

School of Computing Science,
University of Newcastle upon Tyne



A Grid Application Framework based on Web Services Specifications and Practices

Savas Parastatidis, Jim Webber, Paul Watson,
and Thomas Rischbeck

Technical Report Series

CS-TR-825

January 2004

Copyright©2004 University of Newcastle upon Tyne
Published by the University of Newcastle upon Tyne,
School of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK.



A Grid Application Framework based on Web Services Specifications and Practices

Authors:

Savas Parastatidis
Jim Webber
Paul Watson
Thomas Rischbeck

Copyright © 2003
North East Regional e-Science Centre
School of Computing Science
University of Newcastle, Newcastle-upon-Tyne, NE1 7RU
United Kingdom

A Grid Application Framework based on Web Services Specifications and Practices

Document version: 1.0

Date published: 12 August 2003

Errata

4 September 2003 – Typos corrected

10 September 2003 – Douglas Purdy's reference corrected

The latest version of this document and supporting material can be found at
<http://www.neresc.ac.uk/projects/gaf>

Abstract

There has been a lot of discussion in the community concerning the relationship between Web and Grid Services. This document presents an analysis of the Grid architecture and its relationship to the Web Services Architecture.

It presents our views on the current version of the OGSF specification and then goes on to propose a different mapping of the Grid infrastructure requirements to Web Services specifications and practices. The aim is to design an alternative mapping that captures the same Grid requirements, but fits more closely with existing Web Services specifications and practices.

This document is work in progress and we actively encourage feedback from the community.

Author contact information:

Savas Parastatidis (Savas.Parastatidis@newcastle.ac.uk)¹

Jim Webber (Jim.Webber@arjuna.com)²

Paul Watson (Paul.Watson@newcastle.ac.uk)¹

Thomas Rischbeck (Thomas.Rischbeck@arjuna.com)²

¹ School of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne, NE1 7RU
United Kingdom

² Arjuna Technologies Ltd.
Nanotechnology Centre, Herschel Building
Newcastle upon Tyne, NE1 7RU
United Kingdom

Contents

1. Introduction	1
2. Grid Requirements and OGSi Solutions	1
2.1. Grid Requirements.....	2
2.2. Service-Oriented vs Object-Oriented for the Grid Infrastructure	2
2.3. Exposure of Service State and the Web Services Architecture	3
2.4. Service Lifetime Management and the Web Services Architecture	3
2.5. Divergence from Web Services Technologies and Tools.....	3
2.6. Overloaded Semantics for Grid Service Instances	4
2.7. OGSi and Low-Level Networking.....	4
2.8. Factorisation	4
2.9. OGSi and Other Web Services Specifications	4
3. WSA-based Solutions for the Grid Application Framework.....	5
3.1. Stateful interactions	5
3.1.1. Stateful Interactions in OGSi	5
3.1.2. Contexts and Activities	6
3.2. Data References	8
3.2.1. Data Exposed Through Grid Service Instances	8
3.2.2. Data as Structured Documents.....	9
3.3. Support for Services with “Publicly Visible State”	11
3.3.1. The OGSi Service Data Elements	11
3.3.2. Attributes in WSDL 1.2.....	11
3.4. Service Deployment and Management.....	11
3.4.1. Issues with Dynamic Creation of Grid Service Instances on a Different Administration Domain.....	12
3.4.2. A High Level Service for Remote Service Deployment.....	13
3.5. Lifetime in the Grid Application Framework.....	14
4. Proposed Architecture	15
5. Applying Existing OGSA Concepts: Examples	16
5.1. Stateful Interactions – The Counter Service Example.....	16
5.2. Datarefs – Data-oriented Service	18
5.2.1. The Service	18
5.2.2. Usage Patterns	18
5.2.3. Retrieving Referenced Data.....	20
5.2.4. Data-related Registries	21
6. Conclusions	21
7. Acknowledgements	23
8. References	23

1. Introduction

The Open Grid Services Infrastructure (OGSI) [1] specification defines the fundamental concepts and characteristics of Grid Services. It introduces the concept of a Grid Service Instance, which is “a Web Service that conforms to a set of conventions (interfaces and behaviours)” [1] and adds features to Web Services including statefulness; stateful interactions; the ability to create new instances; service lifetime management; notification of state changes; and, service groups. It is these additional features that define the semantics of Grid Service Instances, the foundation of the Grid infrastructure.

The reason that the Grid community chose to base the OGSI framework on Web Services was so that those building Grid applications could leverage the huge investments being made in the much larger wider Web Services world, including tools, specifications, services, and educational materials.

The conventions embodied in Grid Services were designed to meet the perceived requirements of Grid applications. OGSI offers one particular solution to meeting these requirements through defining a particular mapping of the Grid concepts onto Web Services. This document investigates the current OGSI specification, and its relationship to common Web Services concepts and practices. It argues that OGSI diverges from some key, common Web Service practices, particularly in the important area of dealing with stateful service interactions. The document proposes an alternative mapping of Grid concepts onto Web services that aims to capture the same requirements as OGSI but in a way that is more natural to the Web Services architectural model and does not deviate from Web Services specifications. In many cases, this is made much more straightforward by the fact that the Web Services community is working on providing solutions—in the form of specifications and tools—to similar requirements that the Grid community is addressing. We believe that a close relationship between the two communities is important in allowing those building Grid Services to leverage the much larger investment in Web Services tools, specifications, and services.

The main incentive behind this work is to start discussions within the Grid community on the future of OGSI and its relation to Web Services specifications and the Services Oriented Architecture (SOA). It aims to show how the OGSI model could directly leverage Web Services technologies to fulfil its requirements. We view the Grid infrastructure as an application framework that sits on top of existing and emerging Web Services technologies, concepts, and practices. This promotes the coexistence of Grid Services with non-infrastructure specific specifications, such as BPEL [2].

In proposing this particular Web Services Framework for the Grid, we used as input the discussions and decisions of the OGSI working group [3] and the identified requirements for the Grid infrastructure. We assume that the reader has a basic knowledge of OGSI [1].

The rest of the document is structured as follows. Section 2 presents the requirements for a Grid Application Framework and analyses how OGSI addresses them. Then, in Section 3, an alternative approach, based on the Web Services Architecture (WSA) [4], is presented and compared with OGSI. In Section 4, the architecture for the proposed Grid Application Framework is introduced. Section 5 gives examples of the ways the proposed Grid Application Framework can be used to build Grid applications or OGSA high level services. Finally, the overall conclusions are presented in Section 6.

2. Grid Requirements and OGSI Solutions

In this section we summarise the key requirements for Grid Services, and the issues that exist with the current version of the OGSI specification.

2.1. Grid Requirements

The key requirements for a Grid Application Framework which are crystallized in the current OGSi specification are to provide consistent mechanisms for:

- Stateful interactions between consumers and services
- Exposure of a web service's "publicly visible state"
- Access to (possibly large amounts of) identifiable data
- Service lifetime management

In the rest of this section we examine what we believe to be issues in the way in which the current OGSi specification addresses these requirements.

2.2. Service-Oriented vs Object-Oriented for the Grid Infrastructure

It is the fundamental tenet of this whitepaper that the Grid should be an application framework built on the Service Oriented Architecture (SOA) model, the concepts of which are realised by the Web Services Architecture (WSA) [4]. Essential to this is the *service*, which can be defined as follows:

A service is a well-defined set of actions, it is self-contained, stateless, and does not depend on the state of other services. (The definition is consistent with the definitions found in [4] and [5])

Here, stateless means that each time a consumer interacts with a Web Service, an action is performed. After the results of the service invocation have been returned, the action is finished. There is no assumption that subsequent invocations are associated with prior ones.

"The description of a service in a SOA is essentially a description of the messages that are exchanged. This architecture adds the constraint of stateless connections, that is where all the data for a given request must be in the request." [4]

Any application framework built on the SOA concepts should not attempt to change the defined behaviours and semantics of a service - a Web Service in the case of WSA. The mere use of WSDL [6] and SOAP [7] does not constitute an implementation of the SOA concepts.

It is our view that the current OGSi specification builds an object-oriented application framework around the concept of the Grid Service Instance, which is analogous to an object with a well-defined interface and private state. Furthermore, the use of Grid Service Handles (GSHs) as identifiers of Grid Service Instances resembles the use of pointers that provide access to objects. OGSi gives semantics to Web Services like statefulness and transience, closely following the CORBA object model.

We believe that object-oriented infrastructures for building distributed applications are more suitable for closed systems since they encourage tight integration of distributed components (e.g., through the creation of complex webs of references between them). The loosely-coupled nature of the Organisation to Organisation (O2O) Grid applications suggests that the infrastructure based on SOA and realised by WSA is more appropriate since the paradigm more closely matches the expected deployment scenarios than does object-orientation.

"Service Oriented Architecture and Web Services enable loose coupling, but do not guarantee it" Douglas Purdy, Microsoft Corporation [8]

Dynamically created service instances with lifetime characteristics not only deviate from the loosely coupled nature of WSA, but encourage a mode of thinking that closely resembles object-orientation. Under such a scheme developers may not realise the potential pitfalls in creating and invoking service instances in a fine-grained fashion since this is precisely the mental model that object-oriented developers are used to. While we understand that such abuse can be avoided with clever design, we believe that the notion of service instances inevitably corrals developers into an object mindset, to the detriment of their applications. This may lead to tight coupling of

distributed components and the development of fragile and non-scalable Grid applications as consumers and services are allowed to bind directly to transient instances, creating dependencies which cross organizational boundaries.

In contrast, as will be shown in section 3.1, we believe that there are existing Web Service technologies which can meet the identified requirements, while remaining consistent with the SOA concepts and not diverging from current Web Service practices and specifications.

2.3. Exposure of Service State and the Web Services Architecture

One of the requirements for the Grid infrastructure given above was “Exposure of a web service’s publicly visible state.” It is our belief that this requirement diverges from the current Web Services Architecture (WSA) where there is no concept of “a service’s publicly visible state.” Further, the scheme breaks the definition of a service given in the previous section since it is possible for other services to refer and create dependencies to the exposed state.

With OGSi it is easy to overuse the concept of a service instance to expose state and the references to it. This may lead to loss of encapsulation and tight coupling, which diverges from the SOA concepts and the best practices in Web Services.

Although we do not adopt the concept of a “service’s publicly visible state,” we do recognise the need to uniquely identify data resources and provide access to them when building data-oriented Grid applications. This document proposes a solution that is consistent with SOA and does not run contrary to current Web Service practices.

2.4. Service Lifetime Management and Dynamic Service Deployment

OGSi adds service lifetime management functionality to Grid Service Instances in order to support their (potentially) transient nature. We believe that services should not be treated as transient entities that can be created on demand as such behaviour would not be consistent with current Web Service practices. Also, as explained in Section 2.2, the ability to dynamically create transient service instances may result in tightly-coupled applications.

The dynamic nature of Grid Service Instances and the use of GSHs to provide location transparency give the false impression to the Grid community that dynamic instantiation of services across administrative domains is easy. We believe that the basic, underlying Grid infrastructure cannot—and indeed does not attempt to—deal with all the practicalities of deploying binary code across organisations, including all the security and policy issues that this raises.

However, we do not rule out scenarios where the dynamic deployment of services is necessary. Indeed, the Web Services community is introducing tools to support this kind of functionality. For example, current implementations of the BPEL specification [2] (i.e., IBM’s BPWS4J [9] and Collaxa Orchestration Server [10]) offer deployment interfaces for Web Services written in BPEL. We believe that the emphasis should be directed at the specification of such high-level interfaces rather than on the creation and destruction of transient service instances at the infrastructure level.

2.5. Divergence from Web Services Technologies and Tools

OGSi introduces extensions, such as Service Data Elements (SDEs),¹ to the underlying Web Services Description Language (WSDL) [6], that diverge from common practices. This means that a suite of Grid Service-specific tools is required, both at the service and consumer ends. If no extensions were introduced then it would be possible to use existing, unmodified industry and open-source tools for the development and deployment of a Web Services Framework for Grid computing.

¹ Section 3.3 (page 1) further expands on SDEs as the means to expose the “publicly visible state” of a Grid Service Instance.

2.6. Overloaded Semantics for Grid Service Instances

We believe that the use of a Grid Service Instance is overloaded as it attempts to capture a number of orthogonal requirements; it is used for stateful interactions, for providing access to data through operations, and for representing transient state. Additionally, a Grid Service Handle can be seen as an identifier of a particular stateful interaction, a “pointer” to an object encapsulating data and a way of providing access to the data through operations, and a handle to transient state. Such overloading forces an all-or-nothing approach that mixes several logically separate semantic concerns. This approach is unlike the modular mix-and-match style favoured by Web Services proponents.

2.7. OGSi and Low-Level Networking

We believe that the current OGSi specification focuses too much on the details of the underlying network infrastructure and, in some cases, extends that infrastructure in a proprietary fashion. For example, OGSi adds stateful interaction semantics to Grid Service Instances; it introduces the concepts of Grid Service Handle (GSH) and Grid Service Reference (GSR); it adds transience semantics; it augments WSDL in order to add previously non-existent interface elements, etc.

We argue that the Grid community should concentrate on building the higher level services that will realise the vision of Grid computing [11-15]. We believe that the Grid community can benefit from adopting Web Services technologies wholesale, thus freeing resources to concentrate on the domain-specific aspects of the Grid, and delegating the work of creating XML-based network protocols to the Web Services community.

2.8. Factorisation

In addition to the basic behaviours for Grid Service Instances, the OGSi specification defines interfaces for handle resolution,² factory,³ registry, and notification services. We argue that registry and notification should have been high level services built on top of OGSi and not part of it. Despite the fact that such interfaces are optional, incorporating them into OGSi, places greater demands on the developer of a Grid service, while such functionality could be subsumed by the higher-level services, perhaps forming the core for the OGSA specification. For example, the definition of specialised registries and discovery mechanisms (e.g., P2P, UDDI-like federation, etc.) and their interfaces should, we believe, be deferred to other specifications from the Web or Grid Services communities. The same should be the case for services offering notification. Our argument is that it is not possible to predict all the requirements of the high level services or their usage patterns when architecting an underlying application framework for the Grid.

2.9. OGSi and Other Web Services Specifications

Due to their additional semantics Grid Service Instances cannot coexist without problems with some of the existing Web Services specifications. For example, the Business Process Execution Language for Web Services (BPEL) [2] allows the composition of existing Web Services into a business process which can be deployed and used as a new Web Service. It is not obvious how a Grid Service Instance using BPEL could be written due to the additional OGSi-defined semantics that must be supported. Changes to the BPEL specifications and existing tools may be necessary, or application level workarounds implemented.

Also, the Web Services community is meeting the requirement for contextualised interactions, a subset of which are stateful or session-based interactions, using implicit context propagation between services (e.g., WS-Coordination [16], WS-Transaction [17], OASIS BTP [18], OASIS WS-Security [19], OASIS WS-ReliableMessaging [20], WS-CAF⁴ [21]). We believe that contextualised interactions are more flexible than the Grid Service Instance concept, and are consistent with the

² Grid Service Handle (GSH) to Grid Service Reference (GSR) resolution.

³ Dynamic creation of Grid Service Instances.

⁴ WS-CAF is a suite of three specifications for dealing with composite Web Services-based applications. It was released by Arjuna Technologies, Fujitsu, Iona, Oracle, and Sun.

direction in which the wider Web Services community is moving. The proposal in Section 3 adopts WS-Context [22], which is part of the WS-CAF [21] suite of specifications. Since the use of WS-Context is not intrusive to the Web Services Architecture – it is SOAP-native, utilising the header extensibility mechanism, rather than creating new entities - its coexistence with other protocols is assured.

3. WSA-based Solutions for the Grid Application Framework

This section contains more details of how OGSF meets the requirements given in Section 2.1, and then examines how the same requirements may be met in a way that is more consistent with Web Service specifications and practices.

3.1. Stateful interactions

Recall from Section 2.2 that by default Web Services are stateless entities and represent a set of actions that can be requested through message exchange. That is, each time a consumer interacts with a Web Service,⁵ an action is performed, completed, and a message may be returned. The Grid, along with other Web Service applications, possesses the need for stateful or session-based interactions between consumers and services. In OGSF, this is handled by the dynamic creation of Grid Service Instances, identified by Grid Service Handles (GSHs), which can maintain state between invocations.

We now identify issues with the current OGSF solution, before proposing an alternative based on a Web Services context specification.

3.1.1. Stateful Interactions in OGSF

A Grid Service Instance that is created to maintain interaction-specific state (or, session-based information) is identified by a Grid Service Handle (GSH), which has to be resolved to a Grid Service Reference (GSR) before the identified service instance can be accessed. We highlight the following problems with this approach:

- If more than one consumer or service is to participate in the same logical unit of work (interaction), a GSH or a GSR of the service instance representing that unit of work must be shared explicitly.
- The OGSF specification allows more than one GSH to identify the same Grid Service Instance. This may cause problems when consumers and services use just GSHs to identify the logically shared unit of work in which they participate. For example, if many GSHs point to the same Grid Service Instance, which happens to represent a stateful interaction between a number of consumers and that service instance, care must be taken to share only one of the GSHs amongst the participants in the unit of work. Otherwise, a consumer would have to go to great lengths to check whether two GSHs refer to the same Grid Service Instance (i.e., the same logical unit of work).
- A GSH is just a URI [23]. It is created as a name for a Grid Service Instance and, as such, it cannot carry any additional information about the stateful interaction that the service instance represents. The same applies to GSRs, which carry endpoint-specific information for accessing the referenced service instance, along with lifetime constraints. This suggests that if participants in stateful interactions require access to interaction-specific information, they have to go through the GSH to GSR resolution process and communicate with the referred Grid Service Instance. Information about a stateful interaction has to be maintained by a service at all times.
- If two or more units of work (or interactions) are to be associated with each other in an application-specific manner, additional services (perhaps registries) are necessary because there is no way for GSHs or GSRs to be related to each other without some out-of-band mechanism.

⁵ From now on we use the terms *Web Service* and *service* interchangeably.

3.1.2. Contexts and Activities

Grid services are not the only network services which require stateful interactions—Web Services applications often exhibit this requirement. However, the way in which Web Services-based applications have tackled this requirement is substantially different (e.g. WS-Coordination [16], WS-Transaction [17], OASIS BTP [18], OASIS WS-Security [19], OASIS WS-ReliableMessaging [20], WS-CAF [21]) from the way in which OGSF meets it using service instances.

In a true service oriented architecture, there are no “instances” of a service as are used in the OGSF model. Instead, a service is a stateless entity which exposes some functionality to the network. The lifecycles of services are determined by the service owner, and a service can be used by many consumers at any one time. This contrasts to the OGSF model where the consumer is encouraged to create service instances for which (typically) is the only user.

However even in a stateless Web Services environment it is often the case that a sequence of operations invoked on a service need to be logically related such that the results of one operation can be used as the basis for others. In order to achieve this in a way which is complimentary to the service oriented architecture, Web Services consumers *contextualise* their operations. What this means is that a consumer will embed (typically in a SOAP header block) a context identifier which is understood by the service, and which the service can use (in combination with the application payload of the message) to re-establish the correct service state for this interaction.

A typical example of this in the Web Services arena is the WS-Transaction [17], BTP [18], or WS-TXM [24] protocols where transaction contexts are used to establish the scope of a transaction within which an operation (or, more usually, operations) is (are) executed.

The primary drawback with this approach is that, until now, there had been no standard means for creating contexts, which meant that any previous context-based schemes would be unlikely to interoperate. However, with the recent release of the WS-Context specification [22], the Web Services community has provided the basic building blocks for creating and managing *activities* over Web Services. An activity is simply a (distributed) unit of work, and may be in one of three states: active, completing or completed. The concrete nature of an activity, which is actually an abstract entity, is only defined by the application in which it is used. At the message level, activity contexts are embedded into a SOAP header block, thus contextualising the message and providing the recipient of the message with additional metadata with which to execute an operation in response to the receipt of the message.

Activity contexts support stateful interactions with Web Services. Context semantics are a superset of the instance-based approach semantics exemplified by OGSF, but are sympathetic to WSA since they are SOAP-native, using only the extensible header mechanisms from SOAP. This is shown in the example of Figure 1:

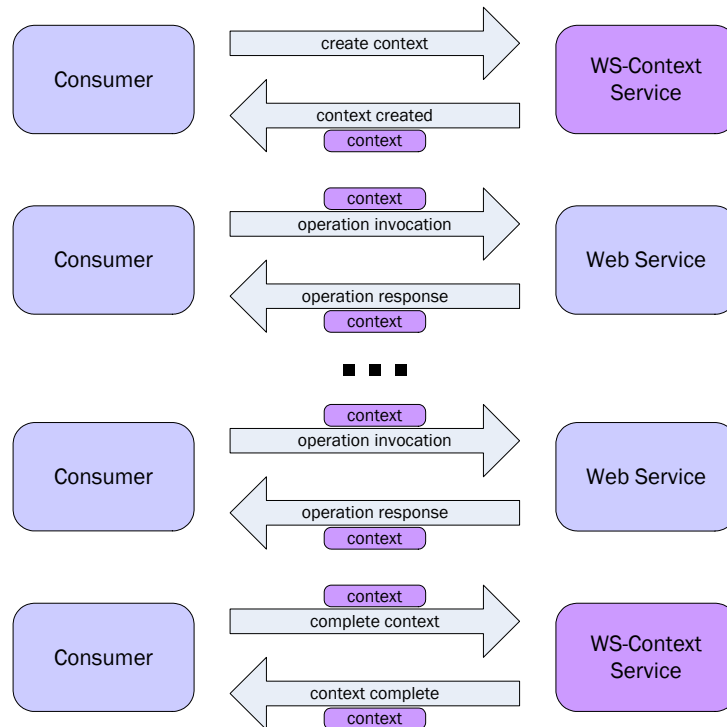


Figure 1: Stateful Interaction with a Single Web Service using WS-Context

In this example, the consumer initiates an activity and is given a unique context by the WS-Context service. This context is then piggy-backed on each invocation on the consumer's target Web Service in a SOAP header block. The Web Service receiving the invocation from the consumer can use the context and application payload to access any state that the invocation requires and hence provide a stateful interaction.

Although a WS-Context service is used to create new context, it is possible for a Web Service to provide a context automatically based on some application-specific semantics (e.g., a WS-Context could be created automatically as a result of a *newDatabaseSession* operation on a Data-oriented Web Service).

In the simple case of a single consumer to service interaction, the Grid Service Instance approach to reasoning about the state of that interaction is equivalent to the context approach. However, unlike Grid Service Instances, the activity context-based approach can extend past simple consumer-service instance interactions, and can be used to implicitly group work across arbitrary services where, for instance, a consumer may interact with multiple services within the context of a single activity. In OGSI, such dynamic composition of arbitrary services in one activity requires explicit management of GSHs, which must be shared amongst the activity participants. Also, it is not possible to add further, activity-related information to the GSH once it is created. A context, on the other hand, may be dynamically augmented with additional application or interaction specific information while it is implicitly transferred between participants. Finally, any service invoked within the scope of an activity may propagate the activity context onto any other services which it invokes at the back-end.⁶

For additional flexibility, activities can be nested to allow hierarchies of related work to be built, and no constraints are placed on which of the parties involved in an activity are allowed to end the lifecycle of the activity.

Specifications that address non-functional requirements, like WS-Coordination [16], WS-CF [25], WS-Transaction [17], WS-TXM [24], BTP [18], OASIS WS-Security [19], OASIS WS-ReliableMessaging [20] are very important for Web Services and Grid applications. Since all these

⁶ It has not escaped our attention that basing our approach on SOAP-native constructs like WS-Context permits dynamic adaptation for security, transactionality, and so forth using the same common mechanisms. This cannot be easily implemented with the current GSH/GSR-based system.

specifications depend on contextualised interactions, we anticipate the use of context to be pervasive in the Grid.

While the activity context-based scheme provides more flexibility than the current instance-based approach, that flexibility comes at little or no additional cost to the developer. For instance, consider the pseudo Java code of Listing 1 where an axis-like tool can abstract the use of contextualised interactions for a particular service.

```
// Get a proxy from an Axis-like service locator
MyService myService = myServiceLocator.getContextualisedService();
myService.someOperation(...);
```

Listing 1: Pseudo Code Demonstrating a Simple Consumer-Service Contextualised Interaction

A slightly more complicated approach, using existing WS-Context-enabled tools, can enable the implementation of multi-party interactions within an activity, and even nested activities shown in Listing 2.

```
Context ctx = ContextService.begin();
myService.someOperation(...); // Will receive top-level context
// Create a nested context
Context myNewContext = ContextService.begin(ctx);
anotherService.someOperation(...); // Will receive both contexts
yetAnotherService.someOtherOperation(...); // Will receive both contexts
myNewContext.end();
yourService.someOtherOperation(...);
ctx.end();
```

Listing 2: Pseudo Code Demonstrating the Use of Nested Contexts

The above example shows that the developer has a minimal overhead when using activity contexts (which are used only at the discretion of the service provider in any case). All the developer has to do is remember to create and complete contexts within their application. This is straightforward and is implicitly familiar to programmers who have worked with now commonplace transactional systems. Furthermore, while sharing a GSH must be explicitly programmed, sharing context (once a context has been established) is implicit, and the same is true of any nested contexts which enable federated units of work to be assembled into a composite whole.

3.2. Data References

One of the most fundamental use cases for the Grid is the ability to share data between computational resources. This requirement can be supported by the OGSF model through the concept of a Grid Service Instance, and must be supported in any other approach.

3.2.1. Data Exposed Through Grid Service Instances

Fundamental to the WSA model is the concept of XML messages that are exchanged between consumers and services. In most cases, the response XML message received by a consumer will contain the results of the action performed and no more message exchanges will be required. This is because WSA encourages coarse interactions between consumers and services and promotes messages with a payload that is descriptive and independent of any state maintained by a service. The focus is on well structured XML documents that travel between consumers and services. For example, queries issued to data-oriented services can be answered directly, as shown in the diagram of Figure 2.

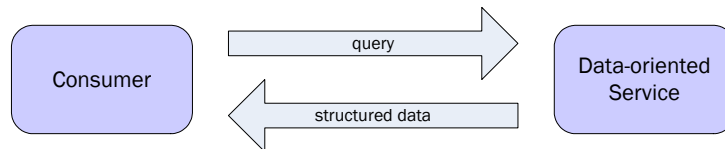


Figure 2: Data Returned as Structured XML

There are situations, for example in the case of services that provide access to database resources, where it is inefficient to return large amounts of data as the result of operation requests. This is important in Grid computing where accessing and sharing large datasets is a common use-case. Using the concept of a Grid Service Instance, it is possible to encapsulate data and use the GSH as a way to implement “by-reference” semantics. The interface implemented by the Grid Service Instance can provide access to the encapsulated data

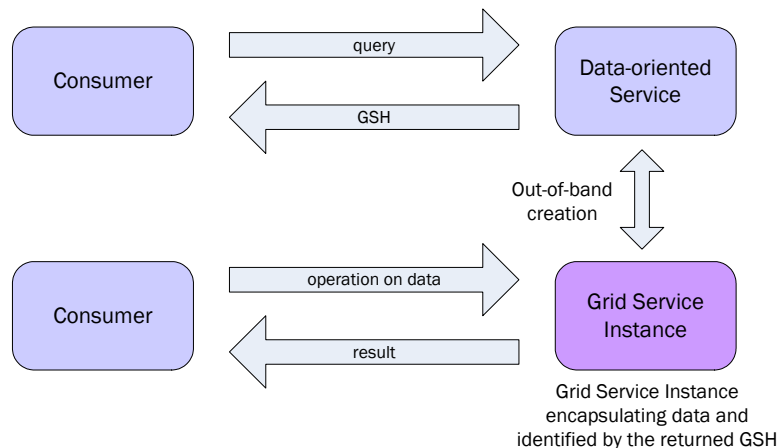


Figure 3: Data Accessed Through a Grid Service Instance Interface

In the example of Figure 3, a consumer submits a query to a service that provides access to a database resource. Since the result of the query is a large dataset, the data-oriented service decides to create a new Grid Service Instance that implements the well-defined interface for datasets and returns its GSH to the consumer, instead of returning the actual dataset. The consumer, after resolving the GSH to a GSR, can call operations on the Grid Service Instance that either process the dataset hidden behind the interface or provide access to parts of the dataset.

This is reminiscent of an object-orientated approach, and suffers from the limitations of that methodology. Being able to give a unique name to a piece of data via the GSH that identifies its service instance is useful since it allows multiple parties to share the same data by sharing the GSH. However, the emphasis is on defining the interface of the referenced service instance rather than on the structure of a document that is to be exposed. That means the data is not directly available (though it could be exposed through some well-known Service Data Element); the interface designers must anticipate all the possible ways the data is to be accessed or processed; and the consumers are limited in using only those available operations.

3.2.2. Data as Structured Documents

It would be more flexible if we were able to treat data simply as structured XML documents which (in the case of large sets of data) could be simply referred to by a GSH-like identifier, yet not be coupled to any particular service instance (and thus the set of operations that the instance supports).

For example, where the consumer requires the service to directly return the results of executing the query, it can invoke the appropriate operation to retrieve data schematised according to the design of the service provider. In fact this is no different in practice from any other Grid/Web services interaction. However where the size of the data set involved is significant or where the consumer knows in advance that data is to be transferred to a third-party service for processing, the consumer should use the operation that the service exposes which returns a Grid *dataref* (Figure 4).

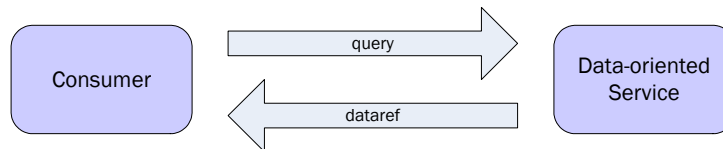


Figure 4: Dataref to Structured XML Returned

A dataref consists of the URI that uniquely identifies the referenced data (the *id* element), the qualified name of the type (e.g., XML Schema type) of the data, and an optional set of endpoints. An endpoint consists of a WS-Addressing [26] EndpointReference element (though other addressing schemes could easily be substituted) and two time-related elements (influenced by the design of the OGSi GSR) hinting on the validity of that particular endpoint.

The *id* element can be used to globally identify the data to which the dataref refers. For example, it could be a Life Sciences ID (LSID) [27] which is used to uniquely identify Life Sciences related resources. A dataref could be stored in a database or given to a high level service (such as a registry service) that updates the set of endpoints that can operate on the referred data.

An endpoint carries location and protocol specific information, similar to a Grid Service Reference. The endpoints in a dataref are a hint to the consumer on the services and interfaces that can operate and/or access the referenced data. We expect that those interfaces and the semantics of their operations would be defined by the OGSA high level services (such as database access and integration services). There is no restriction on how many services, or interfaces within the same service, can be used to operate on the same data.⁷

An XML schema suitable for describing datarefs is shown in Listing 3.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  targetNamespace="http://www.neresc.ac.uk/gaf/dataref"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
    schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
  <xs:complexType name="dataref">
    <xs:sequence>
      <xs:element name="id" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>
      <xs:element name="type" type="xs:QName" minOccurs="1" maxOccurs="1"/>
      <xs:element name="endpoint" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="wsa:EndpointReference"
              minOccurs="1" maxOccurs="1"/>
            <xs:element name="valid-from" type="xs:dateTime"
              minOccurs="0" maxOccurs="1"/>
            <xs:element name="valid-until" type="xs:dateTime"
              minOccurs="0" maxOccurs="1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
  
```

Listing 3: Defining the Dataref XML Schema

Note that the absence of *valid-from* indicates that the endpoint is immediately valid and the absence of *valid-until* means the endpoint has a logically infinite lifetime (a simplification of the OGSi approach).

⁷ Section 5.2 (page 1) contains examples of possible usage patterns for dataref.

Where datarefs are used, the message exchange pattern differs from the simple case where structured data is immediately returned, since at some point the dataref may be used to access the referenced structured data or call operations that on a Web service that can process all or part of that data.

The requirement for referencing data is orthogonal to modelling stateful interactions. The dataref and context-based solutions demonstrate the separation of concerns and allow greater flexibility in usage patterns. For example, a single piece of data could be accessed by multiple consumers in a single context, or alternatively in a separate context per consumer. This is in contrast to the OGSi approach which uses the Grid Service Instance as a single solution to capture both of these requirements, making it difficult to vary each independently.

3.3. Support for Services with “Publicly Visible State”

The Grid community has suggested that the ability to deploy and use services with “publicly visible state” is a useful feature that is currently missing from Web Services.

3.3.1. The OGSi Service Data Elements

The OGSi group defines Service Data Elements (SDEs) as the mechanism by which the internal state of Grid Services Instances may be accessed. Since the WSDL 1.1 specification did not provide any means to declare the “publicly visible state” of a Web Service, OGSi uses the WSDL extensibility mechanism to enrich the interface of Grid Services Instances with service data declarations.

The introduction of SDEs in WSDL, although syntactically consistent with the WSDL specification, is not consistent with common practices of the Web Services community. Tools that consume WSDL are unable, without modifications, to understand the service data declarations in WSDL documents, and so they are ignored. Web Service specifications are also unaware of SDEs.

The OGSi specification adds two mandatory operations to the interface of every Grid Service Interface to support the concept of SDEs. These operations allow consumers to retrieve, query, and set the values of SDEs belonging to the instance on which they are called. The use of these operations is necessary because the service data declarations that are part of a Grid Service Instance’s interface do not necessarily represent the entire set of all SDE-related messages that the instance understands; the set of available SDEs may change through the lifetime of that Grid Service Instance. This is akin to weak typing in a programming language. Of course, some SDEs may be declared as always being available, but the fact is that OGSi diverges from the aim of WSDL which is to define a contract between a service and its consumers on the messages that can be exchanged.

3.3.2. Attributes in WSDL 1.2

There has been a push to introduce the concept of an *attribute* in the next version of WSDL (v1.2 or v2.0). From discussions to date, it appears that attributes, if adopted, will introduce only a small subset of the OGSi-defined SDE semantics.

We strongly believe in compliance with Web Services specifications, semantics, and practices. Therefore, we are not defining any mechanism that may offer functionality and semantics similar to those of Service Data Elements found in OGSi. Instead, we propose to adopt whatever mechanism is defined by the WSDL community.

3.4. Service Deployment and Management

One of the biggest perceived gains from the notion of Grid service instances is the ability to create a service instance dynamically, sometimes outside the administrative domain of the hosting environment. This stems from requirements to, for example, process data on a resource close to where that data is produced.

3.4.1. Issues with Dynamic Creation of Grid Service Instances on a Different Administrative Domain

While we are sympathetic to the requirement of being able to create and consume a Grid Service Instance from an arbitrary administrative domain (security and policy permitting), we do not believe that it is possible in the general case. A service is a logical manifestation of physical resources (databases, programs, computers, humans) and so the notion of “deploying” a service instance, while appealing in the abstract, is practically impossible unless we can constrain the deployment circumstances. For example, deploying an arbitrary Java program as a service is, in the general case, doomed to failure because the resources which that program utilises are not likely to be accessible at the remote location where the service instance is ultimately deployed.

Although the OGSi specification hints at this kind of functionality, as far as we know, no high-level services have yet been built that enable the dynamic deployment of Grid Service Instances on a different administration domain. However, it is the case that Grid Service Handles and References have the potential to play a part in providing location transparency for OGSi Grid Service Instances and, hence, providing the basis for such a high-level service. We believe that there Web Services specifications that could provide equivalent functionality (e.g., WS-Referral [28] and WS-Routing [29]), or that patterns that leverage the underlying SOAP fault mechanism to indicate invalid endpoints (and possibly convey new endpoint information) could be created.

In the simplest case, the SOAP fault mechanism can be used to indicate an invalid address, and propagate valid addressing information as shown in Figure 5.

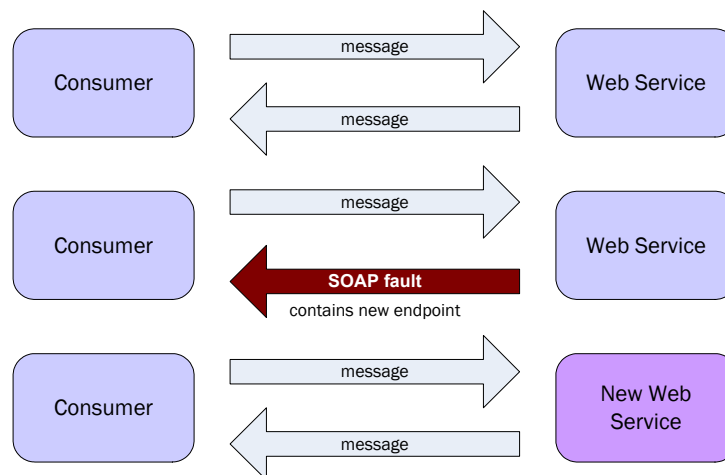


Figure 5: Using SOAP Faults to Handle Service Relocation

The approach taken in Figure 5 has the advantage that it is SOAP-native, and it easily supports simple point-to-point invocation of Web services. Normally the consumer of a service sends a request message, and (usually) receives a response at some later time. However, if the service deployment location has changed in-between successive invocations, the service creates a SOAP fault and packages the new location of the service as a WS-Address inside the SOAP fault structure. The infrastructure supporting the consumer may then re-bind to the new endpoint, or may pass a structured exception back to the consumer, which—according to the application logic—may decide to re-bind to the service or attempt to find a new one.

In those situations where a Web service invocation passes through a number of WS-Routing and WS-Referral based SOAP nodes, we can leverage those protocols to provide seamless re-binding to services, as demonstrated in Figure 6.

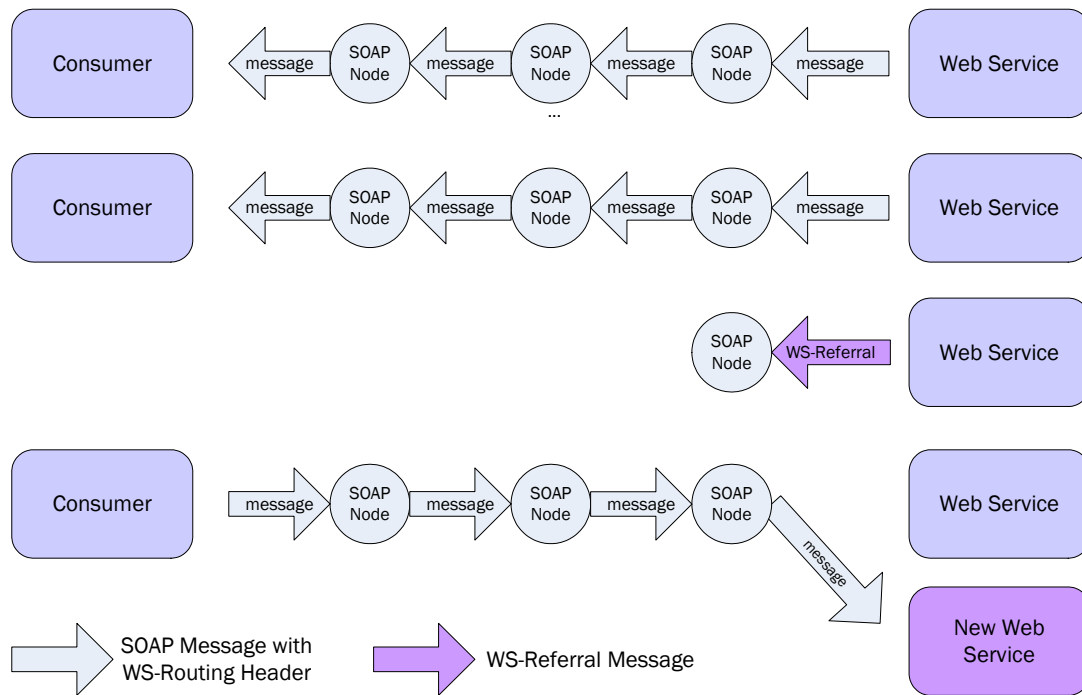


Figure 6: Handling Service Redeployment with WS-Routing and WS-Referral

While the example shown in Figure 6 is a simple case of what can be done with WS-Routing and WS-Referral, it shows the typical use case for re-binding to re-deployed services in a Grid environment. The normal case is that a consumer sends a message out onto the SOAP network which is routed via SOAP nodes according to the content of its WS-Routing headers. Eventually the message arrives at the Web Service and is consumed. At some point later the reverse happens with a response message being sent out onto the network to be propagated back to the consumer usually, though not necessarily, by following the same path in reverse.

If we have a WS-Routing capable network, we can then use WS-Referral to configure routing tables on the fly. Though we do not seek to mandate particular strategies here, an obvious simple case (as demonstrated in Figure 6) is for a service which is being redeployed to send a WS-Referral update message to the SOAP node immediately ahead of it to update that node's WS-Routing tables to forward messages to a different endpoint for consumption.

The advantage of this solution is that it is more flexible than the SOAP Fault-based approach, though it requires additional Web services technology to implement, and so we would envisage the SOAP Fault-based approach being most widely accepted as its entry point is much lower. However, irrespective of which approach is adopted for a particular Grid application, no proprietary (i.e. non-Web Services) technology is utilised, nor are the architectural principles of SOA violated.

3.4.2. A High Level Service for Remote Service Deployment

There is a need for certain types of Grid applications to dynamically deploy services on a dynamically acquired host, possibly close to the location of the resources on which they are operating. We believe that dynamic deployment capabilities should be defined by a high level service, rather than the underlying Grid infrastructure. For example, with the advent of technologies like BPEL [2] which are constrained to deal only with Web Services, we have an opportunity to deploy service logic without having the concerns that deploying a binary service raises. Deploying a BPEL (or equivalent) defined service involves only the transfer of an XML document (the BPEL script) to a service which offers to expose that script as a Web service. This is shown in Figure 7.

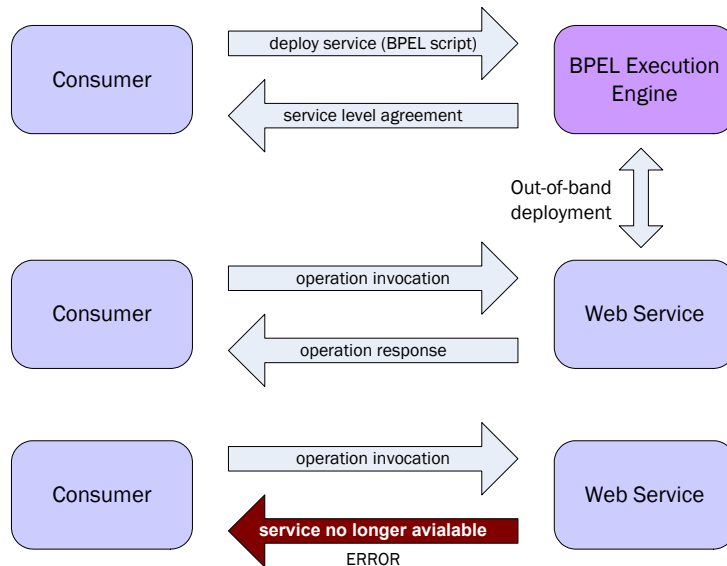


Figure 7: Dynamically Deploying a Service Implemented with a Constrained Programming Language

In the first instance, a consumer deploys its BPEL-encoded logic to a BPEL execution engine, and receives in response a service level agreement which describes (for example) time to live or number of invocations allowed on that service. At deployment time the deployer may specify further constraints such as who is allowed to invoke the deployed service, though this is at the discretion of the BPEL execution service.

For the given deployment lifetime of the service (which might be determined by number of invocations, or by time to live, or other parameters) the deployed service is accessible to authenticated consumers. If the deployment location has been carefully chosen to take advantage of spatial locality of dependent services, performance should be enhanced.

We believe that the Grid community will define the requirements, functionality, and interface of a high level service enabling dynamic deployment and management of services. Issues like description of resources and their discovery, security and policies, service level agreements are already addressed by Global Grid Forum [30] working groups. The outcome of these working groups could be used in the definition of a service enabling dynamic deployment of other services. It is not inconceivable that the community will define the necessary constraints for enabling the dynamic deployment of services written in some binary, host environment specific language like Java or C#, though because of the generality of such languages those constraints will be necessarily severe.

We also expect the Grid community to define the interfaces for accessing and modifying information like the lifetime, access policies, security and service-level agreements of deployed services.

3.5. Lifetime in the Grid Application Framework

Lifetime is an important aspect in OGSF. The Grid Application Framework proposed in this section is built around four concepts: service, context, data, dataref. The lifetime characteristics of these concepts are as follows:

- **Service.** There is no notion of lifetime for a service. A service in this framework either exists or does not exist; a service is deployed or undeployed. There may be situations, however, where a high-level service may wish to associate some lifetime characteristics to a deployment of a service (e.g., a particular service-level agreement may only allow the dynamic deployment of a service for a limited amount of time). We do not see the need to provide a framework-based solution for such situations.

- **Context.** A WS-Context may have a timeout value hinting on the validity of the context. The value may be changed by the participants of the activity that the context identifies. A service may also decide to update the timeout.
- **Data.** The lifetime of a particular set of data is application-specific and is not a concern for this framework.
- **Dateref.** The dateref has no lifetime associated with it although a high-level service may add such information (e.g., to hint that the referred data may go away and hence one should not try to access it through the set of endpoints). However, every endpoint in the dateref has lifetime characteristics associated with it.

4. Proposed Architecture

By having a compulsory layer between Web Services and OGSA, OGSI forces all Grid Services to sit above that layer, whether they benefit from it or not. Further, it makes a binary divide between services that are Grid services and those that are not and makes everyone reliant on an implementation of OGSI, such as GT3 [31].

We envisage a Grid application framework implemented strictly within the vision of the Web Services Architecture. Such an application framework is built only on existing specifications, can coexist with other Web Services specifications, and is able to leverage all the available Web Services tools without modifications. A stack-based view of the envisaged architecture is presented in Figure 8 and contrasted with the existing approach which is presented in Figure 9

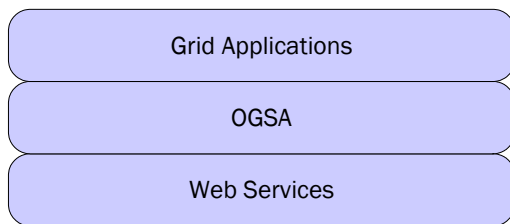


Figure 8: Proposed Approach

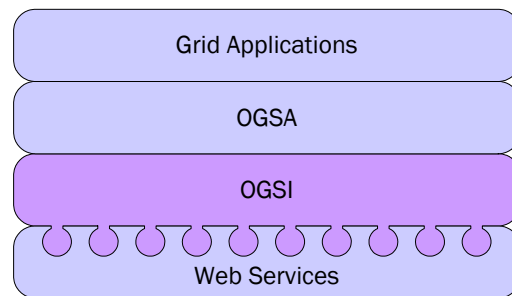


Figure 9: The OGSI Approach

The proposed architecture suggests that when designing a service, a Grid architect could simply select those aspects of the Grid Application Framework that were important to that particular service. In some cases, there may be no such need. So there is no longer the concept of Grid Services: there are just Web services that use, where and if appropriate, specifications adopted by the Grid Application Framework. We believe that this is in line with the approach taken by Web Services community.

A stack view of a service implementation within such a Grid Application Framework is shown in Figure 10.

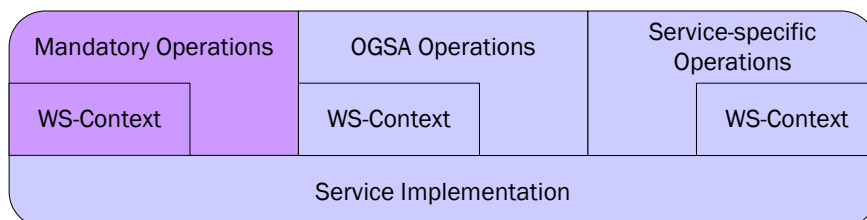


Figure 10: The Stack for a Web Service for the Grid

We expect there to be (at most) three distinct aspects to the interface of a Grid service.

- **Mandatory operations.** The first of these are the mandatory operations that all Grid services must support, some of which may be contextualised. To date, however, we have yet to identify any such operations, but we do not discount the possibility of their existence. If such

operations were found, they would become the equivalent of an OGSi layer underneath the OGSA layer in Figure 8.

- **OGSA operations.** The second aspect of a Grid service is its optional interfaces which are specified by special interest groups within the Grid community as part of the OGSA platform. For example, one such interface may mandate how a Grid service is managed, while another may describe the standardised interface of a specific kind of service (such as a database or specific computational service). Given our focus on achieving stateful interactions with services through the use of WS-Context, operations in the OGSA interfaces may be contextualised, though equally there may be circumstances where stateful interactions are not required.
- **Service-specific operations.** The remaining interfaces support operations which expose the particular functionality of a specific Grid service. Those operations may be contextualised, and will differ depending on the service's purpose and implementation.

5. Applying Existing OGSA Concepts: Examples

In this section we present examples of how the proposed Grid Application Framework could be used to implement Grid applications. Our focus is on stateful interactions and datarefs.

5.1. Stateful Interactions – The Counter Service Example

We use the Counter Grid Service example from the OGSi Primer document [32] to demonstrate the use of stateful interactions. A Counter Grid Service allows Grid applications to incorporate counter-related functionality (i.e., perform increase/decrease operations and get the value of the counter). The WSDL interface of a Counter Service is presented in Listing 4 below:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- contextualised "counter" service -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/??"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.neresc.ac.uk/gaf/examples/counter"
  xmlns:counter="http://www.neresc.ac.uk/gaf/examples/counter">
  <wsdl:types>
    <xs:schema>
      <xs:element name="getValueResponse" type="xs:integer"/>
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="increaseMessage"/>
  <wsdl:message name="decreaseMessage"/>

  <wsdl:message name="getValueMessage">
    <wsdl:part name="value" element="counter:getValueResponse"/>
  </wsdl:message>

  <!-- wsdl:interface is the new name for wsdl:portType in the next version
  of the WSDL specification -->
  <wsdl:interface name="CounterInterface">
    <wsdl:feature required="false"
      uri="http://www.webservicestransactions.org/wsctx/2003/03"/>

    <wsdl:operation name="increase">
      <wsdl:input message="counter:increaseMessage"/>
    </wsdl:operation>

    <wsdl:operation name="decrease">
      <wsdl:input message="counter:decreaseMessage"/>
    </wsdl:operation>
  </wsdl:interface>
</definitions>
```

```

<wsdl:operation name="getValue">
  <wsdl:output message="counter:getValueMessage "/>
</wsdl:operation>
</wsdl:interface>
</definitions>

```

Listing 4: WSDL interface for the Accumulator service

Note the use of the `wsdl:feature` element, which is one of the proposed new elements that may be available with the next version of the WSDL specification. Its use is not required by the proposed Grid Application Framework, but Listing 4 demonstrates the way we expect Grid Service interface designers to declaratively specify whether a context is required. In this case, the `wsdl:feature` declaration specifies that the *CounterInterface* interface understands the WS-Context specification but does not require it.

In OGSi, multiple Grid Service Instances would be implemented to provide stateful interaction semantics. In this proposal, we use contextualised interactions to achieve the same behaviour with a single Web Service. Note that no factory lookup and Grid service instantiation are required; neither are GSH and GSR resolution. In line with SOA principles, it is the conversational context that provides “stateful” and “transient” properties rather than additional Grid-specific mechanisms. This is in line with the way stateful interactions are handled in the wider Web Services world.

The Counter Service could be used in three ways:

- **The counter could be shared amongst all consumers of the service.** No context is therefore necessary. The implementer of the service just maintains a “global” value for the counter.
- **The counter could be seen as being unique for a particular consumer.** The consumer that wishes to use the Counter Service should make sure that a new WS-Context is implicitly transmitted during its first interaction with the service. The WS-Context can be created either by a Context Service or by the Counter Service itself. For example, if the semantics of the service are that there is no such thing as a “global” value, then the Counter Service could automatically generate a new WS-Context (by acting as a consumer of a WS-Context service itself), whenever it receives a message that does not carry a WS-Context.
- **The counter could be shared between only a subset of the possible consumers (Figure 11).** The consumer that initiates a stateful interaction with the Counter Service could make its “private” counter available for others to use, hence giving the impression that the value of the counter is shared. This is achieved just by sending the WS-Address of the Counter Service to the new participant. The WS-Context will be implicitly transmitted.

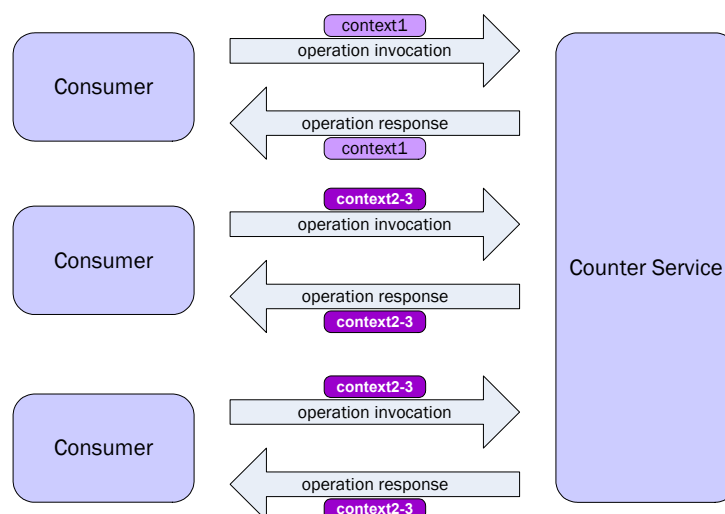


Figure 11: Interaction with the Counter Service

5.2. Datarefs – Data-oriented Service

The dataref construct allows us to reference structured data or resources and provides hints on the available services and the interfaces that can operate or provide access to that data. In this section, we explore possible usage patterns of datarefs with a data-oriented service.

5.2.1. The Service

For the purposes of this example, a data-oriented service provides access to database resources and allows queries to be submitted without requiring a stateful interaction to be established first. We would expect Web/Grid Services offering access to database resources to support the use of contextualised interactions (sessions) (e.g., DAIS [33]).

A simple interaction with a data-oriented service is shown in Figure 12. A consumer sends a query to the data-oriented service but instead of the resulting dataset document it receives a dataref. This may be for many reasons: efficiency concerns (the dataset may be very large); service semantics (the service owner may not wish to expose the data directly but only through particular interfaces); the resulting dataset may already reside somewhere else (the service somehow knew that the result for the query was an external dataset identified by the returned dataref); or, the service was asked to directly deliver the resulting dataset to a data-processing service using a faster communications infrastructure (the WS-Address of the data-processing service was given to the service). It is up to data-oriented service specifications or implementers to decide where the use of datarefs is appropriate.

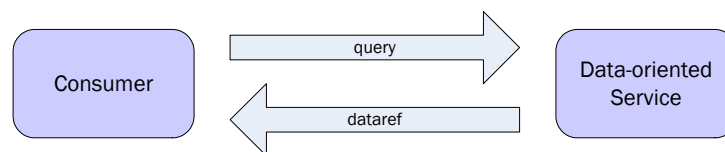


Figure 12: A Simple Interaction with a Data-Oriented Service

In the example, we will assume that the data-oriented service supports a number of interfaces which are defined by the community: query interface, data processing interface, and data access interface (Listing 5). The community has also defined the structure of the dataset document, with <http://www.data-oriented-services.org/dataset> as its namespace.

```
<wsdl:interface name="QueryInterface">
  <wsdl:operation name="query">
    <wsdl:input message="tns:queryRequest_queryString"/>
    <wsdl:output message="data:queryResponse_dataref"/>
  </wsdl:operation>
</wsdl:interface>

<wsdl:interface name="DataProcessingInterface">
  <wsdl:operation name="sort">
    <wsdl:input message="tns:sortRequest_dataref"/>
  </wsdl:operation>
</wsdl:interface>

<wsdl:interface name="DataAccessInterface">
  <wsdl:operation name="get">
    <wsdl:input message="tns:getRequest_dataref"/>
    <wsdl:output message="tns:getResponse_dataset"/>
  </wsdl:operation>
</wsdl:interface>
```

Listing 5: Examples of the Three Interfaces Supported by the Example Data-Oriented Service

5.2.2. Usage Patterns

The proposed framework is silent on the possible ways datarefs could be used. Here we present three simple usage scenarios for achieving equivalent functionality.

Figure 13 shows the exchange of messages between a consumer and a service supporting the interfaces presented in Listing 5. Note that the use of dataref is explicitly passed as an argument to the operation.

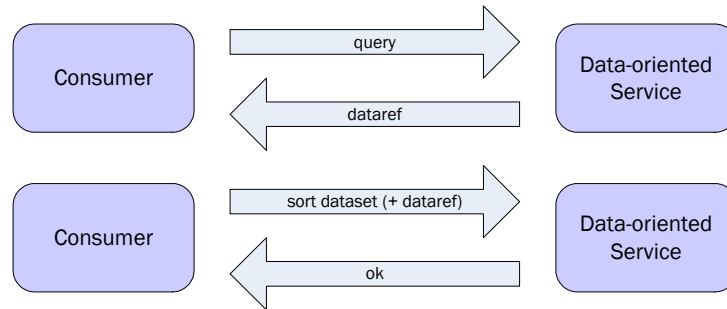


Figure 13: Explicitly Using a Dataref to Efficiently Access a Dataset

There may be occasions where we wish to repeatedly operate on the same dataset, and as such we would not want to have to explicitly reference the same data explicitly per operation. In such cases we would prefer that operation invocations were contextualised by the dataref. The interface of a service may declaratively define such a requirement (Listing 6).⁸ The consumer of the service is seen as having access to a set of operations that work on the same data without the need to create new instances of that service, as it is the case with OGSi. This is illustrated in the diagram of Figure 14.

```
<wsdl:interface name="DataProcessingInterface">
  <wsdl:feature required="true" uri="http://www.neresc.ac.uk/gaf/dataref"/>
  <wsdl:operation name="sort"/>
</wsdl:interface>
```

Listing 6: Revision of the DataProcessingInterface to Support dataref-Contextualised Interactions

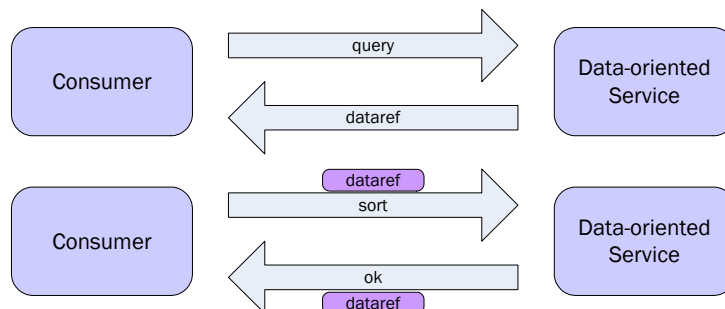


Figure 14: Implicitly Using a Dataref to Simplify Multiple Interactions with the Same Dataset

Figure 14 captures the conceptual message exchanges for the dataref context and the following (pseudo) code snippet shows how this optimisation might simplify matters for developers.

```
DataRef myDataRef = dbService.query(myQuery);
Dataset dataset = myDataRef.getContextualisedAccessToDataset();
/* Get a valid service endpoint from the
   dataref, contextualise the operations
   and present it as a dataset object */
dataset.sort();
// or more queries on the dataset (if such an operation was supported)
dataset.query(queryOnDataset);
dataset.query(anotherQueryOnTheSameDataset)
```

Listing 7: Pseudo code demonstrating the use of dataref

⁸ The declarative approach to describing features that are supported/required by a service interface is introduced in WSDL 1.2.

Finally, another way to use the dateref would be in conjunction with a contextualised interaction. After a dateref has been received, the consumer establishes a stateful interaction with the service that is to operate on the data. An operation like *use* is called to associate the particular dateref with the stateful interaction. From now on, the operations on the dateref may be performed without the need to explicitly identify the dateref. The revised data-processing interface is shown in Listing 8, while the conceptual exchange of messages is shown in Figure 15.

```
<wsdl:interface name="DataProcessingInterface">
  <wsdl:feature required="true"
    uri="http://www.webservicestransactions.org/wsctx/2003/03"/>
  <wsdl:operation name="use">
    <wsdl:input message="tns:useRequest_dateref"/>
  </wsdl:operation>
  <wsdl:operation name="sort"/>
</wsdl:interface>
```

Listing 8: Revision of the DataProcessingInterface to Support Association of datarefs with Context

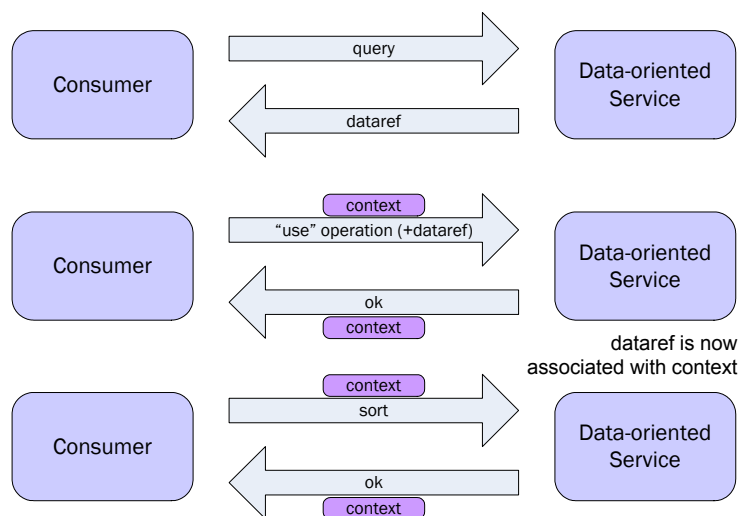


Figure 15: Associating a dateref with a Context to Simplify Multiple Interactions with the Same Dataset

Of course, the scenarios presented in this section correspond only to a small subset of the possible ways a dateref could be used and are meant merely to illustrate possibilities rather than provide a normative definition.

5.2.3. Retrieving Referenced Data

As demonstrated in the previous section, interfaces can be defined with operations that can operate on data given a dateref. A dateref may contain a number of endpoints that can be used to access the referred data. However, as we discussed in 3.2 (page 8), one of the characteristics of the dateref construct is that it points to a typed XML document rather than just one or more service/interfaces. A question arises on whether a consumer could retrieve that XML document and treat it as if it was returned to it directly, which is the normal behaviour a consumer would expect from a Web Service.

This proposal does not define any specific access mechanism. Instead, it delegates the provision of such a mechanism to high level services and the implementer of a service. For example, the Data Access and Integration community may decide that such a mechanism is useful for moving data and, hence, define an interface that allows for a Dataset XML document to be retrieved given a dateref.

A particular service may decide not to support such an interface, hence, not allowing the data to travel outside its administration domain. In the example of the previous section, the data-oriented service supports the *DataAccessInterface*. That means that it may allow consumers to retrieve the entire dataset XML document. Such an example is presented in Figure 16. A consumer submits a query and retrieves a dateref as a result. Then, it goes to a data-processing service and

requests that an operation be executed on the referenced data. The data-processing service checks that one of the endpoints in the *dataref* points to an interface that supports the well-known *DataAccessInterface*. Hence, it can send a request to the data-oriented service and retrieve the dataset XML document before processing with the execution of the operation the consumer requested.

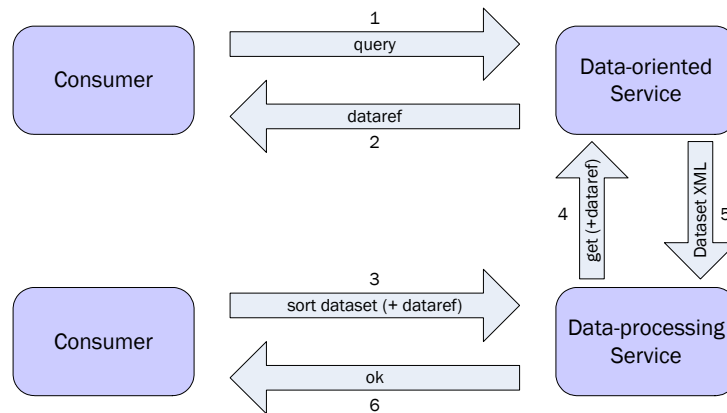


Figure 16: Example of Retrieving the XML Dataset Document Referenced by a *dataref*

In the above example, if the consumer knew which data-processing service it was going to use before it submitted the query to the data-oriented service, it could have requested the delivery of the data directly to the data-processing service, provided the appropriate interfaces existed and, of course, any security and policy restrictions were met. A WS-Address identifying the receiving data-processing service would have been required (message 1 of Figure 17).

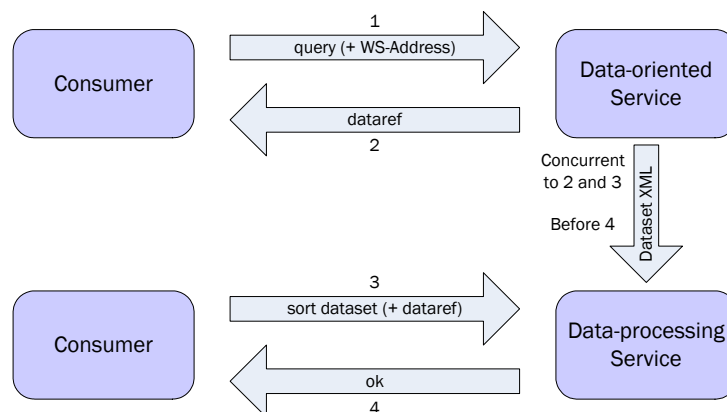


Figure 17: Example of Delivering an XML Dataset Document to a Service

5.2.4. Data-related Registries

The information in a *dataref* does not have to be static. Although the values of the *id* and the *type* elements cannot change, the set of endpoints is variable. The proposed Grid Application Framework does not define any mechanism for updating the endpoint-related information in a *dataref*. We expect specifications of high level services to talk about the process by which the set of endpoints in a *dataref* is modified. We envisage specifications for data-specific registries that can, for example, provide the endpoints for replicas of a referenced dataset, or, Peer-to-Peer infrastructures could provide services to update *datarefs* with the endpoints of services maintaining cached copies of a referenced document (in a similar way to current filesharing software).

6. Conclusions

The question “What is a Grid Service?” is still often asked and is sometimes answered with “Just a Web Service.” This answer causes great confusion since if the two are the same then there is clearly no value in distinguishing between them. However, the two **are** different due to the

additional behaviours associated with a Grid Service Instance that simply do not exist in a normal Web Service (e.g., statefulness, transience, SDEs, lifetime management, etc.).

However, we believe that the new concepts introduced by OGSi with their resulting practices for developing services, on which Grid applications are built, introduce the danger that the Grid community will follow a separate path for building distributed applications from that adopted by the wider Web Services community. This would have major implications, reducing the ability of Grid application developers to leverage the huge amounts of investment in Web Service specifications, tools, services and educational materials.

When the OGSi working group started the work on providing solutions for the Grid requirements, they made the sensible decision to use Web Services technologies for their infrastructure, and invented solutions that met Grid requirements. However, at that time, the number of Web Service specifications with which they could work was limited, and hence, they had to devise solutions to problems that the Web Services community encountered at a later stage.

Today, there is a great deal of work and specifications produced by the Web Services community that could be leveraged to implement a Grid Application Framework for building Grid solutions. In key areas, the Web services community has now tackled the same requirements as the Grid community, but has come up with different solutions. Therefore, we believe that the Grid community should now consider how to bring OGSi into line with the current and upcoming specifications and the practices adopted by the Web Services community. This will allow Grid developers to exploit Web service tools, specifications, services and practices, so avoiding the need to build a parallel set of solutions.

This document has presented our views on the current issues with OGSi's relationship to the practices of the wider web service world, and has gone on to propose an architecture for a Grid Application Framework that is, we believe, more in line with practices in that wider computing community. The proposal is based on the Web Services Architecture specification [4], other Web Service specifications and common practices from that world.

One major result from this investigation is our belief that the OGSi Grid Service Instance is overloaded, as it is an attempt to use one mechanism to capture a set of orthogonal concepts: stateful consumer-service interactions, the interface to data, state encapsulation and transience. Merging these concepts in one construct limits the ways in which they can be combined. The proposal in this document separates out these orthogonal concepts, so allowing them to be independently controlled.

A central part of the proposal introduced in this document is to deal with stateful interactions between consumers and services, not through dynamically created Grid Service Instances as in OGSi, but through contextual interactions (as supported by the WS-Context specification). This has the advantage of meeting the same requirement, but in a way that is common in the wider Grid service world, and is also completely independent of data and state encapsulation. To allow orthogonal access to data, the proposal also introduces the concept of a dataref, which incorporates a global data identifier, and a set of endpoints that might offer access to that data. This allows services to return references to data that consumers can use when they require access to it. Because datarefs are orthogonal to context, access to data may or may not be contextualised. We believe that the separation of issues and the provision of distinct solutions allows for the implementation of a more flexible architecture, the adoption of existing tools, and the support for more complex usage patterns.

To conclude, this document has presented our views on the relationship between Web and Grid Services, discussed issues with the current OGSi specification and proposed a new framework that is in line with wider Web Service specifications and practices. We hope that it will help answer some of the questions on the relationship between OGSi and the Web Services Architecture with its related specifications. Projects that have adopted Web Services as their implementation platform (e.g., myGrid [34]) should now, we hope, have an idea of the issues that must be addressed when moving to an OGSi-based implementation. Conversely, projects that have used OGSi as their implementation platform but now wish to provide similar functionality in

a WSA-only manner (e.g., OGSA-DAI [35]) can consider the proposal of Section 3 as a way of providing at least as much functionality, but in a way that is more conformant to wider Web Service specifications and practices. For those developing OGSF services, two parts of that proposal—context and data references—provide mechanisms that could be exploited even without moving entirely away from the current Grid Service Instance-based design. Finally we hope that this document will be used as input to discussions within the OGSF community on the future direction of the Grid infrastructure.

7. Acknowledgements

We are grateful to the following people, for their feedback to this document:

Tim Banks (IBM), Dave Ingham (Arjuna Technologies), Mark Little (Arjuna Technologies), Jim Smith (University of Newcastle upon Tyne).

8. References

1. Tuecke, S., et al., *Open Grid Services Infrastructure (OGSI) - Version 1.0*. 2003: <https://forge.gridforum.org/projects/ogsi-wg>.
2. OASIS Web Services Business Process Execution Language: <http://www.oasis-open.org/committees/wsbpel>.
3. OGSF Working Group, <https://forge.gridforum.org/projects/ogsi-wg>.
4. Web Services Architecture: <http://www.w3.org/TR/ws-arch>.
5. Service-Oriented Architecture (SOA) Definition: http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.
6. Web Services Description Language (WSDL): <http://www.w3.org/2002/ws/desc>.
7. SOAP Version 1.2 Part 1: Messaging Framework: <http://www.w3.org/TR/soap12-part1>.
8. Purdy, D., *Service Oriented Architecture (presentation)*. 2003: <http://www.douglasp.com/talks/web400.zip>.
9. IBM BPWS4J: <http://www.alphaworks.ibm.com/tech/bpws4j>.
10. Collaxa BPEL Server: <http://www.collaxa.com>.
11. Foster, I. and C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*. 1998, Morgan Kaufmann.
12. Foster, I., et al., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002, Global Grid Forum.
13. Foster, I., *The Grid: A New Infrastructure for 21st Century Science*. *Physics Today*, 2002. **55**(2): p. 42-47.
14. Foster, I. and D. Gannon, eds. *Open Grid Services Architecture Platform (OGSA)*. 2003, <https://forge.gridforum.org/projects/ogsa-wg>.
15. Berman, F., G. Fox, and T. Hey, eds. *Grid Computing: Making The Global Infrastructure a Reality*. 2003, John Wiley & Sons.
16. WS-Coordination: <http://msdn.microsoft.com/ws/2002/08/WSCoor>.
17. WS-Transaction: <http://msdn.microsoft.com/ws/2002/08/wstx>.
18. OASIS Business Transaction Protocol: <http://www.oasis-open.org/committees/business-transaction>.
19. OASIS Web Services Security: <http://www.oasis-open.org/committees/wss>.
20. OASIS Web Services Reliable Messaging: <http://www.oasis-open.org/committees/wsrn>.
21. Web Services Composite Application Framework (WS-CAF): <http://www.iona.com/devcenter/standards/WS-CAF>.
22. Web Services Context (WS-CTX): <http://www.iona.com/devcenter/standards/WS-CAF/WSCTX.pdf>.
23. Berners-Lee, T., R. Fielding, and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*. 1998: <http://www.ietf.org/rfc/rfc2396.txt>.
24. Web Services Transaction Management (WS-TXM).
25. Web Services Coordination Framework (WS-CF).
26. Web Services Addressing (WS-Addressing): <http://msdn.microsoft.com/ws/2003/03/ws-addressing>.
27. I3C Life Sciences Identifiers (LSIDs): <http://www.i3c.org/wgr/ta/resources/lid/docs/index.htm>.
28. Web Services Referral Protocol (WS-Referral): <http://msdn.microsoft.com/ws/2001/10/Referral>.
29. Web Services Routing Protocol (WS-Routing): <http://msdn.microsoft.com/ws/2001/10/Routing>.
30. Global Grid Forum: <http://www.gridforum.org>.
31. The Globus Toolkit: <http://www.globus.org/toolkit>.
32. OGSF Primer: https://forge.gridforum.org/docman2/ViewCategory.php?group_id=43&category_id=340.

33. *Data Access and Integration Services (DAIS)*: <https://forge.gridforum.org/projects/dais-wg>.
34. *myGrid*: <http://www.mygrid.org.uk>.
35. *Open Grid Services Architecture - Data Access and Integration (OGSA-DAI)*: <http://www.ogsa-dai.org.uk>.