Protecting Application Developers – A Client Toolkit for OGSA-DAI

Tom Sugden, Alastair Hume, Mike Jackson, Mario Antonioletti,

Neil Chue Hong, Amy Krause and Martin Westhead

EPCC, University of Edinburgh, James Clerk Maxwell Building,
Mayfield Road, Edinburgh EH9 3JZ, UK.

Abstract

The OGSA-DAI project has developed a client toolkit for the OGSA-DAI middleware. This toolkit comprises of a set of high-level APIs intended to protect application developers from changes in Grid specifications that impact upon OGSA-DAI and remove the requirement to manipulate XML documents either by hand or programmatically. In doing so, the client toolkit aims to improve the usability and shorten the learning curve for developers wishing to write client applications that utilize OGSA-DAI services. This paper describes the underlying motivation for the client toolkit, provides an overview of certain aspects of its design, and presents a number of use-cases to highlight the benefits that are offered.

1. Introduction

The *Open Grid Services Architecture – Data Access and Integration* (OGSA-DAI) project [1] began in February 2002 with the aim of developing a component library for accessing and manipulating data in a Grid for use by the UK and international Grid community. This component library was to provide the middleware glue to interface existing databases and other data resources and tools to each other in a common way based on the Open Grid Services Architecture (OGSA) [2].

The first production release – Release 3 – was made in July 2003 shortly after the *Globus Toolkit 3.0* (GT3) [3] release. Code development was undertaken by two teams, one based at EPCC, the other at IBM UK, with design input being provided by the other project members. This release provided the base services and functionality required to construct higher-level data integration services, such as OGSA-DQP [4]. The DAIT project is now continuing to develop the OGSA-DAI software, concentrating in particular on improving performance and usability, tackling data integration issues, and moving towards compliance with developing standards.

1.1 Shifting Sands

Proposed specifications in the area of Web and Grid services have been in a state of constant flux for some time. OGSA-DAI must try to evolve in tandem with these specifications, complying with emerging standards once they become stable, but balancing this with the desire to provide backwards compatibility for the sake of established users. OGSA-DAI must also evolve in response to changes in the tooling upon which it depends (most notably GT3) that is itself in a state of evolution.

Small changes to specifications and tooling can entail major alterations to the OGSA-DAI code base. Whenever such changes take place, there is a ripple effect propagating from specification, to GT3, to OGSA-DAI, and up to application developers who consequently must alter their code to maintain consistency. Failure to do so may result in applications that rely on obsolete or unsupported versions of OGSA-DAI and so are unable to exploit improvements or inter-operate with higher-level services. In this situation, there exists a risk that application developers may stop using OGSA-DAI and seek a more stable solution.

1.2 Potholes

The OGSA-DAI Grid Data Service (GDS) supports a document interaction model whereby clients express their data access and interaction requirements via an XML document known as a perform document. Whether created by hand or programmatically, these documents can become complex and prone to error, with misspellings, invalid XML and semantically incorrect values being introduced. Errors of these sorts can incur costly implement-test-debug cycles for application developers.

The results of an OGSA-DAI interaction are often expressed in XML and this can require cumbersome DOM manipulation in order to process. These factors and the steep learning curve caused by the dependence on various different concepts, specifications, APIs and tools may deter developers from adopting OGSA-DAI at the outset.
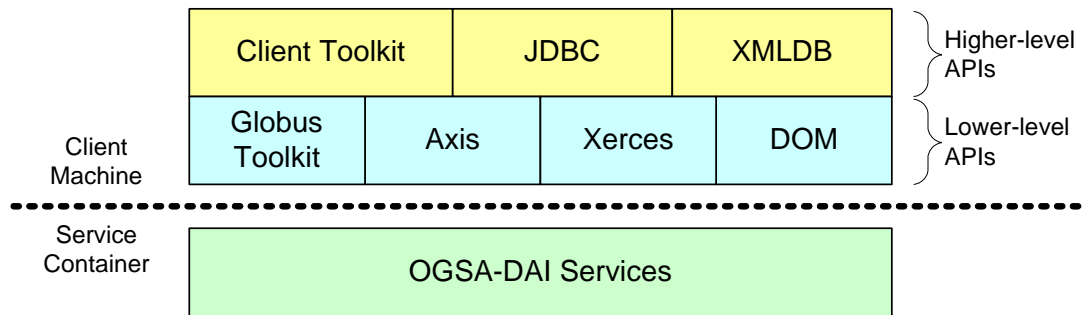
Figure 1: The Client Toolkit provides a higher-level API for interacting with OGSA-DAI services

2. The Client Toolkit

In order to address the problems discussed in the previous section, a client toolkit has been developed for OGSA-DAI. This consists of a higher-level API designed to protect the application developer from the effects of changing standards and tooling, the risk of error introduced by direct XML manipulation, and the complexity of lower-level APIs. A technology preview of the client toolkit was included in OGSA-DAI R3.1 and the first documented release with OGSA-DAI R4.

The client toolkit has three main aims:

- To define a clear and simple API built around solid abstractions, such as the registry, factory, GDS, and activity framework.
- To minimize the specialist knowledge and number of steps required to retrieve results. Ideally the application developer should not be exposed to lower-level APIs such as GT3, Axis [5], Xerces [6] and DOM [7].
- To protect the application developer from future changes to OGSA-DAI interfaces arising from changes to specifications, XML Schema and GT3.

Developers of conventional non-Grid database applications are able to make use of standardised high-level APIs for accessing and interacting with data. For relational databases, the JDBC API for Java provides cross-DBMS connectivity to a wide range of SQL databases, and for XML databases, the XMLDB API serves a similar purpose. Although OGSA-DAI provides additional capabilities to these two standards, such as data transformation and delivery to a 3rd party, the client toolkit API has been defined at a similar level of abstraction. Furthermore, support for the standard JDBC and XMLDB interfaces used for processing results, the JDBC ResultSet and XMLDB ResourceSet,

has been integrated into the client toolkit. Figure 1 illustrates the layering of APIs and shows how the client toolkit sits at a higher-level together with the JDBC and XMLDB APIs, abstracting away details of the lower-level APIs used for interacting with grid services and processing XML data.

Using the client toolkit, a developer can construct objects to represent activities such as a database query and a delivery method and location. These objects can be linked together to form a request that can be passed to an object representing a GDS. Behind the scenes, the client toolkit will compose the corresponding perform document to express the query-and-deliver request, removing one source of error, then send it to the GDS. When the response document is received, the client toolkit will interpret it and extract the results that then become available via simple access methods and standardised interfaces, thus protecting the developer from the complexities of the lower-level APIs.

The client toolkit enables the application developer to interact with OGSA-DAI services at a level closer to their application-specific goals. By avoiding altogether the low-level details of using OGSA-DAI, the client toolkit aims to yield a gentler learning curve for new adopters.

2.1 OGSA-DAI Interactions

This sub-section provides an overview of a typical OGSA-DAI interaction. The concepts and stages involved in such an interaction have guided the design of the client toolkit.

OGSA-DAI defines the following types of grid service:

- DAI Service Group Registry
- Grid Data Service Factory
- Grid Data Service

A DAI Service Group Registry (DAISGR) stores meta-data describing other OGSA-DAI
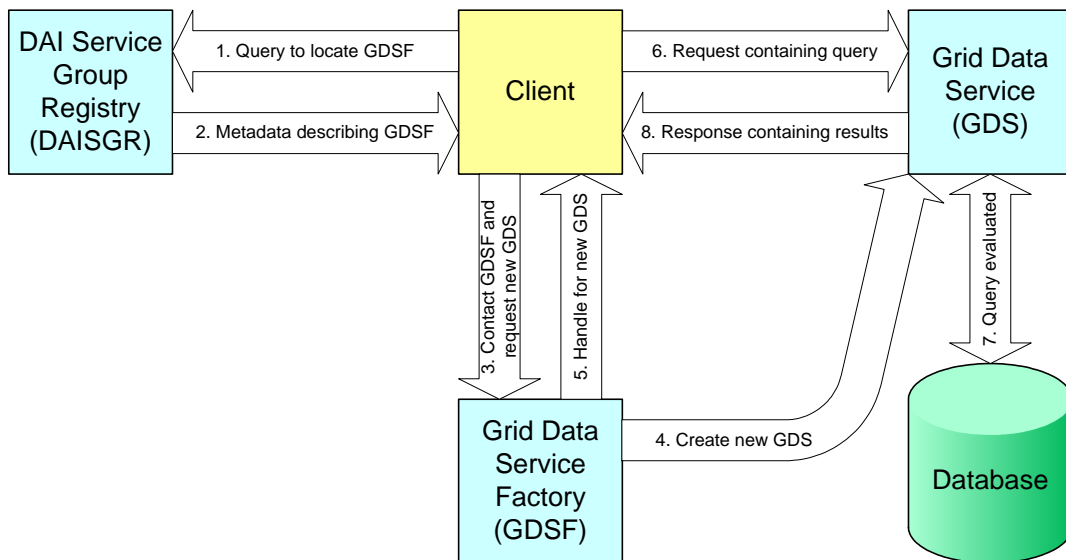
Figure 2: Typical OGSA-DAI client-service interactions

services that clients may access and query. A Grid Data Service Factory (GDSF) represents a single logical data resource such as a relational database. To access a data resource a client must use the appropriate GDSF to create a Grid Data Service (GDS). The GDS then facilitates interactions with the data resource.

The stages involved in a typical OGSA-DAI interaction are illustrated in Figure 2 and summarised below:

1. The client queries a known DAISGR to obtain details of a registered GDSF representing a particular data resource.
2. Metadata is returned to the client describing the capabilities of the GDSF and containing a handle that can be used to access the GDSF.
3. The client accesses the GDSF and instructs it to create a new GDS that will be used by the client for interacting with the data resource.
4. The GDSF creates a new instance of a GDS for the data resource that it represents.
5. The GDSF returns a handle to the client that can be used for accessing the new GDS.
6. The client accesses the GDS and sends a request consisting of a sequence of one or more activities to evaluate.
7. The GDS interprets the request and evaluates the activities described by it. In the case of a query activity, the GDS interacts with the database.
8. The results of the activity evaluation are sent back to the client in the response.

In some cases the client will already know the handle of the GDSF they wish to use and so stages 1 and 2 will be omitted. When the client has no more requests to send to the GDS, a destroy message is issued.

2.2 Services

The client toolkit provides interfaces to represent the three OGSA-DAI service types and a helper class named ServiceFetcher to assist with instantiation. A simplified UML class diagram is shown in Figure 3.

Using these service interfaces and the static methods of ServiceFetcher, a client can access DAISGR, locate GDSF, create and destroy GDS and perform requests without needing to use the Globus Toolkit APIs or perform any DOM manipulation. The UML sequence diagram shown in Figure 4 demonstrates a series of typical client interactions.

Note that the metadata objects returned by the listServices and getActivityMetaData methods provide simple interfaces for accessing information such as the service handle and activities supported by a particular GDSF. However, the definition of standards in the area of metadata is outside the scope of the OGSA-DAI project.

By using a helper class and factory method to separate the construction of ServiceGroup-Registry, GridDataServiceFactory and GridDataService instances from their implementations, the client toolkit protects the client developer from changes to the implementations and dependent technologies.
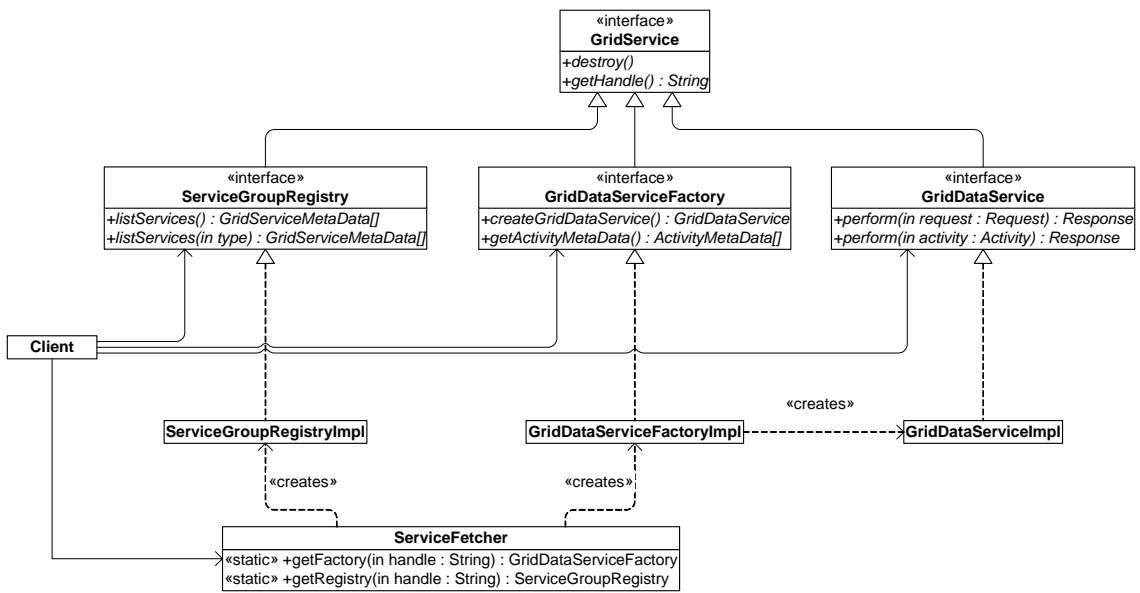
«interface»
GridService
+*destroy()*
+*getHandle() : String*

«interface»
ServiceGroupRegistry
+*listServices() : GridServiceMetaData[]*
+*listServices(in type) : GridServiceMetaData[]*

«interface»
GridDataServiceFactory
+*createGridDataService() : GridDataService*
+*getActivityMetaData() : ActivityMetaData[]*

«interface»
GridDataService
+*perform(in request : Request) : Response*
+*perform(in activity : Activity) : Response*

Client

ServiceGroupRegistryImpl

GridDataServiceFactoryImpl

«creates»

GridDataServiceImpl

«creates»

«creates»

ServiceFetcher
«static» +getFactory(in handle : String) : GridDataServiceFactory
«static» +getRegistry(in handle : String) : ServiceGroupRegistry

Figure 3: Class diagram showing OGSA-DAI services and ServiceFetcher

:Client

:ServiceFetcher

getRegistry(handle)

<<create>>

reg:ServiceGroupRegistry

reg

listServices(factoryType)

metadata containing factory handles

getFactory(handle)

<<create>>

fac:GridDataServiceFactory

fac

createGridDataService

<<create>>

gds:GridDataService

gds

process requests using gds

destroy
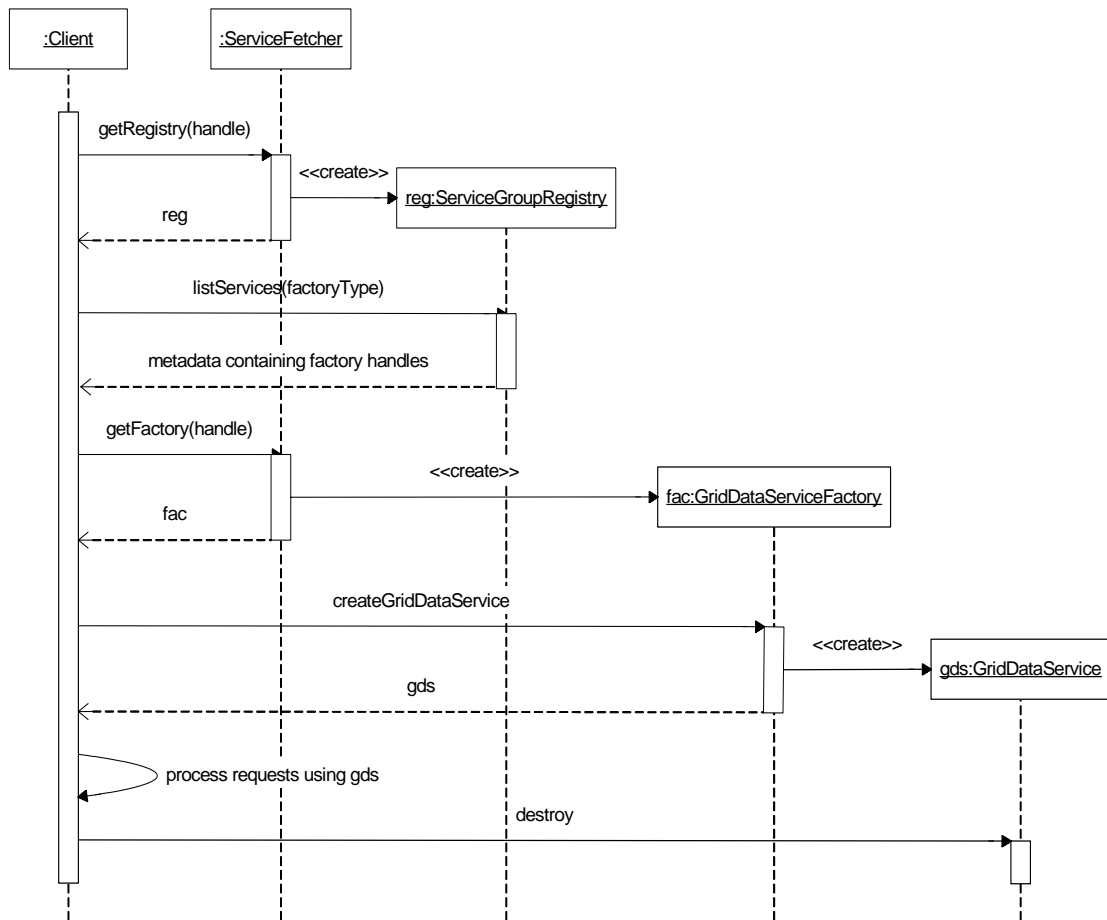
Figure 4: Sequence diagram demonstrating service access and creation

Activity

ActivityOutput
+getName() : String
+getBytes() : byte[]

Request

SQLQuery
+SQLQueryActivity()
+SQLQueryActivity(in expression : String)
+getOutput() : ActivityOutput
+getResultSet() : ResultSet
+setExpression(in expression : String)

XSLTransform
+XSLTransform()
+XSLTransform(in transform : String, in data : ActivityOutput)
+getOutput() : ActivityOutput
+setInput(in data : ActivityOutput)
+setTransform(in transform : String)

ActivityRequest
+ActivityRequest()
+ActivityRequest(in activities : Activity[])
+addActivity(in activity)
+addActivities(in activities : Activity[])
+clear()

Figure 5: Class diagram showing activities and activity request

:Client   query:SQLQuery   tran:XSLTransform   req:ActivityRequest   :GridDataService

getOutput
output
setInput(output)
Connect activities
addActivity(query)
Assemble request
addActivity(tran)
perform(req)
response
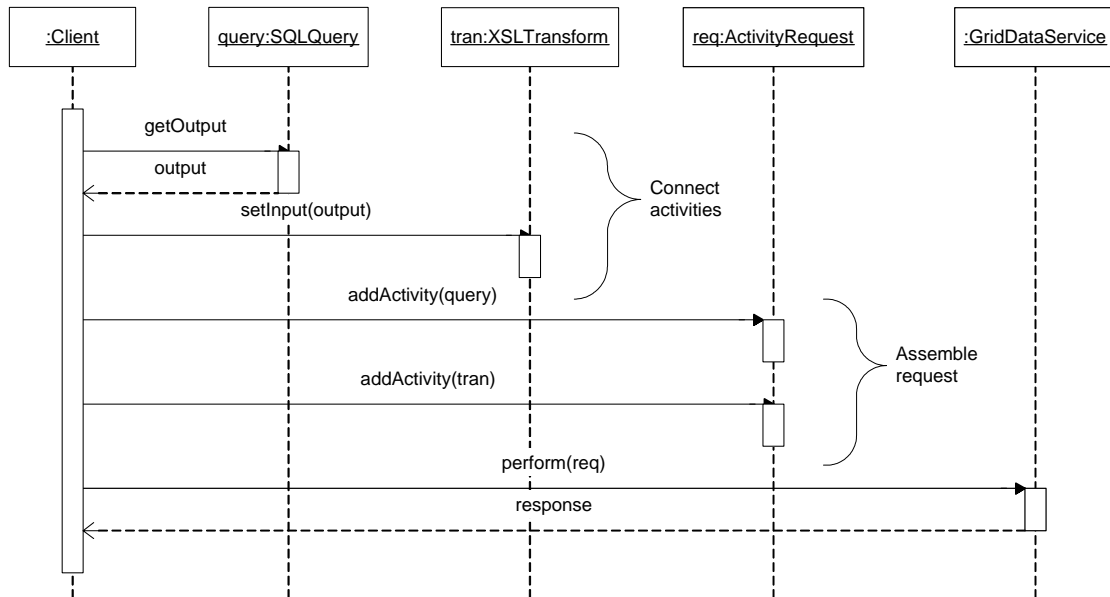
Figure 6: Sequence diagram demonstrating how to connect activities and assemble a request

2.3 Activities and Requests

The concept of an activity is central to OGSA-DAI and the client toolkit. An activity dictates an action to be performed the GDS. Broadly speaking, activities fall into four categories – access, update, transformation and delivery – but the activity framework is an extensibility point that enables OGSA-DAI service providers to develop and deploy their own activities to perform arbitrary tasks.

An activity can have multiple inputs and outputs, and the output from one activity can be connected to an input of another to form an activity chain. In order to process a chain of activities, the client toolkit provides an ActivityRequest object that can be assembled using the addActivity method before being sent to the GridDataService perform method. The UML class diagram in Figure 5 shows the ActivityRequest class and two activity implementations. The UML sequence diagram in Figure 6 demonstrates how an SQLActivity and an XSLTransform activity can be constructed by the client, chained together so that the output of the query will be transformed, assembled into an ActivityRequest and then sent to a GridDataService via the perform method. Note from the class diagram that the SQLActivity implementation also provides a getResultSet method to access query results using the standard java.sql.ResultSet interface. This allows an OGSA-DAI client application to process query results in the same manner as a conventional non-Grid application.

OGSA-DAI R4 includes a large number of useful activities for accessing and updating relational and XML data, performing data transformations and compressions, and delivering data via various protocols. The client toolkit enables developers to construct and connect these activities programmatically in order to achieve anything from a simple database query to a complex distributed data integration scenario with relative simplicity.

3. Use Cases

This section presents two use cases to illustrate the difference between OGSA-DAI client development with and without the client toolkit. A simple metric of lines of code required when limited to 80 characters in length has been used to assess the usability benefits. The number of import statements required has been used as another measure of complexity.

3.2 Processing Query Result

This use case considers a client application that must create a GDS, query a relational database table, and display the first x entries. The relational table has the following structure:

LittleBlackBook	
PK	<u>ID</u>
	Name Address Phone

Figure 7: Table structure

The bar graph in Figure 8 shows the number of lines of code and import statements required to implement this client using the client toolkit compared with using the lower-level APIs:
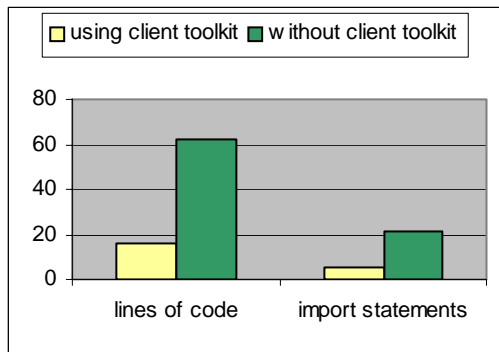

Figure 8: Processing query results with and without the client toolkit

It can be seen the client toolkit has reduced the number of lines of code required by almost ¾ from 62 to 16. The number of import statements required has been reduced by a similar factor, from 21 to 5. This is largely due to the wrapping of DOM and ExtensibilityType (an OGSI class) manipulation, and the default service termination times provided by the client toolkit.
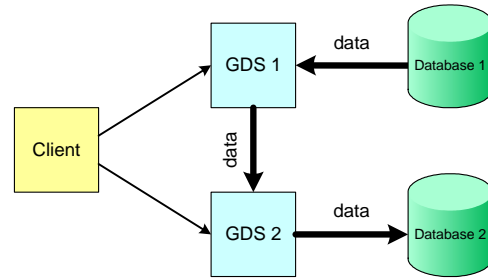
3.2 Data-replication

Figure 9: Replicating data between 2 GDS

This use case considers a more complex scenario of data replication between two remote GDS. The client must query a relational database table accessed through first GDS, then deliver the results to the second GDS, where they are imported into an existing table, as shown in Figure 9. This scenario is similar to that used by both the *FirstDIG* [8] and *EdSkyQuery-G* [9] data integration projects.

The bar graph in Figure 10 shows the number of lines of code and import statements required with and without the client toolkit:
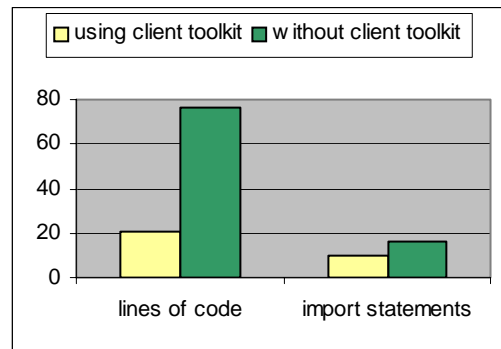

Figure 10: Replicating data between 2 GDS with and without the client toolkit

In this scenario, the client toolkit has again offered a benefit, reducing the number of lines of code from 76 to 21, and the import statements from 16 to 10. The gain is slightly less than the previous use case because there is no need to manipulate response documents or process results in this scenario, all processing being performed by the services.

As well as reducing code complexity, clients built using the client toolkit are protected from changes to the service WSDL interfaces, perform document and activity schemas, and dependent tooling. So long as the client toolkit APIs remain stable, these clients can be updated to make use of newer versions of OGSA-DAI without requiring any code modifications.

4. Conclusions

The two use cases that were analysed in the previous section indicate that the first release of the client toolkit has achieved its design objectives. Although concepts of clarity and simplicity are rather subjective, the measurements of code complexity that were taken show that the client toolkit provides a more straightforward API for interacting with OGSA-DAI services than was previously available. The reduced number of import statements that were required indicates that lower-level APIs such as GT3, Apache Axis, Apache Xerces, and DOM have been successfully abstracted away from the developer.

The third aim of protecting developers from future changes to specifications, schema and dependent tooling is more difficult to qualify. Although the client toolkit provides looser coupling to underlying technologies and removes the need to write and manipulate schema validated XML, the decision to build upon the concept of the factory and grid data service may have been unnecessary. In fact, it has been suggested that the client developer need only be concerned with the structure and capabilities of the underlying data resource. The current process of contacting a factory in order to instantiate a grid data service is extraneous and could be avoided if the client toolkit was to hide all factory interactions. This would simplify the API and loosen coupling to the service architecture, which is likely to change during the transition from OGSI to WS-RF compliance planned for release 5.

For the client toolkit to truly protect client developers from future changes it will be necessary for the OGSA-DAI project to adopt a deprecation policy similar to that used by Sun for Java. When changes to the client toolkit APIs are introduced, any superseded methods must enter a deprecation phase during which they are preserved for a certain number of releases before being removed. Any time a method is deprecated notification must be sent to the OGSA-DAI users mailing list providing details and alternatives. This approach would foster good relations with our developer community and increase their confidence in writing programs based on OGSA-DAI by assuring them of sufficient time and information to update their code whenever changes are made.

The areas of metadata and service-data have not yet been addressed adequately by the client toolkit. Once a metadata standard has emerged for describing data resources, the registry service will become useful for locating particular data resources and the client toolkit should be extended to better expose this information. A simple interface is also needed for accessing and subscribing to service data elements (the "state" of grid services), perhaps conforming to the Observer [10] design pattern.

In summary, the client toolkit improves the usability and shortens the learning curve for OGSA-DAI. As the project continues, a suitable deprecation policy will be needed to protect developers while the APIs are extended and improved to provide a higher-level abstraction of the service architecture and improved exposure of metadata and service-data.

References

[1] OGSA-DAI Project
    http://www.ogsadai.org.uk
[2] Open Grid Services Architecture (OGSA)
    http://www.globus.org/ogsa/
[3] Globus Project
    http://www.globus.org
[4] DQP Project
    http://www.ogsadai.org.uk/dqp
[5] Apache Axis
    http://ws.apache.org/axis/
[6] Apache Xerces
    http://xml.apache.org/xerces2-j/
[7] Document Object Model (DOM)
    http://www.w3.org/DOM/
[8] FirstDIG Project
    http://www.epcc.ed.ac.uk/~firstdig
[9] EdSkyQuery-G Project
    http://edskyquery.forge.nesc.ac.uk
[10] Gamma, E., R. Helm, R. Johnson, J. Vlissides, *Design Patterns*, Addison-Wesley (1995)