# The WS-DAI Family of Specifications for Web Service Data Access and Integration

Mario Antonioletti,
Amy Krause
EPCC, University of Edinburgh,
Edinburgh EH9 3JZ, UK
(mario, amrey)@epcc.ed.ac.uk

Norman W. Paton
School of Computer Science,
University of Manchester,
Manchester M13 9PL, UK
norm@cs.manchester.ac.uk

Andrew Eisenberg
IBM, Westford, MA 01886
andrew.eisenberg@us.ibm.com

Simon Laws
IBM, Hursley Park
Winchester, SO21 2JN, UK
simon_laws@uk.ibm.com

Susan Malaika
IBM, San Jose, CA 95141, USA
malaika@us.ibm.com

Jim Melton
Oracle Corp., Sandy, UT 84093
jim.melton@acm.org

Dave Pearson
Oracle Corp,Thames Valley Park,
Reading, RG6 1RA, UK
Dave.Pearson@oracle.com

## Guest Column Introduction

This month, we are pleased to provide to our readers a column that addresses an important aspect of grid computing: data access.

Grid computing is important and relevant because it provides middleware that supports secure and managed sharing of networked computational resources. This is valuable because many activities involve collaborations that stand to benefit from more efficient and systematic access to computational and data resources across management domains. The GGF is important because the development of grids depends on shared abstractions and consistent interfaces; the GGF is the principal standards body for grid computing. For the most part, the GGF is developing web service standards for resource management and use that can be used relatively independently or as part of a wider service-based architecture.

The authors of the note belong to two overlapping groups: the chairs of the GGF Data Access and Integration Services Working Group, and the members of the Design Team that was responsible for writing the specifications, and for evolving them in the light of input from other members of the working group and the wider community.

## Introduction

The WS-DAI (Web Service Data Access and Integration) family of specifications defines web service interfaces to data resources, such as relational or XML databases. The specifications have been developed by the Database Access and Integration Services Working Group [1] of the Global Grid Forum [2], and can be used independently, or as part of a wider service-based grid architecture. The specifications include properties that can be used to describe a data service or the resource to which access is being provided, and define message patterns that support access to (query and update) data resources. The specifications include a data model-independent specification (WS-DAI), which is extended in two *realizations* to include model-dependent properties and operations in relational (WS-DAIR) and XML (WS-DAIX) specifications. It is anticipated that further realizations will be developed in due course, for example to support access to RDF and object databases.

The specifications do not provide fully transparent access to existing resources; various messages convey requests written using existing query languages, and the specifications do not impose any requirements for implementers to parse such language statements. Thus, for example, the WS-DAIR and WS-DAIX specifications implement similar message patterns and share the properties defined in WS-DAI, but users of the relational specification express requests in SQL and users of the XML specifications express requests in XPath, XQuery (http://ww.w3.org/CR/) or XUpdate (http://xmldb.org.sourceforge.net/xupdate).

The WS-DAI specifications define web services; as such, they are described using WSDL, and messages are sent to WS-DAI services using SOAP. The specifications have few additional dependencies on web service standards, although they have been designed to be able to be used in

conjunction with emerging standards from the grid and web services communities. In particular, the WS-DAI specifications are part of a wide ranging activity to develop the Open Grid Services Architecture (OGSA) [3] within the GGF. Data can be expected to exist in many forms in grids [4]; file access, movement and replication are central to many grid applications, and database systems are widely deployed in grids both for managing application data and for storing information of relevance to the middleware itself [5].

Many of the other web service specifications for security (e.g. WS-Security [6]), resource description and identification (e.g. the WS-Resource Framework [7]), and transactions (e.g. WS-AtomicTransaction [8]) are likely to be used extensively within service-oriented grids. The current state of play with OGSA is that the first version of a top-level architecture has been defined [3], and working groups are developing specifications that fit into the architecture. The WS-DAI specifications [9][10][11] form part of the OGSA Data Architecture, which will also include services for data movement, replication and storage management.

## WS-DAI: A Framework for Data Access and Integration

The WS-DAI specification defines concepts, properties and messages that can be used in the definition of data services. A *data service* is a web service that implements one or more of the DAIS-WG specified interfaces to provide access to data resources. A *consumer* is an application that exploits the interface provided by a data service to access a data resource.

In principle, a *data resource* represents any system that can act as a source or sink of data. In practice, two kinds of data resource are distinguished in WS-DAI. An *externally managed data resource* is one in which the data is stored using an existing data management system. For example, a relational database of protein sequences stored using an installation of MySQL would be an externally managed data resource. An externally managed data resource:

1. Normally has an existence outside the DAIS service.
2. Has its lifetime managed in ways that are not specified in the DAIS specifications.

The DAIS-WG specifications do not provide operations for carrying out database administration functions on a database management system; instead they provide access to data managed by such systems using the capabilities of a service-based middleware. We note that the specifications are silent about how a service is implemented, so different properties and operations can be supported in different ways. For example, an *external data management system* could be a centralized database or a federation, and a federation could be constructed using WS-DAI services to access the federated resources.

A Service Managed Data Resource, by contrast:

1. Does not normally have an existence outside the service-oriented middleware.
2. Has its lifetime managed in ways that are specified in the DAIS specifications.

For example, the result of a query over a protein sequence database could give rise to a result set that contains information on all the proteins found in yeast. Such a result set could be made available as a data resource in its own right through a data service. As such a data resource has been created by a data service, the service is considered to have responsibility for providing access to the data resource and for enabling the destruction of the resource when it is no longer required; messages are provided to support such capabilities.

Following the OGSA naming scheme [3], data resources may be identified using *abstract names* or *concrete names*. An *abstract name* is a location-independent persistent name, which is a URI. A *concrete name* specifies the location of a data resource. As such, a single abstract name may be used to refer to a data resource accessible through multiple data services, but the concrete name is specific to the location of the service through which the data resource is being accessed.

In WS-DAI, a concrete name may consist of a combination of a service address plus an abstract name, or it may consist of a WS-Addressing endpoint reference as used in the WS-Resource Framework (WSRF) [7]. The WSRF specifies both an approach to resource identification in which resources are identified explicitly in the header of a message, and a range of associated behaviors, for example for representing the properties or the lifetime of a resource. In WS-DAI, messages to a data service must include within their SOAP body the abstract name of the resource being accessed by way of the service; where WSRF is being used to identify a data resource, the address to which a message is sent contains enough information to identify the resource within the data service, and the abstract name is also included in the SOAP body only for consistency with non-WSRF data resources. Both approaches are supported because the WSRF family of specifications is only now completing its standardisation process within OASIS (http://www.oasis-open.org/), and the level of adoption it will experience is not yet clear,

partly due to the presence of competing specifications for resource representation.

The properties of a data service and the principal messages that can be sent to a data service form the core of the specification, and are available whether or not data resources are represented using WSRF. Where data resources are represented using WSRF, the functionalities associated with WSRF for soft state lifetime management and fine-grained property access become available for managing the lifetime of (service managed) data resources and for accessing the properties defined in WS-DAI and in the realizations. For non-WSRF resources, properties are only available through retrieval of the entire property document, and lifetime management is restricted to an explicit destruction of the data service-data resource relationship.

## Properties

It is important that a consumer of a data service can interrogate the service to: (i) determine whether the service is suitable for use in a given setting; and (ii) obtain the information required to enable valid requests to be sent to the service. To support this, the WS-DAI specification defines a collection of properties and provides a *GetDataResourceProperty-Document* operation, which, given a resource name, returns an XML document that describes all the properties associated with a resource on a service. If the data resource is represented using WSRF, the fine-grained operations provided by WSRF can be used to access the properties individually.

The properties in the WS-DAI specification are applicable to any type of data resource; individual realizations extend the properties listed here to include paradigm-specific features, such as the XML Schemas associated with the collections in an XML database. The following properties are defined for all data services:

*DataResourceAbstractName:* URI representing the abstract name of the data resource.

*ParentDataResource:* If this resource was derived from another, this has the abstract name of the parent data resource.

*DataResourceManagement:* An enumeration indicating if the data resource is *ServiceManaged* or *ExternallyManaged*.

*DatasetMap:* A mapping between the QName of a message and the URI of a dataset type representing the result types supported for the messages. For example, as there are many different XML representations for relational result sets, this property allows a service provider to indicate to a consumer the representations that the service can return. A consumer can then specify the format in which data should be returned to them. The dataset URIs are not defined by DAIS.

*ConfigurationMap:* A mapping between the QName of a message and the URI of an expression language. For example, a message which supports XPath queries may support XPath Version 1.0 expressions or XPath Version 2.0 expressions. DAIS does not define new expression languages but wraps those defined by others in DAIS messages. Expression languages are expected to evolve independently of the DAIS specifications. For example, the development of the DAIS specifications has no impact on the further development of the SQL or XQuery specifications. DAIS uses, but does not define, URIs to identify supported expression languages.

Suitably precise identifiers for languages do not always exist. In this regard, it is hoped that vendors and language standards bodies will develop schemes that allow the language supported by a data resource management system to be unambiguously identified.

*LanguageMap:* A mapping between the QName of a message and the URI of an expression language. For example, a message which supports SQL queries may allow such queries as SQL:1999 expressions or SQL:2003 expressions. DAIS does not define the URIs of supported languages.

*DataResourceDescription:* A human readable description of a data resource.

*Readable* and *Writeable:* These properties indicate whether or not the data resource is able to be read from or written to from the data service.

*ConcurrentAccess:* Has the value true if a data service is able to process more than one message concurrently otherwise it has the value false.

*TransactionInitiation:* Describes under what circumstances a transaction is initiated in response to messages. The values are as follows: *NotSupported* – does not support transactions; *Automatic* – transaction initiated automatically for the duration of each message; *Manual Transaction* – context under control of the consumer, for example using an existing transaction specification, such as WS-AtomicTransaction [8].

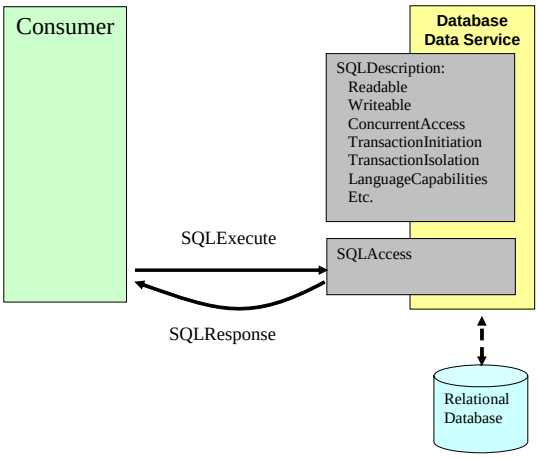
*TransactionIsolation:* describes how transactions behave with respect to other ongoing transactions.

*ChildSensitiveToParent* and *ParentSensitiveToChild:* indicates whether a parent or child data resource is sensitive to changes made to the other.

## *Direct Access*

The DAIS-WG specifications, for the most part, rely on the existing query facilities of database systems for extracting data from and modifying the contents of data resources. Operations are provided that pass query statements as strings, which in turn are passed on to the underlying data resource management system; as such, the specifications can be seen as defining web service wrappers for the underlying databases.

The WS-DAI specification supports two patterns for obtaining the results of requests directed at a data resource, referred to as *direct data access* and *indirect data access*. Direct data access follows the typical web service type of interaction where a consumer expects the data or status of a query in the response to a request. For example, passing a message containing a SQL query to a data service will result in a response message containing the rowset representing the result of the query as an XML document, as illustrated below.




For direct data access, the WS-DAI specification defines a single query-language-independent message, and a template for realizations to follow in defining language-dependent operations.

### Generic Query

A message for passing generic queries to a data resource. The actual query language payload is implicit in the QName of the message, and is specified by a *LanguageMap* property. Operation descriptions are not given in WSDL here; the WSDL is provided in the specification [9]. The following terminology is used to indicate multiplicity in the examples below: a "?" indicates that a parameter is optional; "*" denotes 0 or more; "+" denotes one or more; and no modifiers signifies that one such element must be present.

Input. *GenericQueryRequest*:

1. *DataResourceAbstractName*: abstract name of the target resource.
2. *DatasetFormatURI*: An element that can be used to define the type of the response message. This element must contain a URI from the set that appears in the *MessageDatasetMap* properties defined by the data service. When only one URI is advertised this element may be omitted, in which case the format of the response message will follow that of the type reference by the advertised value.
3. *GenericExpression*: the query expression document.

Output. *GenericQueryResponse*:

1. The *DatasetFormatURI* used to format the response.
2. The response document formatted according to *DatasetFormatURI*

*GenericQuery* is an extensibility point for a service; the specifications are not prescriptive with regard to the languages that might be supported in different contexts, and thus, like other extensibility points, interoperability is traded for flexibility.

### Query Template

The WS-DAI specification does not provide any query-language-specific operations; these are the responsibility of the realizations. Rather, WS-DAI defines a structure for direct data access request messages that is followed by the realizations. The structure is as follows:

```
<RequestMessage>
  <wsdai:DataResourceAbstractName/>
  <wsdai:DatasetFormalURI/>?
  <RequestDocument/>
</RequestMessage>
```

The elements within the template are as follows:

1. *RequestMessage*: This is the root element for a request message. The type of this element is specific to each message. For example, the relational realization defines *SQLExecuteRequest* as a *RequestMessage*.
2. *DataResourceAbstractName*: abstract name of the target resource
3. *DatasetFormatURI*: as in Generic Query.
4. *RequestDocument*: The request statement. The structure of this document is specific to the statement being used. For example, the relational

realization defines a *SQLExpression* element as a *RequestDocument*.

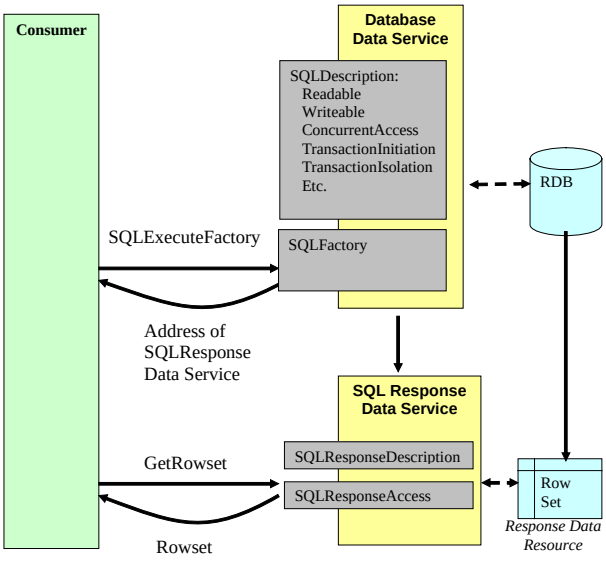
The structure of a direct access response message is:

```
<ResponseMessage>
  <wsdai:DatasetFormat/>
  <wsdai:DatasetData>
    Data formatted according to the
    DatasetFormatURI goes here.
  </wsdai:DatasetData>
</ResponseMessage>
```

## Indirect Access

Indirect data access essentially implements the factory pattern, whereby the result of a request is not returned directly to the user, but rather made available as a data resource in its own right, for access through a data service. The consumer thus gets an end-point reference in the response message through which the data may be accessed. The WS-DAI specification does not define any generic indirect access operations, but it defines a template that must be followed by the realizations that implement this type of operation.

To see an example of indirect access usage we thus turn to the relational realization example.




A consumer sends a *SQLExecuteFactory* message to a *Relational Data Service*. As a result of the message, the data service makes available a service managed *Response Data Resource* through a *SQL Response Data Service*. The result of the *SQLExecuteFactory* message is the concrete name of the associated *Response Data Resource*. Subsequent messages from the same consumer, or a different consumer, can be used to access the contents of the *Response Data Resource*.

This pattern has been introduced into the DAIS-WG specifications in part to support third party delivery (and also to avoid unnecessary data movement). The specifications support third-party-pull delivery directly – a consumer *C1* can invoke a *SQLExecuteFactory* operation and pass the concrete name of the result data resource to consumer *C2*, which can then access the result directly, avoiding the need to transfer the result to *C2* via *C1*. Other patterns of third party delivery, such as third-party-push delivery, are expected to be supported by implementing portTypes from OGSA data movement services on the data services used to provide access to the results of factory operations.

As in the case of direct data access, the specification defines a template that is implemented by the realizations. The structure of a message implementing the factory pattern is:

```
<RequestMessage>
  <wsdai:DataResourceAbstractName/>
  <wsdai:PortTypeQName/>?
  <wsdai:ConfigurationDocument/>?
  <wsdai:PreferredTargetService/>?
  <RequestDocument/>
</RequestMessage>
```

The components of the template are as follows:

1. *RequestMessage*: This is the root element for a request message. The type of this element is message-specific. For example, the relational realization defines *SQLExecuteFactoryRequest* as a RequestMessage.
2. *DataResourceAbstractName*: The abstract name of the target resource.
3. *PortTypeQName*: The QName of the port type that the resulting resource will be accessed through. This must correspond to one of the QNames defined in the *ConfigurationMap* property of the service. If no *PortTypeQName* is specified the port type specified by the first *ConfigurationMap* property is assumed.

6. *RequestDocument*: This is the request part of the message. The structure of this document is message-specific. For example, the relational realization defines a *SQLExpression* element as a *RequestDocument*.

The structure of the factory pattern response message is:

```
<wsdai:DataResourceAddressList>
  <wsa:EndPointReference>
    <wsa:ReferenceParameters>
     <wsdai:DataResourceAbstractName/>?
    </wsa:ReferenceParameters>
  </wsa:EndPointReference>*
</wsdai:DataResourceAddressList>
```

The components of the template are as follows:

1. *wsdai:DataResourceAddress*: This is the root element for the response message. For example, the relational realization defines a *SQLExecuteFactoryResponse* element.
2. *wsa:EndPointReference*: A list of end points of the service(s) that provide access to the newly created data resource(s). The EPR *ReferenceParameters* element contains the abstract name of the data resource to which the address refers.

# WS-DAIR: The Relational Realization

The relational realization [10] extends the properties defined in WS-DAI, instantiates the templates for direct and indirect data access, and defines interfaces for accessing the results of requests directed at a relational data service. The relational realization builds on the SQL standard throughout.

## *Properties*

A relational data service defines a single additional property, namely *CIMDescription*: the description of the database accessible through the service, described using the model of the Database Technical Committee of the Distributed Management Task Force (DMTF – http://www.dmtf.org).

## *Direct Access*

The *SQLExecute* operation directs a SQL statement to a relational data resource, instantiating the template defined in WS-DAI.

Input: *SQLExecuteRequest*. The elements are as defined in the WS-DAI direct access template, where the *RequestDocument* becomes *SQLExpression* which contains the actual SQL query plus optional parameters.

Output: *SQLExecuteResponse*.

1. *DatasetFormatURI*: The format in which the data is being returned.
2. *DatasetData*: Any data returned in response to a query.
3. *SQLUpdateCount**: The number of rows that were affected by an SQL update if this was the type of SQL statement used.
4. *SQLOutputParameter**: Any output from a SQL stored procedure output parameter.
5. *SQLReturnValue?*: The return value from any stored procedure.
6. *SQLCommunicationsArea+*: Any output from the SQL Communications Area.

The following is an example of an input message to *SQLExecuteRequest*:

```
<SQLExecuteRequest>
  <DataResourceAbstractName>
     urn:dais:dataresource27
  </DataResourceAbstractName>
  <SQLExpression>
    <Expression>
      SELECT * FROM P
   </Expression>
  </SQLExpression>
</SQLExecuteRequest>
```

## *Indirect Access*

The *SQLExecuteFactory* operation directs a SQL statement to a relational data resource, instantiating the template defined in WS-DAI. As such, the result of the SQL statement is made available to the consumer as a data resource by way of a data service.

Input: *SQLExecuteFactoryRequest*. The elements are as defined in the WS-DAI indirect access template, where the *RequestDocument* becomes *SQLExpression*.

Output: *SQLExecuteFactoryResponse*. A list of addresses of the data resources for the result(s).

The following is a fragment of an example *SQLExecuteFactoryRequest*, which includes both the query and the properties used to configure the data service that will provide access to the result:

```
<SQLExecuteFactoryRequest>
  <DataResourceAbstractName>
   urn:dais:mydataresource1234
  </DataResourceAbstractName>
  <PortTypeQName>
      dair:SQLResponsePT
  </PortTypeQName>
  <ConfigurationDocument>
    <Readable>true</Readable>
    <Writeable>false</Writeable>
    <ConcurrentAccess>
      false
    </ConcurrentAccess>
    <TransactionInitiation>
      NotSupported
```

```
    </TransactionInitiation>
     …
  </ConfigurationDocument>
  <SQLExpression>
    <Expression>
     SELECT * FROM P
    </Expression>
  </SQLExpression>
</SQLExecuteFactoryRequest>
```

Although the DAIS-WG specifications allow other interfaces to be used to provide access to the results of a *SQLExecuteFactoryRequest*, the *SQLResponse* and *SQLRowset* data services have been defined for this purpose.

### SQLResponse

A *SQLExecuteFactoryRequest* may return the concrete address of a data resource based on the result of any SQL statement. As such, the *SQLResponse* data service provides properties that describe the actual result and operations for accessing those results.

Space does not allow a detailed description to be provided, but the names of the properties are fairly self-explanatory, and are: *NumberOfSQLRowSets*, *NumberOfSQLUpdateCounts*, *NumberOfSQLReturnValues*, *NumberOfSQLOutputParameters* and *NumberOfSQLCommunicationsAreas*.

Associated with these properties are a collection of data access operations, namely *GetSQLResponseItem*, *GetSQLRowSet*, *GetSQLUpdateCount*, *GetSQLReturnValue*, *GetSQLOutputParameter* and *GetSQLCommunicationsArea*.

In a third-party pull data delivery, the results of a query can be obtained by a consumer using the collection of operations defined in *SQLResponse*.

### SQLRowSet

A *SQLRowSet* data service essentially provides access to a single table. It has the properties: *AccessMode*, which indicates if the rowset can be read only sequentially or whether access by position is supported; and *NumberOfRows*, which indicates how many tuples are stored in the rowset. An operation, *GetTuples*, provides access to a group of tuples from the rowset, indexed by position.

## WS-DAIX: The XML Realization

The XML realization [11] extends the properties defined in WS-DAI, instantiates the templates for direct and indirect data access, and defines interfaces for accessing the results of requests directed at a XML data service. Space restrictions preclude a fuller description of WS-DAIX here, but it follows a similar pattern to that described for WS-DAIR. Key features include: properties and operations for manipulating collections of documents, direct and indirect access using both XQuery and XPath, and data modification using XUpdate.

## Conclusions

The development of standards for accessing databases from programming languages is well established, and has saved countless hours of developer effort though improved portability and interoperability. Such standards are widely deployed for relational databases, where Embedded SQL is part of the SQL standards process (ISO/IEC 9075 at http://www.iso.org/) and JDBC® is part of the Java Community Process (http://jcp.org/en/jsr/-detail?id=054). Programming language APIs used to access XML databases are less well established, although several native XML database systems support XML:DB (http://xmldb-org.source-forge.net/xapi/), and the development of the XQJ standard within the Java Community Process is now well advanced (http://www.jcp.org/en/jsr/-detail?id=225). The WS-DAI family of specifications should bring similar benefits to service-based computing, by making data resources available though consistent WSDL interfaces. Benefits that result from the provision of data access standards as web services include: (i) no need to deploy database connectivity software on clients; (ii) seamless integration with other web service standards, for example, for security, transaction management and data movement.

The argument here is not that databases should always, or even normally, be made available in a service-based setting using the interfaces described in this paper; indeed most access to databases in service-based applications will be completely transparent, hidden behind domain-specific services. However, at some level in any distributed application it becomes necessary to direct requests at resources, whether data or computational resources. Service-oriented grid middleware seeks to make the secure and coordinated interaction with distributed computational resources more systematic, more consistent and more compositional, and thus more cost effective. The WS-DAI family of standards seeks to contribute to this process by making databases first class citizens in service-based grids. At the time of writing, the specifications have been submitted by the DAIS-WG to the public comment phase of the GGF standardization process; as such it is anticipated that the specifications will be refined in

the light of feedback for adoption as standards during 2006.

*Availability*: Prototype implementations of WS-DAIR and WS-DAIX will be made available from http://www.ogsadai.org.uk during the first half of 2006.

*Acknowledgements*: Many people have contributed at different stages to the development of the DAIS-WG specifications, including Malcolm Atkinson, Brian Collins, Shannon Hastings, Stephen Langella, James Magowan and Greg Riccardi; fuller acknowledgements are provided on the specifications themselves.



## References

[1] Database Access and Integration Services Working Group (DAIS-WG), http://forge.gridforum.org/projects/dais-wg

[2] Global Grid Forum (GGF), http://www.ggf.org

[3] Open Grid Services Architecture Version 1.0, I. Foster *et al.*, Technical Report GFD-I.30, Global Grid Forum, 2004.

[4] Data Access, Integration and Management, M.P. Atkinson, *et al.*, in The Grid: Blueprint for a New Computing Infrastructure (Second Edition), I. Foster and C. Kesselman, Morgan-Kaufmann, 391-430, 2004.

[5] Grid Database Access and Integration: Requirements and Functionalities M.P. Atkinson, V. Dialani, L. Guy, I. Narang, N.W. Paton, D. Pearson, T. Storey, and P. Watson. Technical Report GFD-I.13, GGF, 2003.

[6] Web Services Security: SOAP Message Security 1.0 (WS-Security), A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. OASIS, 2004.

[7] Web Services Resource Framework (WSRF) Primer, T. Banks, http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf, OASIS, 2005.

[8] Web Services Atomic Transaction (WS-AtomicTransaction) 1.1. E Newcomer, I Robinson (eds), Working Draft, http://www.oasis-open.org/committees/download.php/15719/WS-AT%20Working%20Draft.pdf, 2004.

[9] Web services data access and integration – the core (WS-DAI) Specification, Version 1.0. M. Antonioletti, M. Atkinson, A. Krause, S. Laws, S. Malaika, N.W. Paton, D. Pearson, G. Riccardi. GGF, 2005.

[10] Web services data access and integration – the relational realization (WS-DAIR), Version 1.0. M. Antonioletti, B. Collins, A. Krause, S. Laws, S. Malaika, J. Magowan, N.W. Paton. GGF, 2005.

[11] Web services data access and integration – the XML realization (WS-DAIX), Version 1.0, M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Laws, S.Malaika, N.W. Paton. GGF, 2005