OGSA-DAI 3.0 – The Whats and the Whys

Mario Antonioletti[1], Neil P. Chue Hong[1], Alastair C. Hume[1], **Mike Jackson**[1], Kostas Karasavvas[2], Amy Krause[1], Jennifer M. Schopf[1,2,3], Malcolm P. Atkinson[2], Bartosz Dobrzelecki[1], Malcolm Illingworth[1], Nicola McDonnell[1], Mark Parsons[1] and Elias Theocharopoulos[2].

1. EPCC, The University of Edinburgh, JCMB, Mayfield Road, Edinburgh EH9 3JZ, UK.
2. National e-Science Centre, University of Edinburgh, Edinburgh EH8 9AA, UK.
3. Distributed Systems Laboratory, Argonne National Laboratory, Argonne, IL, 60439 USA.

Abstract

OGSA-DAI provides an extensible framework that allows data resources to be incorporated into Grid fabrics. The current OGSA-DAI release, version 3.0, is a complete top-to-bottom redesign and implementation of the OGSA-DAI product. A number of fundamental conceptual and design changes are introduced in this release. In this paper we describe the motivation behind this redesign and provide an overview of OGSA-DAI 3.0, comparing and contrasting it with OGSA-DAI 2.2. We also outline the implications that these changes have for the OGSA-DAI user community.

1. Introduction

The *Open Grid Services Architecture – Data Access and Integration* (OGSA-DAI) project [1], currently funded as part of the Open Middleware Infrastructure Institute UK (OMII-UK) [2], aims to provide the e-Science community with a middleware solution to provide access to and integration of data for applications working across administrative domains. Early Grid applications focused principally on the storage, replication, and movement of file-based data, but many applications now require full integration of database technologies and other structured forms of data through Grid middleware. Not only do many Grid applications already use databases for managing metadata, but increasingly many are associated with large databases of domain-specific information. For example the AstroGrid [3] and DGEMap [4] projects utilise large databases of astronomical and biological data respectively.

OGSA-DAI offers services that add data access and integration capabilities to the core functionality of service-oriented Grids. Structured data resources, whether these are databases, files, or other types of data, can be made available to Grid applications. OGSA-DAI is now widely used within the UK e-Science community as well as in other Grid projects world-wide [5].

This paper provides an overview of the new version of OGSA-DAI, 3.0, released in June 2007. Since 2002, when the OGSA-DAI project started, there have been regular releases of the OGSA-DAI product at approximately 6 month intervals. However, 3.0 was released a full year after 2.2 in order to allow for fundamental conceptual and design changes. This paper explains the reasons for this and gives a summary of the improvements that users experience as a result of the changes made. Section 2 acquaints the reader with some of the more common concepts and terms that will be used in the paper. Section 3 provides a brief motivation for the changes, highlighting the evolution of previous OGSA-DAI releases. Sections 4–7 describe the different improvements made in OGSA-DAI 3.0. The implications of migrating to release 3.0 for users of 2.2 and earlier releases are then outlined in Section 8. A brief summary of related work is presented in Section 9. Finally, we conclude, in Section 10, with information on the future plans of the OGSA-DAI project.

2. OGSA-DAI Concepts and Terms

Comprehensive descriptions of OGSA-DAI are available in [1][6][7][8]. Some of the basic concepts are:

Activity – a named data-related operation supported by OGSA-DAI. Example activities include executing an SQL query, compressing a set of data, listing the files in a directory, performing a transform on an XML document, and delivering data to an FTP server. Outputs from an activity may be connected to inputs to

another activity to form a workflow. Compositions of activities enable a powerful data transforming facility that can move computations closer to the data.

Block – a chunk of data, for example a number, a boolean, a string, a byte, a byte array, or a tuple.

Workflow – one or more sequences of connected activities. A workflow can have multiple sequences of activities that can be executed in series or in parallel.

Request – a workflow submitted by a client to OGSA-DAI. A request is a workflow with a unique identity.

Session – a named context that allows multiple requests to share state.

Activity framework – a software component that creates, initialises, and processes workflows and activities contained in a request.

Engine – a software component that queues requests and submits them for processing to the activity framework. The engine is a generic component and has no knowledge of activities or workflows.

OGSA-DAI core – software components that provide the fundamental OGSA-DAI functionality, including the activity framework and engine as well as data resource interaction components, persistence and configuration components, and utilities.

Presentation layer – a front-end to the OGSA-DAI core via which clients can access and use OGSA-DAI. These can expose the OGSA-DAI core via Web services.

Data service – a Web service that provides access to the OGSA-DAI core. Part of a Web services-based presentation layer.

3. OGSA-DAI 3.0 – a New Genesis

Every OGSA-DAI release up to and including OGSA-DAI 2.2 (April 2006) has been an evolution of the previous version. With OGSA-DAI 3.0 we have taken the decision to implement a complete top-to-bottom redesign and rewriting of OGSA-DAI from scratch which will stabilise the OGSA-DAI framework for developers and end users.

The number of OGSA-DAI users has increased to over 2800 in the last five years, and as those numbers have grown the requirements OGSA-DAI must satisfy have grown as well. OGSA-DAI 2.2 was not suitable for addressing this set of requirements, in particular those relating to data streaming, standardisation of activity inputs, and outputs and supporting arbitrary presentation layers in a straightforward way.

Therefore, to address this need OGSA-DAI was redesigned from scratch, including the activity framework and engine through to the service-based presentation layers. This peer reviewed design phase was followed by an extensive testing phase which has extended the coverage of our tests, and provided a new set of scenarios to test against.

4. Improvements to Activities

The activity model in OGSA-DAI allows new functionality to be created, deployed and used in OGSA-DAI services in a similar way to plug-ins in many other pieces of software. This provides two important benefits to developer-users: a well-defined, composable, workflow unit, and an easy way to extend functionality without needing to understand the underlying infrastructure.

4.1 Defining a Standard Activity Library

In previous versions of OGSA-DAI, the set of available activities grew in a relatively *ad hoc* manner as specific requirements were met. Consequently, these activities did not form a complete set, lacked consistency, and did not compose easily.

In release 3.0, a complete analysis and redesign of the activity library gave us the opportunity to improve on these shortcomings. Specifically, having more experience of what our users want, we defined a specification of a comprehensive and consistent standard set of activities [9] that covers the data access and integration requirements of many users.

4.2 Activity Structure

In OGSA-DAI release 2.2, activity input, output, and parameter naming were activity specific and tied to the activity representation. Parameters differed from inputs in that they were specified once in the request. Previous activities in the flow could not provide a parameter value to a downstream activity at run-time – inputs and parameters were not interchangeable which meant it was difficult to compose activities and enforce connection semantics.

In release 3.0, we have formulated a generic activity representation where inputs and parameters have been homogenised – now we only have inputs. A special kind of input, called an *input literal*, replaces parameters in requests. Since we have a common passing mechanism for all inputs, clients are free to choose whether an input value is specified in the request or is obtained from the output of another activity.

4.3 Activity Inputs and Outputs

The new set of recommended inputs and outputs defined in the activity library set includes Java's basic types (Object, String, Integer, Long, Double, Number, Boolean) as well as some other types like char[], byte[], Clob, Blob. We have also defined two types – tuple and MetadataWrapper – that are of particular importance.

A *tuple* is used to represent a row of relational data. Each element of a tuple represents a column. Relational resources are commonly used in many OGSA-DAI scenarios and using a preferred (and recommended) format has many benefits, one of which is that we need only one set of transformation activities, e.g. for projection, sorting, etc. In previous releases we used more types internally, including CSV, WebRowSet and ResultSet so for example, it was possible for one activity to project data represented in CSV and another to project data represented in WebRowSet. Release 3.0 uses data conversion activities to convert tuples to CSV or WebRowSet if required.

A *MetadataWrapper* can wrap any object that the users want to be treated as metadata by activities. It is left to individual activities to handle metadata blocks as they see fit. OGSA-DAI is agnostic of metadata representations and allows users to use any kind of domain-specific metadata they might need.

4.4 Activity APIs

The redesign of activity inputs and outputs also yields a simpler API for activity developers. This has been simplified further by the fact that activity developers no longer need to parse XML fragments to elicit activity input and output names and configuration settings. The activity framework provides this information in a standard set of Java objects. Activity developers also do not need to write XML Schema documents describing their activities.

Activities can implement "extension" interfaces to specify what they require from the activity framework to be able to execute, for example: Do they need data resources? Do they spawn activities? Do they produce events for monitoring? Do they need a security context? This provides a simpler method for activity developers to specify to the activity framework their initialisation and configuration requirements and removes some of the overhead of developing this themselves.

5. Improvements to OGSA-DAI Workflows

The OGSA-DAI activity model allows activities to be composed together. The OGSA-DAI activity framework and engine, which execute the data-centric workflows from clients, have been completely rewritten to support a number of scenarios which benefit developers and users of OGSA-DAI.

5.1 Lists and Data Grouping

In OGSA-DAI 3.0 we have introduced *lists*, blocks that are inserted into a stream of data to mark the beginning and end of a group of data blocks. A list groups related data together as one unit. For an example consider the *SQLQuery* activity in Figure 1.

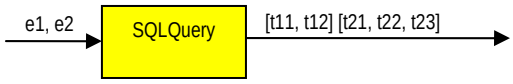

Figure 1 – Two executions of an SQLQuery.

SQLQuery can dynamically take any number of SQL expressions as input. The output for each expression is a number of rows represented as tuples. Without a way to group the output tuples we would have no way to differentiate between the results of queries e1 and e2. We can use lists (illustrated with the square brackets in the figure) to group the output of each execution of the query. This results in two lists, one with the output of e1 and one with that of e2. Lists play an important role in the streaming of data through activities. They allow us to describe activities in terms of different granularities instead of only different types in terms of their inputs and outputs. For example an activity could take a simple string per execution or a list of strings per execution and then iterate over the strings internally. That decision is up to the activity developer. In our standard activity library we have defined activities in their lowest sensible[1] granularity to enhance composability. To homogenise granularity, release 3.0 provides utility activities that allow lists to be flattened,. *ListRemover*, or flattened and then reconstructed, *ListProcessingWrapper.*

[1] Sometimes we chose not to use the lowest possible granularity for optimisation reasons.

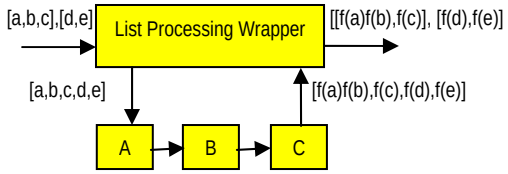

Figure 2 - Flattening and reconstructing lists.

Figure 2. shows an activity A that takes a single element as input for one execution while the initial input that *ListProcessingWrapper* gets is a list. Effectively, the *ListProcessingWrapper* activity iterates over the activities A, B and C by providing the single values that are required while at the same time adding the correct list markers in the output to maintain the relationships between the inputs and outputs.

5.2 Workflows and Data Resources

One major change in OGSA-DAI 3.0 is to support workflows that target multiple data resources.

OGSA-DAI 2.2 (see Figure 3, top half) had a data service resource (DSR) which was associated with a single data resource, e.g. an instance of a database. A DSR was accessed via a data service, which could expose any number of DSRs, and thus any number of data resources. Each DSR used its own instance of an engine and activity framework to execute workflows received from clients. A major shortcoming of this model was that a workflow submitted to a specific DSR could not access, or reference, data resources exposed by other DSRs under the same data service.

In OGSA-DAI 3.0, the DSR has been superseded by the *data request execution resource* (DRER) (see Figure 3, bottom half). In common with a DSR, a DRER uses an engine and activity framework to execute workflows. Unlike DSRs however, a DRER can be associated with any number of data resources and different activities within the same workflow can be targeted at different data resources known to the DRER.

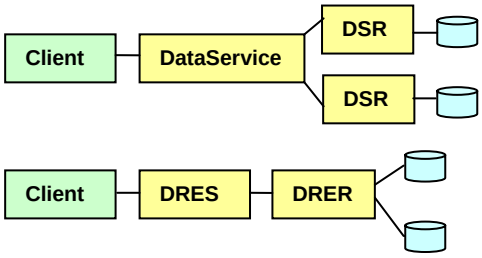

Figure 3: Services and resources in OGSA-DAI 2.2 (above) and 3.0 (below).

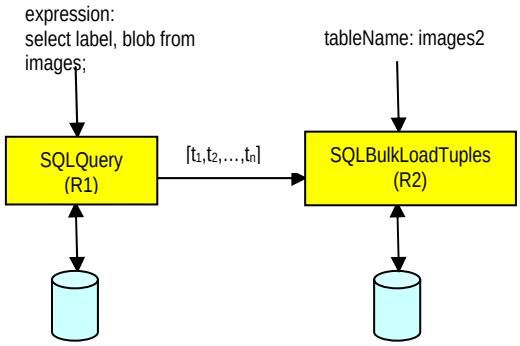

Figure 4: Using multiple resources in the scope of a single request.

This redesign allows OGSA-DAI to execute workflows, for example as shown in Figure 4, where a transfer of data from data resource R1 to data resource R2 does not include transferring data external to the data service. An SQL query on R1 produces data, which is then written to R2 via the *SQL Bulk Load Tuples* activity. In OGSA-DAI 2.2 this scenario would have required the submission of requests to two OGSA-DAI services and the transfer of the data from one service to the other.

The restructuring in 3.0 allows clients to undertake data transformation and integration scenarios across various data resources within the scope of a single OGSA-DAI workflow.

5.3 Resource Groups

One of the features of OGSA-DAI 2.2 was the *multi-resource* – a data resource that opaquely federated N DSRs (and so N data resources) through the same data service. This is still supported in OGSA-DAI 3.0 via the notion of a *resource group* – a data resource that maintains the identifiers of a number of other data resources available on an OGSA-DAI server. An OGSA-DAI deployer can easily configure a server to expose a resource group that clients can then use, e.g. the *SQLBag* activity can be targeted at a resource group that federates N relational resources with the same database schema.

OGSA-DAI 3.0 also provides a *CreateResourceGroup* activity that allows clients, rather than OGSA-DAI deployers, to create and use resource groups. This flexibility means that clients can set up and use their own federations according to their application-specific requirements. In this scenario the client is aware of the individual data resources that make up the group.

6. Improvements to Data Streaming

OGSA-DAI can be used as a workflow processing system that is designed to stream data through a set of activities in a pipelined manner. Take, for example, a workflow consisting of a data producing activity, a data transformation activity and a data delivery activity. If the activities are well-defined then they can each execute concurrently, with each processing a different portion of the data stream. This approach leads to efficient processing of arbitrarily large data with a small memory footprint. The introduction of lists and data grouping provide the foundations for better data handling in OGSA-DAI 3.0. This has been extended to also provide better handling of BLOBs and binary data, and improved asynchronous data delivery.

6.1 BLOBs and Binary Data

The activities shipped with OGSA-DAI have been designed to support binary data in two different representations. Binary data obtained from databases as BLOBs are stored as BLOB objects within tuples and references to the entire BLOB are passed between activities. Binary data obtained from other sources, such as FTP, is streamed through activities as a list of relatively small byte arrays.

It is important to support both mechanisms as they each have their advantages. BLOBs are very useful when it is desirable to keep the binary data grouped with other data elements in a tuple. The list of byte arrays representation better fits the pipeline streaming model of OGSA-DAI as it allows multiple activities to be processing different portions of the binary data stream at the same time. In OGSA-DAI 3.0 all the binary data processing activities have been designed to operate with both these representations of binary data.

6.2 Asynchronous Data Delivery

OGSA-DAI 2.2 supported asynchronous data delivery via OGSA-DAI data services. Input or output streams would be set up and then data could be pushed or pulled to and from these via the data services. One problem with this approach was that the data service and its associated DSR were used to execute workflows as well as transmit or receive data. This meant that the data service was overloaded.

In release 3.0, input streams and output streams have been replaced by data sink and data source resources, respectively. A request can be submitted that creates a data source or a data sink resource. Clients can then interact with these resources using dedicated data source or data sink services which only support interaction with data source and sink resources, rather than via a single overloaded data service. Furthermore WS-ResourceLifetime [10] and WS-ResourceProperties [11] operations can be used to query the state of these resources, via their dedicated services, and to terminate them if required.

This approach yields a more modular set of services, which is in line with the WSRF resources model. This also leads to a more consistent handling of data sources and sinks – they are simply resources, and like data resources or resource groups, they can be created and utilised via activities or dedicated services.

7. Improvements to OGSA-DAI Resources and Services

There have been a number of improvements in OGSA-DAI 3.0 at the level of resources and services.

7.1 Sessions and Requests

All versions of OGSA-DAI have supported requests and recent releases, including 2.2, supported sessions. One problem however was that all interaction and management of the services and requests, including creation, destruction, and updates, was done via a DSR, its supported activities, and its service, which was overloaded.

In OGSA-DAI 3.0, sessions and requests are viewed as types of resource. When a client requests a new session then a *session resource* is created. Activities in OGSA-DAI workflows can then interact with these sessions, using them as repositories to store and retrieve data within the scope of the OGSA-DAI server upon which they are created. Sessions can be used across multiple requests – data or other state stored during one workflow can then be accessed by a subsequent workflow. A session service supporting standard WS-ResourceLifetime operations can be used to manage the lifetime of these session resources.

Likewise every request submitted to a DRER has an associated resource created. A request service supporting standard WS-ResourceProperties and WS-ResourceLifetime operations, and directed at the request resource, can be used to query the progress of the request and to prematurely terminate the request if desired.

By exposing sessions and requests as resources, clients that create and use these have

more control, and also modular services and resources result.

7.2 Requests and Request Status Objects

OGSA-DAI 2.2 and its predecessors were based around the use of XML documents. A data-centric workflow to be executed by an OGSA-DAI service was presented in a *perform document* and the execution status, and often the resulting data itself, were returned in a *response document*. The use of perform documents was central to the activity framework and engine, which form the core of OGSA-DAI.

For the 3.0 release, the document concept has been purged from the OGSA-DAI core – the activity framework and engine now use a Java *request* object and return a Java *request status* object. The conversion to and from XML becomes the responsibility of Web service-based presentation layers. This allows OGSA-DAI core components to be created and used directly, and potentially allows the engine and activity framework to be embedded in other applications.

7.3 Web Services and Workflows

When using the OGSA-DAI 3.0 Web service presentation layers the request and request status are still communicated via XML (SOAP). Clients wishing to use OGSA-DAI construct the XML documents themselves or use the client toolkit. In OGSA-DAI 3.0, the OGSA-DAI Web service operation signatures have been changed so that they fully describe the format of a request or a request status. Accessing the WSDL of a service provides access to an XML representation of a request and a request status. Running an automated code generation tool, such as Apache Axis WSDL2Java, on a 3.0 data request execution service will provide a complete set of objects which can be used to construct a request or traverse a request status in code, avoiding the need for direct XML manipulation. It is also hoped that this change will make the inclusion of OGSA-DAI services in established workflow engines, an interest of such projects as SAW-GEO [12], more straightforward.

7.4 Configuration and Persistence

In OGSA-DAI 2.2, all OGSA-DAI configuration was specified via XML configuration files. In OGSA-DAI 3.0, an explicit persistence layer has been introduced to hide the nature of the persistence media from the core. OGSA-DAI 3.0 supports configuration from simple text files. However, as the persistence layer offers an abstraction layer it is possible for this to be replaced by any other means of specifying configurations, e.g. specifying the configuration via a relational database. At every service operation invocation the persistence layer is consulted and the OGSA-DAI server state is created from persisted state and configuration. This means that configuration can be changed without having to restart OGSA-DAI, e.g. deployers can modify the activities supported by an OGSA-DAI resource or expose new data resources. Consultation of a persistence layer necessarily incurs an additional overhead during request execution. This can be mitigated by either choosing a more efficient persistence media or exploiting caching so that persisted state and configuration is only loaded if it has changed since a previous request. Both the choice of persistence media and the OGSA-DAI components that interact with this are an extensibility point in 3.0 allowing for their transparent replacement.

7.5 Deployment

OGSA-DAI 2.2 and its predecessors supported non-standard deployment of OGSA-DAI into the Apache Tomcat servlet container [13] and the Globus Toolkit [14] container. During deployment all OGSA-DAI binaries and related configuration files were copied into the target container and OGSA-DAI deployers were then required to restart the host container.

OGSA-DAI 3.0 supports WAR-based deployment into Tomcat. This is a standard way of deploying Web applications into Tomcat. One benefit of this is that it removes the need to shutdown Tomcat before deploying OGSA-DAI and its services. OGSA-DAI 3.0 also supports GAR-based deployment into the Globus Toolkit as recommended by Globus.

7.6 Resource Identification

OGSA-DAI 2.2 releases identified targeted resources either by appending the resource ID to the query string part of the URL or by using WS-Resource qualified endpoint references (WS-EPRs). For 3.0, WS-EPRs are used in both Axis and Globus Toolkit compliant versions for consistency. This does not preclude the provision, in future, of a presentation layer which uses a different resource identification specification.

8. Implications for OGSA-DAI Users

The changes we have implemented for OGSA-DAI 3.0 affect different classes of end users in

different ways, but we believe in most cases only moderate adaptation is needed, and this is more than compensated for by the improved functionality and flexibility of the new release.

Developers of clients need to be aware of the changes to the client toolkit APIs as a direct consequence of the changes made to services, resources, and requests on the server-side. The initial release of OGSA-DAI 3.0 provides all the relational database functionality that was present in release 2.2, as well as the new functionality described in this paper. Basic functionality for XML databases and file resources is provided in release 3.0, with the remaining functionality released as part of OGSA-DAI 3.0.1 in Summer 2007.

Developers of application-specific activities experience a major change to the activity APIs. This is mainly a simplification of the APIs due to the standardisation of activity input and output types and support for activity iteration. Those working with application-specific data resources also experience changes to the data resource APIs and the resource configuration, as there has been a decoupling of a resource's configuration and how this configuration is provided and persisted.

Developers of application-specific services experience changes to the APIs that OGSA-DAI services use to access the core OGSA-DAI functionality. The APIs have been cleaned and the boundary between services and the core OGSA-DAI functionality redrawn so that only application-specific code need reside in the service layer.

9. Related Work

OGSA-DAI is not the only solution currently available for data in the Grid space. Storage Resource Broker (SRB) [15] developed by the San Diego Supercomputer Center, provides access to collections of data primarily using attributes or logical names rather than using the data's physical names or locations. SRB is primarily file oriented, although it can also work with various other data object types. OGSA-DAI on the other hand takes a database-oriented approach to its access mechanisms. Although SRB is currently not open source, whilst OGSA-DAI is, a new product called iRods [16] is being developed which will be available under an open source licence.

WebSphere Information Integrator (WSII), a commercial product from IBM, provides data searching capabilities spanning organisational boundaries, a means for federating and replicating data, as well as allowing for data transformations and data event publishing to take place [17]. A detailed comparison between previous versions of OGSA-DAI and WSII can be found in [18].

Mobius [19], developed at Ohio State University, provides a set of tools and services to facilitate the management and sharing of data and metadata in a Grid environment. To expose XML data in Mobius, the data must be described using an XML Schema, which is then shared via their Global Model Exchange. Data can then be accessed by querying the Schema using, for example, XPath. OGSA-DAI, in contrast, does not require an XML Schema to be created for each piece of data, rather, it directly exposes that information (data and metadata/schema) and relies on the resource's intrinsic querying mechanisms to query its data.

These three projects all provide mechanisms to share data across organisational boundaries, however they complement the functionality provided by OGSA-DAI.

10. The Future

The OGSA-DAI project has a number of plans to take release 3.0 forward. At the time of writing these include:

- Releasing additional activities to bring the complement of activities supported by release 3.0 in line with those of the standard activity library specification.
- Investigating persistence and configuration of OGSA-DAI using a relational database.
- Investigating clustering and load-balancing to improve the scalability and fault tolerance of OGSA-DAI via the use of multiple JVMs or even multiple hosts behind an OGSA-DAI server, to increase the number of requests that OGSA-DAI can handle concurrently.
- Continuing investigations into the use of transactions within OGSA-DAI.
- Implementing the WS-DAI specifications [20] using OGSA-DAI, using the new architecture to build a WS-DAI presentation layer that interacts with the OGSA-DAI 3.0 core.
- Incorporating the latest version of the OGSA-DQP [21] distributed query processor components that allow queries to be run in parallel across heterogeneous relational data sources with different table schemas. This version features a new pure Java query compiler which means it will run on

any platform supported by OGSA-DAI.

- Working with the OMII-Europe project [22] to port OGSA-DAI to a number of other platforms, in particular Unicore/GS and gLite.
- Working closely with a selection of specific projects to address their application-specific data access and integration requirements.

11. Conclusions

OGSA-DAI release 3.0 includes a number of major changes which we believe will take OGSA-DAI forward to address the data access and integration requirements of the Grid community. Activities have been simplified and unified to be both more extensible and standard, and to allow workflows to be composed more easily. Workflows themselves have been streamlined to allow more flexibility, in particular to allow multiple data resources to be utilised within a single workflow and to allow different activities to operate upon different parts of a data stream concurrently. Data streaming has been made more efficient, and OGSA-DAI resources and services have also been simplified, increasing modularity and reducing overloading of functionality. While the consequent changes in APIs affects various user classes, in general the APIs have been streamlined and the additional functionality and scalability is seen as a more than acceptable trade-off.

This release provides a powerful piece of software for both data integrators and developers of data applications to build upon. This paper has outlined the major changes that have taken place within OGSA-DAI and which we expect will benefit the e-Science community. The current release of OGSA-DAI is available from http://www.ogsadai.org.uk.

Acknowlegements

The OGSA-DAI project is funded by the EPSRC via the Open Middleware Infrastructure Institute UK. We also gratefully acknowledge the input of our past and present partners and contributors to the OGSA-DAI project including: EPCC, IBM UK, IBM Corporation, NeSC, University of Manchester, University of Newcastle and Oracle UK.

References

[1] OGSA-DAI Project, http://www.ogsadai.org.uk/.

[2] Open Middleware Infrastructure Institute – UK, http://www.omii.ac.uk/.

[3] AstroGrid, http://www.astrogrid.org/.

[4] DGEMap (Developmental Gene Expression Map), http://www.dgemap.org/.

[5] OGSA-DAI – projects using OGSA-DAI, http://www.ogsadai.org.uk/about/projects.php

[6] M. Antonioletti, M.P. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The Design and Implementation of Grid Database Services in OGSA-DAI. Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, Pages 357-376, February 2005.

[7] K. Karasavvas, M. Antonioletti, M.P. Atkinson, N.P. Chue Hong, T. Sugden, A.C. Hume, M. Jackson, A. Krause, and C. Palansuriya. Introduction to OGSA-DAI Services. Lecture Notes in Computer Science, Volume 3458, Pages 1-12, May 2005.

[8] OGSA-DAI 3.0 design documentation, http://www.ogsadai.org.uk/documentation/release3.0/.

[9] Karasavvas, K. Atkinson, M.P. and Hume, A.C. OGSA-DAI – Redesigned and New Activities, http://www.ogsadai.org.uk/documentation/release3.0/.

[10] L. Srinivasan (Ed), T. Banks (Ed). Web Services Resource Lifetime 1.2 (WS-ResourceLifetime), Version 1.2, OASIS Standard, 1 April 2006.

[11] S. Graham (Ed), J. Treadwell (Ed). Web Services Resource Properties 1.2 (WS-ResourceProperties), Version 1.2, OASIS Standard, 1 April 2006.

[12] SAW-GEO, http://www.jisc.ac.uk/whatwedo/programmes/eresearch_grid_ogc_collision/Project_SAW_GEO.aspx.

[13] Apache Tomcat. http://tomcat.apache.org/ .

[14] Globus Toolkit. http://www.globus.org/toolkit/.

[15] Storage Resource Broker (SRB), http://www.sdsc.edu/srb.

[16] iRods, http://irods.sdsc.edu/.

[17] Web Sphere Information Integrator (WSII), http://www.ibm.com/software/data/integration/.

[18] R. O. Sinnott and D. Houghton, Comparison of Data Access and Integration Technologies in the Life Science Domain, Proceedings of the UK e-Science All Hands Meeting 2005, September 2005.

[19] Mobius, http://projectmobius.osu.edu/.

[20] Antonioletti, M., Atkinson, M., Krause, A., Laws, S., Malaika, S., Paton, N. W., Pearson, D. and Riccardi, G., Web Services Data Access and Integration – The Core (WS-DAI) Specification, Version 1.0. GFD-R-P.074, Global Grid Forum. July 2006., http://www.ggf.org/documents/GFD.74.pdf.

[21] Alpdemir, M.N., Arijit M., Gounaris, A., Paton, N.M., Watson, P., Fernandes, A.A.A, and Fizgerald, D.J. OGSA-DQP: A Service for Distributed Querying on the Grid. Editors: E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Boehm, Elena Ferrari. Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004. Lecture Notes in Computer Science, Volume 2992, pp858-861, 2004.

[22] Open Middleware Infrastructure Institute – Europe, http://omii-europe.com/.