Computing 7(2) (2009)
17. Said, M.P., Kojima, I.: Semantic Grid Resource Monitoring and Discovery with Rule Processing based on the Time-Series Statistical Data. In: Grid 2008, pp. 358–360 (2008)
18. Stuckenschmidt, H., et al.: Index Structures and Algorithms for Querying Distributed RDF Repositories. In: WWW Conference, pp. 613–639 (2004)
19. Kokkinidis, G., Sidirourgos, L., Christophides, V.: Semantic Web and Peer to Peer. In: Staab, S., Stuckendchmidt, H. (eds.). Springer, Heidelberg (2006)
20. Federated SPARQL, http://www.w3.org/2007/05/SPARQLfed/
21. DARQ, Federated Queries with SPARQL, http://darq.sourceforge.net/
22. Antonioletti, M., et al.: Web Services Data Access and Integration - The Core (WS-DAI) Specification, Version 1.0, DAIS Working Group, Open Grid Forum (2006)
23. Antonioletti, M., et al.: Web Services Data Access and Integration - The Relational Realisation (WS-DAIR) Specification, 1.0. DAIS Working Group, Open Grid Forum (2006)
24. Antonioletti, M., et al.: Web Services Data Access and Integration - The XML Realization (WS-DAIX) Specification, 1.0., DAIS Working Group, Open Grid Forum (2006)
25. Gutierrez, M.E., et al.: WS-DAIOnt-RDF(S): Ontology Access Provision in Grid. In: Grid 2007 (2007)
26. WordNet, http://wordnet.princeton.edu/

27. Dobrzelecki, B., et al.: Profiling OGSA-DAI Performance for Common Use Patterns, UK All-Hands Meeting (2006)
28. Wang, K., et al.: Performance Analysis of the OGSA-DAI 3.0 Software, Information Technology: New Generations. In: ITNG 2008, pp. 15–20 (2008)
29. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2) (2005)
30. JAXB, https://jaxb.dev.java.net/
31. Sekiguchi, S.: AIST SOA for Building Service Oriented e-Infrastructure. In: Sixth IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2006 (2006)
32. Lynden, S.J., Pahlevi, S.M., Kojima, I.: Service-based data integration using OGSA-DQP and OGSA-WebDB. In: GRID 2008, pp. 160–167 (2008)