

# Using Web Services to Build Grid Applications

## The “No Risk” WSGAF Profile

Savas Parastatidis, Jim Webber, Paul Watson

School of Computing Science, University of Newcastle, Newcastle upon Tyne, UK

### Abstract

This document presents design pattern recommendations for building Grid applications based on the principles of service orientation and using existing Web Services technologies. The design pattern described in this document is a subset of the Web Services Grid Application Framework (WSGAF) approach to building Grid applications since it only focuses on the use of the Web Services Interoperability Basic Profile 1.0a and WS-Security. Issues of stateful interactions, logical resource naming, metadata, security, and lifetime management are addressed.

### Keywords

Grid computing, Web Services, Web Services Architecture, Internet-scale computing, distributed computing, Grid Application Framework

## 1. Introduction

In this document we propose a pattern, which is based on existing Web Services specifications and Internet standards, that meets the requirements described in [1] and [2] for building Grid applications and addresses some additional security issues. Our approach is straightforward since it borrows all necessary technology from the (large) Web Services community, which is working on providing solutions – in the form of specifications, tools, and user education – to similar large-scale computing requirements that the Grid community is trying to address.

We view the Grid as an application framework that utilises existing and emerging Web Services technologies, concepts, and practices without changes to the Web Services conceptual model and without creating further infrastructure layers and abstractions. This promotes the coexistence of services built for Grid applications with non-infrastructure specific specifications, such as BPEL [3].

The approach to building Grid applications presented in this document is considered “no risk” since it focuses only on the use of the technologies described in the Web Services Interoperability Basic Profile 1.0a, WS-Security, and other existing Internet standards. We consider the approach presented in [2] as “low risk” since it encourages the use of emerging specifications that have been going through the standardisation for some time (e.g., WS-Context [4]) and/or with a clear position in the Web Services stack and existing implementations (e.g., WS-Addressing [5], OASIS BPEL [6], etc.).

## 2. Profiles

In the Web Services standards arena, there are a broadly three levels of specifications. There are those specifications like SOAP and WSDL which are mature and widely implemented, and importantly have had interoperability issues resolved by inclusion in a WS-I profile. The use of such specifications carries little risk because of widespread understanding, adoption, and standardisation.

The second group consists of those protocols which have achieved standard status, but have yet to be formally included in a WS-I interoperability profile. This includes some important standards like WS-Security (although we understand that a draft of a WS-I Security profile has already appeared). The fact that the interoperability is missing from these standards presents an increased risk compared to the WS-I profiled specifications.

The third group are the emergent specifications which are under consideration by standards bodies such as OASIS. This group of specifications has an increased risk compared to the standardised protocols since while they often have the endorsement of major technology companies, they lack the endorsement of a standards body and lack the rigorous interoperability profiling of the WS-I endorsed specifications.

The final group of specifications are those technologies which have not been submitted to standards organisations for open standardisation. On paper, these specifications present the greatest risk since deployers often have no means of aiding the evolution of these specifications as they are often owned by (groups of) technology companies. However, whether a protocol maintained by a group of powerful technology companies is necessarily any worse off than a protocol subject to the whims and politics of open standardisation is a subjective judgment. Therefore while the strict response would be that this whole family of specifications is the highest risk of all, careful analysis of specific protocols may reveal that the risk of adoption is similar to that of other protocols undergoing open standardisation. For example, although the WS-Addressing [5] specification has yet to appear in a standards organisation, it has already achieved relative wide adoption and open source implementations of it have started to appear [7].

### 3. SOA, WSA, and the Grid

Before embarking upon the discussion of how to build Grid applications using the WS-GAF approach, we first need to define those Web Services concepts used in this document, in order to avoid confusion with the many loose interpretations used by the Web Services and Grid communities.

#### 3.1. Service-Oriented Architecture (SOA) and the Web Services Architecture (WSA)

The notion of service-orientation is not a new one. Distributed application developers have long deployed services as part of their infrastructure. For example, the set of services found in CORBA [8] is an example of the community's efforts to standardise on a number of services that provide functionality needed to support loosely-coupled, distributed object-based applications.

After a significant period of time where object-orientation was the primary methodology for building software, the emergence of XML [9], XML Schema [10], SOAP [11], and the Web Services Architecture [12] has refocused the development community's attention on service-orientation as a means to implement loosely-coupled distributed applications. Unfortunately, the term SOA has become overloaded as researchers and developers have moved their work to be in vogue with the latest buzzwords. In the absence of a well-accepted definition of a service or SOA (some can be found in [13-15]) we define a service in a deliberately minimal fashion as follows:

*A service is the logical manifestation of some physical or logical resources (like databases, programs, devices, humans, etc.) and/or some application logic that is exposed to the network;*

and

*Service interaction is facilitated by message exchanges between services.*

The architecture of a typical service is shown in Figure 1. Such a service consists of resources (such as data, programs, or devices), application logic, and a message processing layer which deals with message exchanges. Messages arrive at the service and are acted on by the application logic, utilising the service's resources as required. Services may be of any scale: from a single operating system process to enterprise-wide business processes.

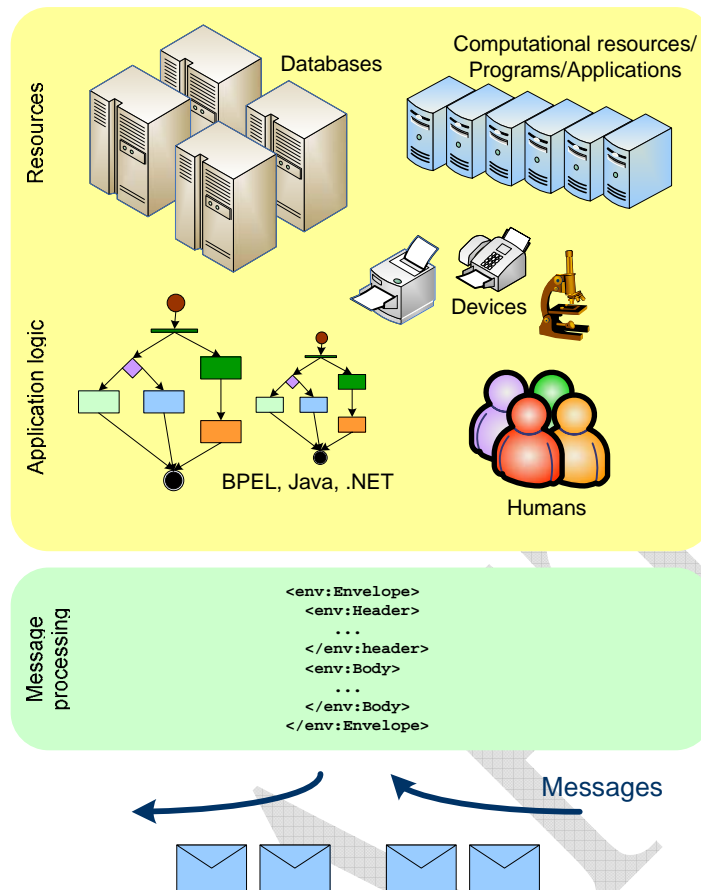


Figure 1: An Example of a service offered by an organisation

When building an application composed from such services, there are a number of design principles that should be followed [16]:

- **Boundaries are explicit:** The boundaries of a service are well-defined when they are incorporated into a distributed application. Other services do not see the internal workings, implementation details, or resource representations of a service.
- **Services are autonomous:** Service implementations are developed and evolve independently from one another.
- **Services share schema and contract, not classes:** In service-oriented architectures, no single set of abstractions (classes) spans an entire application. Services share schemas (contracts) that define the structure of the information that they exchange, not information about their underlying type systems.
- **Policies determine service compatibility:** Services interact with one another only after it has been determined – based on policy assertions – that they can meaningfully exchange information.

The collection of Web Services technologies is an attempt by the industry to define the building blocks for an infrastructure, which is based on the SOA principles, for building loosely-coupled, distributed applications by utilising XML-based technologies. As defined in [12], Web Services interact by exchanging messages in SOAP format while the contracts for these interactions are described through the Web Services Description Language (WSDL) [17]. Also, the Web Services Interoperability Profile 1.0a [18] describes the way in which existing specifications should be used in order to achieve interoperability between different implementations.

### 3.2. Building Service-Oriented Applications

The Grid is an application framework that should be built according to the principles of Service Oriented Architecture, whose concepts are realised by the Web Services Architecture. A software

entity should not claim to be a Web Service merely because it uses WSDL and SOAP, if it does not constitute an implementation of the SOA concepts.

The dynamic and inter-organisational nature of Grid applications suggests that the infrastructure based on SOA and realised by WSA is appropriate since the loose-coupling it offers is more suitable for Grid applications than a distributed object approach would be [19]. Object-oriented infrastructures are more suitable for closed systems since they encourage tight integration of distributed components) [15].

Applying object-oriented principles when building Grid applications deviates from the loosely coupled nature of WSA and encourages a mode of thought where object-plus-methods is the primary abstraction. The object-oriented paradigm encourages a tight-coupling between the state, interface, and identity of an entity (the object). Resulting applications contain complex graphs of inter-relationships between objects that communicate through fine-grained interactions. We believe that such an approach is unsuitable for Internet-scale Grid computing where loose coupling is desirable and independent development of components (services) is the norm.

While we understand that clever design may avoid the potential pitfalls with building Grid applications using the object-oriented paradigm, we believe that the WSA-based notion of services exchanging messages (rather than the classic model of clients invoking servers) does not corral developers into an object mindset to the detriment of their applications. This approach avoids tight coupling of distributed components and encourages the development of scalable Grid applications. It also reduces the fragility of distributed applications as consumers and services are not encouraged to bind directly to resources behind the service boundary (thus creating cross-organisational dependencies).

### 3.3. State: Internal and Interaction

The use of state, transient or persistent, is critical to the operation of most useful Web Services, and of paramount importance to most e-science and e-commerce applications. In a service-oriented architecture we identify two types of state: *service state* (state that exists within the service's back-end – the “Resources” of Figure 1) and *interaction state* (the state of an ongoing message exchange, or conversation, with a service).

#### 3.3.1. State Internal to a Service

Since services are autonomous entities with explicit and well-defined boundaries, any state they maintain and use internally for their operation is encapsulated away from their consumers. As shown in Figure 1, a service may maintain any number of resources, like databases, computational nodes, devices, people, etc. that constitute the service's internal state. Encapsulation gives service implementers the freedom to evolve or completely change the structure of a service's internal resources without affecting any applications that consume it.

#### 3.3.2. Interaction State

There are situations where an uncorrelated request-response message exchange pattern is not sufficient to build the required functionality for a distributed application. To support interactions between services that span to more than one request/response message exchange, knowledge about the state of those interactions has to exist. Furthermore, interactions do not have to be restricted to one consumer and one service. Instead, multiple parties may participate in the same extended interaction (or *activity*). Specifications like WS-Coordination [20], WS-AtomicTransaction [21], OASIS WS-ReliableMessaging [22], OASIS WS-CAF [23], OASIS BPEL [3] all use message correlation as the “*means of associating a message within a specific conversational context*” [12]. The Web Services Architecture document explains message correlation as follows:

*“Message correlation allows a message to be associated with a particular purpose or context. In a conversation, it is important to be able to determine that an actual message that has been received is the expected message. Often this is implicit when*

*conversations are relayed over stream-oriented message transports; but not all transports allow correlation to be established so implicitly.” [12]*

In the general case, interaction state is maintained in Web Services using *contextualisation*, which is based on the transmission of a *context* (usually in a SOAP header) along with the application message (i.e., information that participants in an activity can use to refer to the same shared interaction)<sup>1</sup>. The combination of context plus application message allows the recipient of the message to correlate the application-specific payload with the correct back-end resources or processes.

## 4. Infrastructure Requirements for Grid Applications

Building distributed applications is an inherently complex task, and there are many issues that architects and developers of Grid software will have to deal with. These include, but are not limited to, security, workflow, transactions, notification, naming, discovery, and others (for a description of the high-level service requirements that are part of the Open Grid Services Architecture the reader is referred to [24]). In this section we concentrate on those requirements that are commonly considered when building Grid applications:

- Resource identification
- Metadata
- Lifetime management, for
  - resources
  - metadata information, and
  - interactions
- Stateful interactions
- Service deployment

In the following sections we examine these requirements and discuss the underlying motivations for them.

### 4.1. Resource Identification

In a Web Services environment, we cannot use the address of a service to identify a resource, since not only may a service host many resources, but those resources are strictly private to the service. However, while the canonical form of data interchange for a Web Services-based application is the sending and receiving of rich, descriptive XML messages, there are situations (for example in the case of services that provide access to databases) 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.

In such circumstances it is preferable to identify the resource through a long-lived name, which is independent of the activity which caused its creation, and so can be shared by numerous applications. This is useful, for example, where a name which logically identifies the data can be propagated around an application, rather than the data itself<sup>2</sup>, and allows other applications to operate on the same named logical resource.

Resource identification should not be confused with correlation mechanisms for modelling distributed units of work. Resources provide names which may exist outside the scope of a particular stateful interaction, including the interaction which caused its creation, and merely allow different parts of an activity or disjoint applications to reason about the same logical entity. Furthermore, though resource identifiers could be scoped globally the resolution of an identifier into something

---

<sup>1</sup> BPEL is an exception since message correlation is achieved explicitly through the use of specific information in the messages (akin to database keys) rather than through a general, context-based solution.

<sup>2</sup> Note that we do not believe it to be good software practice to directly expose physical resources to the network; instead the resource names will more often than not identify logical resources (e.g., a “bank account” as a logical entity rather than the database table that stores the information for a particular bank account).

meaningful is service-specific. That is, how a service uses a resource identifier is left for the service designer to decide.

## 4.2. Metadata

Long-lived identifiers for logical resources may be a valid requirement for Grid computing where instruments, results, computational elements and so on usually outlive the scope of any individual applications. However, when logical resource identifiers are shared outside an organisation's boundaries, Grid applications may require access to ontologies of resources, information about relationships between resources, resource location information, lifetime information, ownership/access restriction information, provenance, etc. All this information is part of the metadata about those named resources.

We believe that metadata information, like resource lifetime, access information, etc. should be represented as a machine-readable document. Having access to metadata about a logical resource allows systems to make informed choices as to how to use those resources. However, we understand that metadata differs between applications and services, and so we maintain that any support for metadata must be generic and extensible to support a wide array of use-cases.

## 4.3. Lifetime Management

We identify three distinct cases where it may be necessary to provide lifetime-related information and/or to allow management of that information:

- **Resource Lifetime.** Once the decision has been made to share identifiers for logical resources outside the boundaries of a service, it may be useful to provide consumers with an indication about the resource's associated lifetime. Since lifetime related information can be seen as part of the metadata of a resource, it can be included in a metadata document for storage and transfer.
- **Metadata Lifetime.** Once metadata information about a named resource is made available, it may be necessary to control the metadata's lifetime, independently of the lifetime of the associated named resource. This is because the characteristics of the resource (e.g., the locations of services which are aware of that resource) change over time and so the original metadata can become stale with respect to the resource. Like resource metadata, this kind of information lends itself to inclusion in a metadata document for storage and transfer purposes.
- **Interaction Lifetime.** A stateful interaction may be established with limited lifetime. The lifetime may be included in the context information carried with each message (whether contexts are implicit within application messages or created as an external entity to augment application-level messages) or maintained at the service controlling the information about the interaction (and made accessible to services that receive contextualised messages).

## 4.4. Stateful Interactions

Like other Web Service-based application, the Grid requires support for stateful interactions between consumers and services, where a sequence of operations are related such that they are seen as a single logical unit of work or an *activity*, as described in Section 3.3.2. Clearly, stateful interactions are at the heart of any serious application since services that are not capable of maintaining information about message exchanges (i.e., services which cannot correlate messages) are of limited applicability.

## 4.5. Service Deployment

One of the perceived gains of Grid computing is the ability to dynamically discover and consume computational resources. It may be desirable to leverage such resources in order to deploy a service, sometimes outside the administrative domain of the hosting environment. In the general case this is not a trivial matter due to issues with security and dependencies on specific re-

sources (e.g., hardware architecture, databases, libraries, etc.). However, high-level services built on the existing Web Services infrastructure could provide such functionality. Such services are necessarily constrained to deal exclusively in terms of message exchanges with other services, and provide appropriate programming abstractions to support this view.

## 5. The WS-GAF Architecture

The Web Services Grid Application Framework (WS-GAF) architecture provides a solution that meets the requirements presented in the previous section while being deliberately minimal by design. We observed that for many of the requirements identified by the Grid community, there exists a protocol or specification or a design pattern to meet that requirement available in the Web Services space. Therefore, to say that WS-GAF has an architecture per-se is a little grandiose, but what WS-GAF does provide is a platform for interoperability for Grid applications by choosing appropriate commodity technologies from the Web Services space, rather than advocating the construction and maintenance of a specialised Grid infrastructure.

### 5.1. Design Goals

The primary influence while developing WS-GAF was to avoid re-working existing Web Services protocols, and thus keep the Grid-specific aspects of the work simple. A pragmatic approach was taken to utilise the work of the Web Services community to do all of the “heavy lifting” while constructing simple XML document schemas and conventions for aspects which are truly unique to Grid computing (such as the naming of logical resources and providing metadata about them).

Our design goals were set with the premise that these would help to keep the overall approach to building Grid applications simple while, at the same time, adhering to the principles of service-orientation. To achieve our design goals, we decided on the following guidelines:

- **Factorisation.** The WS-GAF design consists of separate, orthogonal specifications. This approach is consistent with the current approach in the Web Services world where more but smaller specifications are used, and the notion that protocol stacks are composed from these smaller specifications on a per-application or per-service basis.
- **Use Standard Web Services Technologies and Tools.** WS-GAF does not introduce any extensions to the underlying Web Services infrastructure, hence leverages existing, unmodified industry and open-source tools for the development and deployment Grid applications. This also means that future Web Services protocols can be absorbed by the WS-GAF architecture as they emerge, providing the maximum potential coverage of technologies with minimum impact on the architecture of existing WS-GAF applications.
- **Focus on Application-Level Constructs, not Low-Level Networking.** We suggest that the focus of the Grid community should be on building the higher level services that will realise the vision of Grid computing [24-28]. 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 (as those will be defined by the Open Grid Services Architecture, or OGSA [29]), and delegating the work of creating XML-based network protocols to the Web Services community.

### 5.2. The Stack

We envisage a Grid application framework, which we call the “Web Services Grid Application Framework” (WS-GAF), implemented according to the principles laid out by the Web Services Architecture. Such an application framework is built only on existing specifications, is able to leverage all the available Web Services tools without modifications, and utilises the OGSA-defined services. A stack-based view of the envisaged architecture is presented in Figure 2.

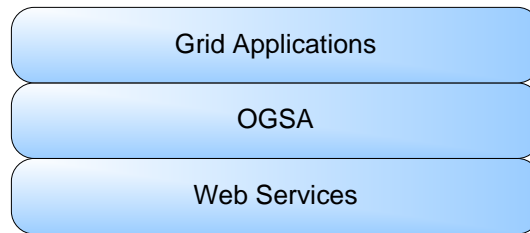


Figure 2: WS-GAF stack

The proposed architecture stipulates that when designing an application or service, a Grid architect could simply select those Web Services technologies that are germane to the particular problem domain.

For an individual Web Service, we expect there to be (at most) three distinct sets of operations comprising the interface of a service.

- **Infrastructure related operations.** These are the Web Services specifications that are used to implement features for security, transactions, coordination, etc.
- **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 (assuming such a platform is WSA-friendly). For example, one such interface may mandate how a service is managed, while another may describe the standardised interface of a specific kind of service (such as a database or specific computational service).
- **Service-specific operations.** The remaining interfaces support service-specific operations connected with the particular application domain problem that the service addresses.

## 6. Meeting Grid Infrastructure Requirements with WS-GAF

In this section we present our proposal for meeting the requirements for building Grid applications as identified in Section 4. The patterns and the specifications suggested in this section are what comprise WS-GAF<sup>3</sup>.

### 6.1. Naming Shared Resources

Once the decision has been made to share the name of a (logical) resource outside a service's boundaries, it is important that the resource is given a unique name with which it can be identified throughout the Grid application. While the name may only need to be unique within the realm of a particular service, if the name of the resource is to be stored in a registry or shared between services, we recommend, as a best practice, the use of globally unique names.

There have been proposals for naming and uniformly providing access to resources, such as the REpresentational State Transfer (REST) [30] model. However, REST depends on a naming scheme for resources which couples the semantics of an interface with the identified state (e.g., `http://domainname/resource-id` suggests that only the HTTP GET/PUT/POST/DELETE operations can be applied on the identified resource). The semantics of the interface are transfer protocol-dependent. We believe that this is unsuitable for heterogeneous systems like the Grid where resources may be accessed/modified by applications in any number of ways – that is we believe that service architects should be given the flexibility to expose their services via arbitrary interfaces.

The Uniform Resource Name (URN) proposal [31] is an existing, protocol-agnostic scheme for identifying resources. URNs *are intended to serve as persistent, location-independent, resource identifiers* [31]. Application domains may choose the schemes that govern their own identifiers (e.g., LSIDs [32]). URNs are independent of any specific distributed computing technology (e.g.,

<sup>3</sup> Note that we are aware that there are other WSA-compliant means of achieving the patterns we present, and that for each protocol we suggest there may be other protocols with similar functionality. However to ease interoperability, we ask the community to indulge us until such point as a stable set of standards, borrowed for example from a WS-Interoperability profile [18], is formed.

CORBA, Web Services, RPC), they are widely adopted and are very simple. Hence, URNs could be used as names for resources.

URNs could typically be used in situations where, for example, a consumer requires that a service returns a URN instead of returning a large set of results from executing a query. This is most appropriate 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 further processing. This pattern is shown in Figure 3, where a URN is used to uniquely identify the named data resource.

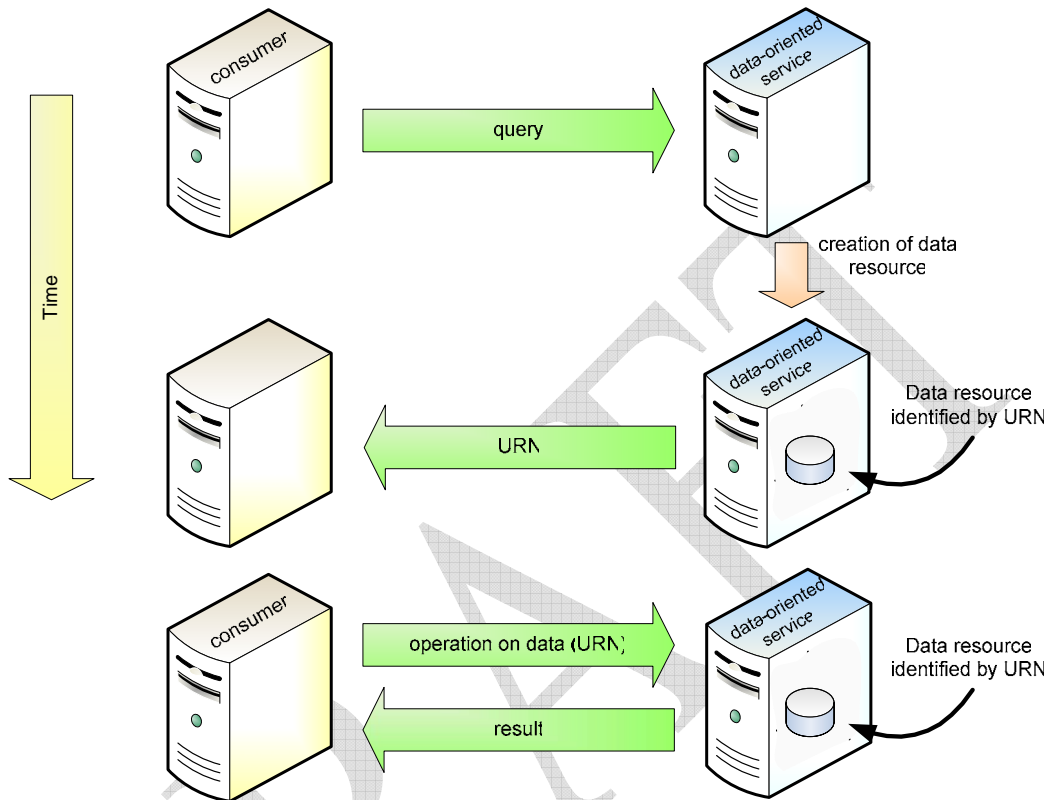


Figure 3: Returning a URN which identifies a (logical) resource

It is important to note that the requirement for naming data is orthogonal to modelling stateful interactions. The URN and context-based solutions demonstrate the separation of concerns and allow significant flexibility. For example, a single piece of data could be accessed by multiple consumers within a single context, or alternatively in a separate context per consumer.<sup>4</sup>

To tailor the URN scheme for the Grid, we propose the Grid Resource Identifier (GRI) which is a URN that uniquely and everlastingly identifies a resource exposed to the Grid. The scheme and the semantics of the information contained in the URN are application specific.

Since the GRI is unique and everlasting, it can be stored in databases and printed in journals, in the knowledge that it could be used at any time in the future to locate any services offering that resource using any network protocol available at the time. In this, it fulfils a similar role to an LSID [32] (and indeed, an LSID can be used as a GRI).

It is important to note that although a resource may have a finite lifetime, its GRI is everlasting, which is an attribute derived directly from the URN RFC [31]. This property allows GRIs to be stored in registries, printed in journals, shared between project collaborators or otherwise used to identify resources ad infinitum<sup>5</sup>.

Examples of GRIs:

<sup>4</sup> For a detailed discussion on the use of contextualisation the reader is referred to [2].

<sup>5</sup> The lifetime of the GRI is infinite which is greater than, or equal to, the lifetime of the resource that it names, which in turn has a greater or equal lifetime than the activity within which the resource and name were created.

```
urn:dais:dataset:guid:b4136aa4-2d11-42bd-aa61-8e8aa5223211
urn:instruments:telescope:nasa:hubble
urn:physics:colliders:cern
urn:lsid:pdv.org:1AFT:1
```

## 6.2. Grid Resource Metadata Document

The Grid Resource Metadata (GRM) document is an example of how metadata information could be captured for those logical resources whose names are shared between services via a Grid Resource Identifier (GRI). A GRM provides an extensible mechanism for the addition of application-specific metadata. The GRM document consists of two parts:

- A core set of metadata information elements that are likely to be common to all named resources:
  - The lifetime of a resource;
  - The location of any services that may be aware of the resource;
  - The GRI and resource type (if known).
- A set of domain-specific metadata entries. The contents of these entries are undefined by the specification as they are domain specific. Examples would be an RDF [33] document describing life-science data, or Dublin Core [34] describing the resource publisher.

Table 1 lists the elements that should or may be included in a GRM document<sup>6</sup>:

Element Name	Cardinality	Description
<b>gri</b>	1	The GRI of the resource with which the metadata in the document is related.
<b>type</b>	0 - 1	In situations where the identified resource could be represented by, or is, an XML document, this element might be used to provide information about the XML type of that document (e.g., the qname of a DAIS [5] dataset).
<b>grm-lifetime</b>	0 - 1	Provides a hint about the lifetime of the GRM document.
<b>resource-lifetime</b>	0 - 1	Provides a hint about the lifetime of the resource identified by the Grid Resource Identifier, the "gri" element.
<b>endpoint</b>	0 - many	Provides information about the endpoint of a service that may be aware of the identified resource. The semantics of what it means for a service to "be aware of a resource" are application-specific. There may be zero to many endpoint elements in a GRM document. An endpoint element consists of the following sub-elements:
<b>lifetime</b>	0 - 1	A hint about the period of time during which the addressed service will be aware of the resource.
<b>endpoint-protocol-information</b>	1	Protocol-specific information about the endpoint. The structure of this sub-element is left undefined. Application domains may use different endpoint-related information, like WS-Addressing [5], IP-port pairs, Database connection string, CORBA IOR, etc.
<b>metadata-entry</b>	0 - many	Contains information about a metadata entry. There may be zero to many metadata entries in a GRM document. Each entry consists of the following sub-elements:
<b>name</b>	1	A name for the metadata entry.

<sup>6</sup> The XML Schema for the GRM document can be found at [35].

<b>modifiable</b>	0 - 1	Suggests whether or not the value for the metadata entry is modifiable.
<b>lifetime</b>	0 - 1	Provides a hint about the period of time the value of the metadata entry would be considered valid.
<b>content</b>	1	The content of the metadata entry. The structure of this sub-element is left open as different application domains are likely to include different metadata-related information.

Table 1: Structure of the GRM Document

The structure of the GRM document is independent of, and so does not address, the means by which such documents are populated, stored, or delivered to interested parties – this is application-specific and therefore out-of-scope for this work.

### 6.3. Lifetimes

WS-GAF treats the lifetimes identified in Section 4.3 separately from one another. The Grid Resource Metadata document contains a number of entries where lifetime related information could be defined, as shown in Table 1. A distinction is made between the lifetimes of the GRM document, of the identified resource, of each service providing access to the resource, and each metadata entry. The semantics of these lifetimes may be different between applications. For example, the lifetime information about a resource may mean that the resource will be destroyed after the time is elapsed but in another case it may mean that the resource may not be cached anymore in a particular location. Hence, the way in which the lifetimes are updated and managed is left to higher-level specifications (e.g., OGSA services) or application-specific mechanisms.

### 6.4. Supporting Stateful Interactions

While in [2] we describe the use of WS-Context [4] as an example of a specification that could support stateful interactions between services, in this document we make the assumption that only specifications from the WS-I Basic Profile 1.0a are used when building Grid applications. Hence, our design pattern uses explicit contextualisation (i.e., the use of information from the body of a SOAP message for message correlation, as in OASIS BPEL) to model stateful interactions between a consumer and a service. Although this approach is as not as flexible as when there is an external entity to model distributed units of work, in many situations it is all that is needed.

Information inside the payload of a SOAP message could be used by services to reason about a particular interaction. For example, the identifier of a session sent as part of the messages' payloads could be used by the service to correlate those messages and treat them as a unit of work, a stateful interaction or a session-based interaction.

### 6.5. Deploying, Redeploying, and Managing Services

One of the main uses of the Grid is the discovery and use of distributed computational resources that can be used to deploy and then consume services (security and policy permitting). Given that a service is a logical manifestation of physical resources (databases, programs, computers, humans, etc.) the notion of “deploying” a service, while appealing in the abstract, is practically difficult when organisation boundaries have to be crossed, unless the deployment circumstances can be constrained.

We believe that dynamic deployment capabilities should be defined by a high level service, rather than the underlying Web Services infrastructure. For example, with the advent of technologies like BPEL [3] 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 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 4.

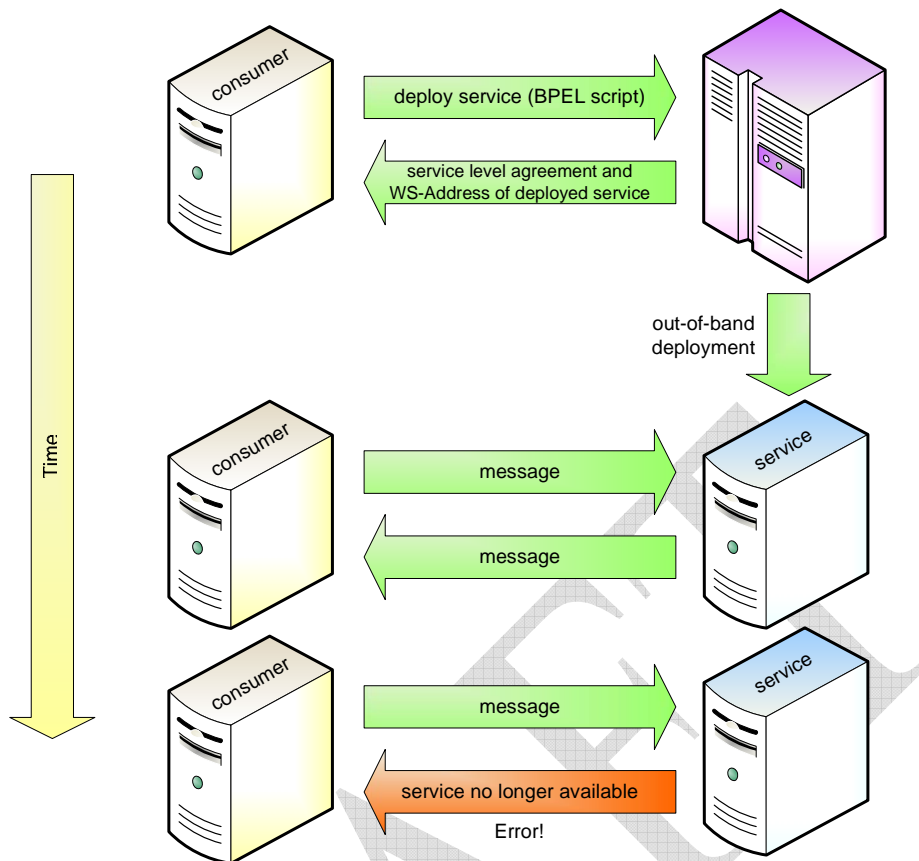


Figure 4: Dynamically deploying a service implemented with a constrained programming language

In Figure 4 an application first deploys its BPEL-encoded logic to a BPEL execution engine, and receives in response a service level agreement which might describe (for example) time to live or number of invocations allowed on the deployed BPEL 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 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 or Web Services community will define the requirements, functionality, and interface of high-level services enabling dynamic deployment and management of services. Issues like the description of resources and their discovery, security and policies, service level agreements are already addressed by Global Grid Forum [36] working groups. The outcome of these working groups could be used in the definition of a service enabling dynamic deployment of other services.

While it is conceivable (though difficult and ill-advised) that the community will define the necessary constraints for enabling the dynamic deployment of binary-level services, because of the generality of such languages the constraints placed on such deployments will be necessarily severe<sup>7</sup>.

## 6.6. Security

In service-orientation where interactions are facilitated through the exchange of messages and where no assumptions are made about the characteristics of the underlying communications infrastructure it is necessary to introduce message-level protocols in order to meet any security requirements. The OASIS Web Services Security standard [37] defines how SOAP messages can

<sup>7</sup> We assume these kinds of constraints would be similar to those enforced by stored procedure execution managers in databases.

carry security related credentials for authentication and authorisation purposes and how SOAP messages can be encrypted and signed.

## 7. Examples

In this section we present examples of how the proposed Web Services Grid Application Framework could be used to implement Grid applications. The focus is on naming resources, the use of those names in message exchanges, and metadata about the named resources. We use services that encapsulate data resources as a basis for our examples.

In our example, a “data-oriented service” is a Web Service that provides database-like functionality. Leaving aside issues related to security for which emerging specifications, like WS-Security [38], provide a solution, we assume that the service permits the creation, modification, deletion, and querying of databases using standard SQL [39]. The implementation details of the underlying databases are hidden and the service could be supported at the back-end using flat files, an open-source or a commercial database management system, or even by delegating to other services.

### 7.1. The Use of GRIs – Data-Oriented Service

The Grid Resource Identifier (GRI) provides a means of attributing a logical name to some structured data or other resources. Where in the previous section there was no need to provide a name for the data-related resources outside the boundaries of the service, in this section we explore possible usage patterns where long-lived names are bound to logical resources.

A simple stateless interaction with a data-oriented service is shown in Figure 5. A consumer sends a query to the data-oriented service but instead of receiving a dataset document it receives a GRI. 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 knew by some out-of-band mechanism that the result for the query was a pre-existing dataset identified by the returned GRI); or, the service was asked to directly deliver the resulting dataset to a data-processing service using a faster communications infrastructure. It is at the discretion of data-oriented service specifications or implementers to decide where the use of GRIs is appropriate.

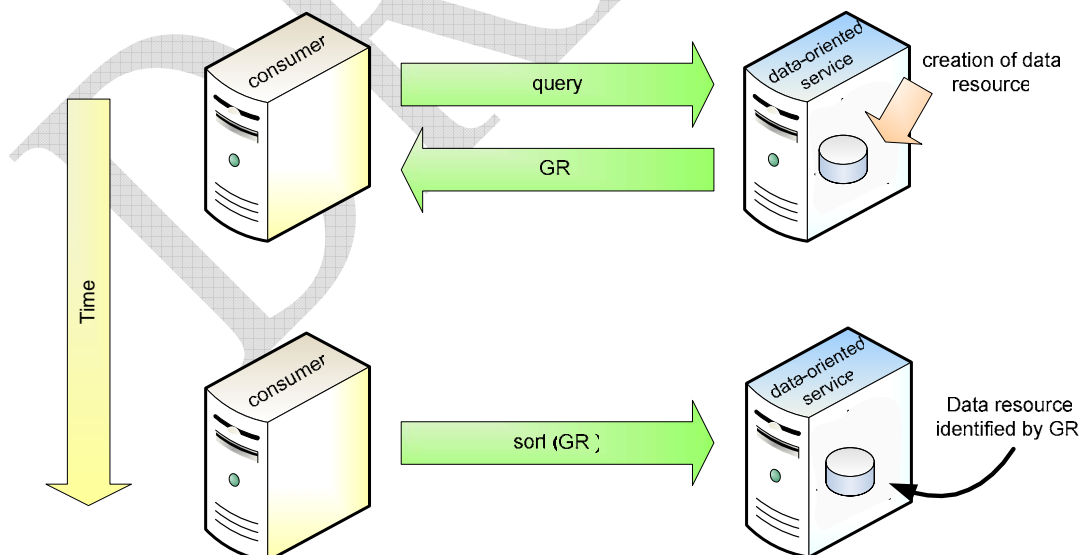


Figure 5: A simple interaction with a data-oriented service

As we suggested earlier, another way to use the GRIs which we expect to be a common pattern, is to just refer to named resources; for example, when a consumer wants to inform other services about a data resource that it has created, when services need to reason about a resource by

name rather than by value, when a resource must be referenced in a printed publication, or its name stored in a database, etc. (Figure 6).

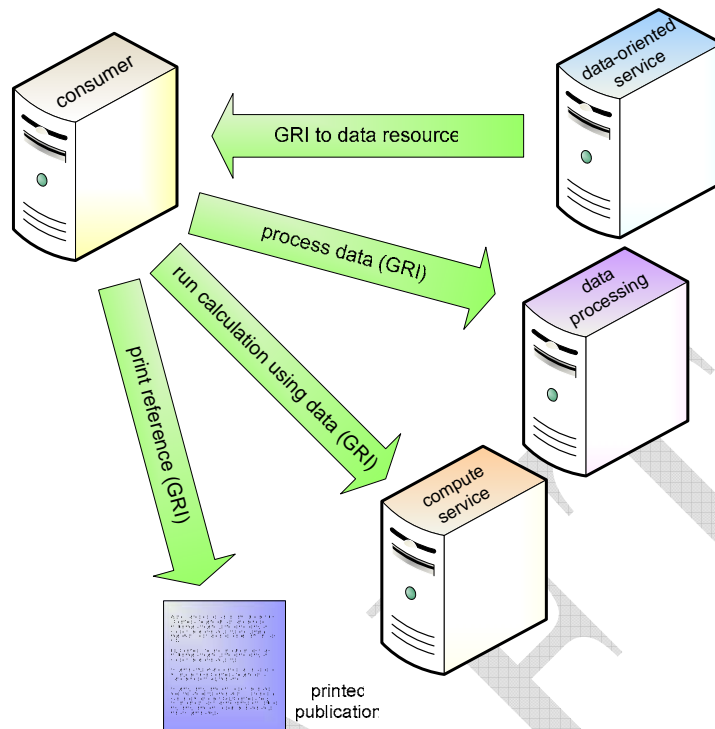


Figure 6: A GRI is used to refer to a named resource

A Grid application may utilise GRIs as part of an extensible context structure, to model stateful interactions with named resources (and indeed plain SOAP headers and WS-Context support this implicitly). Whether a service architect chooses to transport this data as part of an interaction context, or as an application-level “parameter” is out-of-scope for WS-GAF since it is only the service architect, who has domain-specific knowledge, that is in a position to make such judgments. However architects choose to design their Web Services, the point is that GRIs are independent of any particular implementation style but compliment the general style and architectural nuances of WSA Web Services.

It is important to note that the scenarios presented in this section correspond only to a small subset of the possible ways a GRI could be used, and are meant merely to illustrate possibilities rather than provide a normative definition. Again we promote the notion of service architects and developers making informed choices about their specific services, and mandate only that such services align with the principles outlined in WSA.

## 7.2. Metadata about Referenced Resources

Grid Resource Metadata (GRM) documents provide a general way of recording information about the available services and their interfaces from which the referenced resources may be accessed and other metadata information pertinent to that resource.

In this section we present an example that utilises the GRM document to retrieve information about a named (logical) resource.

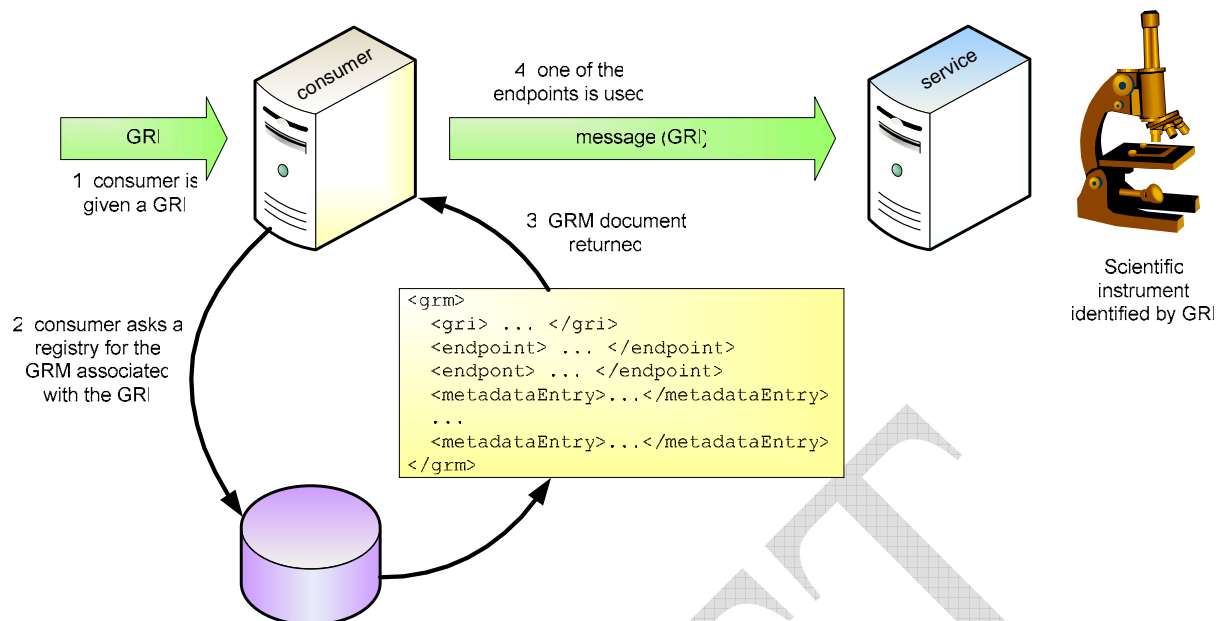


Figure 7: Using the GRM document to locate a service via a GRM-aware registry

In Figure 7, a consumer receives a GRI (e.g., from a database, from another service, from a user, from a journal) that identifies a logical resource, in this case a particular scientific instrument. The consumer does not know the service that currently manages the named resource so it contacts a registry service<sup>8</sup> to get access to a GRM document associated with the given GRI. The returned GRM document contains information about the services that provide access to the resource – protocol-specific information about how the resource could be reached (e.g., WS-Addressing, CORBA IOR, TCP/IP-port number pair, etc.).

The GRM document therefore decouples the resource name from the protocol specific information about how it can be accessed, so supporting resources that are offered by more than one service and cases where the set of services offering a resource changes over time. Since the GRI is based on the URN, it is platform-independent and so it could be printed in a journal in the knowledge that a scientist in the future could use a registry to fetch a GRM document containing information on how to access that resource using whatever distributed systems technology is being used at that time. The GRM document may also contain additional information about the resource, like its capabilities, its expected usage lifetime, its owner, etc. The consumer parses the information and makes a decision on whether to use the identified resource and which services it will use to perform the necessary operations on the named resource.

Services may require that their consumers provide them with a GRM document instead of only a GRI. In such cases, a registry may not be required. Reflecting the data-oriented service example of the previous section (Figure 6), we can see how a data-processing service may require a GRM document from its consumers with metadata information on how to retrieve the referenced data. In the example of Figure 8, a data processing service requires that the GRM document with information about a resource is sent to it directly. From the information contained in the GRM document, the data processing service can determine the necessary information, instead of having to look into a registry, about how to establish a fast file transfer connection with the data-oriented service that holds the referenced data resource.

<sup>8</sup> A registry is not the only means of accessing a GRM document. Other valid means might be a database query, or a search of a peer-to-peer network.

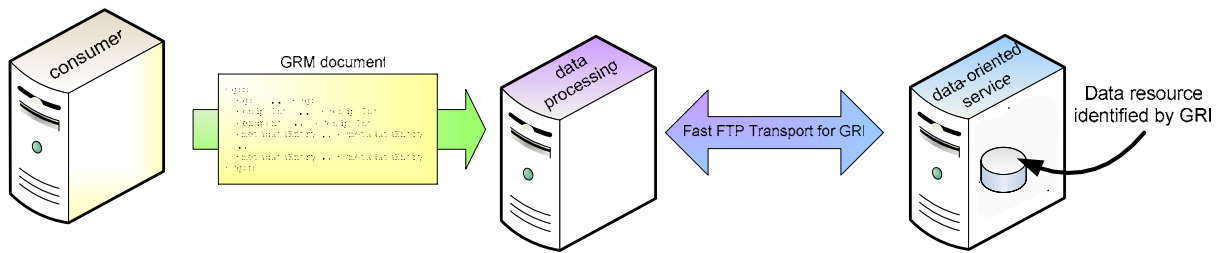


Figure 8: A GRM document conveying information about a suitable service with a fast interconnection

### 7.3. Stateful Interactions

We expect stateful interactions between services to be modelled using an external entity, like WS-Context [4]. However, until the industry agrees on a standard specification and product-quality, well-supported toolkits become available, we recommend the use of application domain-specific solutions. Such approaches would include information in the body of the SOAP messages exchanged in order to identify a stateful interaction.

For example, in order to model a distributed query amongst a number of data-oriented services, a coordinator could send a query identifier (a GRI) to the services with every message. In this way, all the services participating in the same query can have a common understanding about it, as shown in Figure 9. How the services utilise that information and how they reason about the query is left up to them.

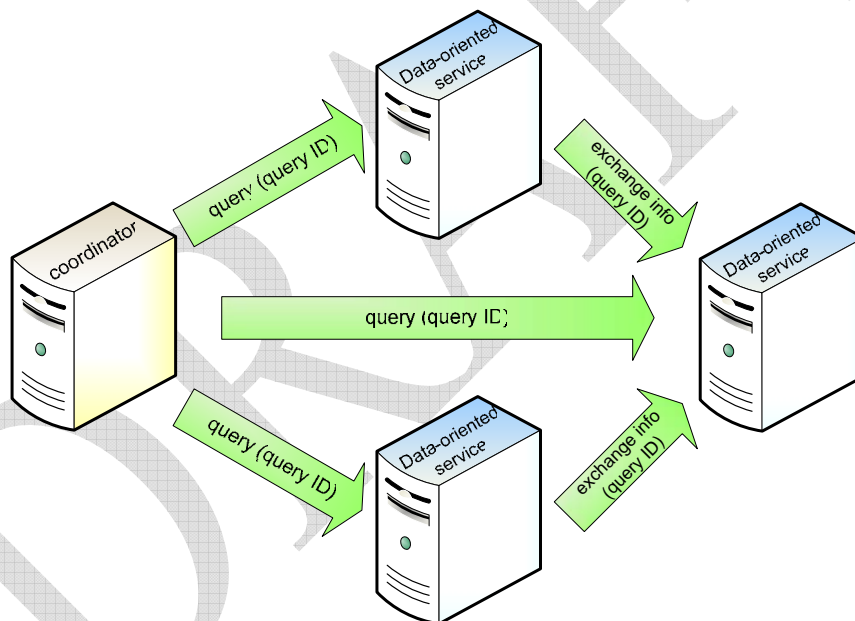


Figure 9: A coordinator issues a distributed query to data-oriented services

## 8. Conclusions

There is a great deal of work that has been and continues to be undertaken by the Web Services community that could be used to implement Grid solutions (e.g., in the areas of security, coordination, transactions, business process orchestration, notification, etc.). Utilising such work allows Grid developers to exploit existing Web Service tools, specifications, services and practices, educational material, and hence avoid the need to build a parallel set of solutions. This frees the Grid community to concentrate on building the higher-level abstractions (i.e., service interfaces and document structures) that are specific to building Grid applications.

This document has proposed a design pattern for a Web Services Grid Application Framework that is, we believe, simple and compliant with specifications and practices in the wider Web Ser-

vices community. The proposal only uses specifications from the Web Services Interoperability Basic Profile and the OASIS WS-Security standard.

To allow orthogonal access to resources, the WS-GAF proposal introduces the concept of a Grid Resource Identifier, which is just a URN that uniquely and everlastingly names a (logical) resource and the Grid Resource Metadata document, which is an extensible placeholder for recording metadata about those (logical) resources. The service interactions can deal in terms of high-level names which can be resolved to something meaningful by applications that create and consume those names. We believe that the separation of issues of identity and metadata from services and the provision of distinct solutions allows for the implementation of a more flexible architecture, the adoption of existing tools, and the support for sophisticated usage patterns.

To conclude, this document has presented our views on the relationship between Web Services and the Grid, and proposed a framework that is in line with wider Web Service specifications and practices. It is not our intention to propose this Grid Application Framework as the only possible way Grid applications could be realised. Instead, we have demonstrated that simple design patterns and the use of existing Web Services specifications give us powerful tools for building loosely-coupled, scalable, organisation-to-organisation Grid applications.

## 9. Acknowledgements

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

Prof. Pete Lee (University of Newcastle upon Tyne), Mark Baker (individual contributor), Tim Banks (IBM), Dave Ingham (Arjuna Technologies), Dr. Mark Little (Arjuna Technologies), Dr. Jim Smith (University of Newcastle upon Tyne).

## 10. References

- [1] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "A Grid Application Framework based on Web Services Specifications and Practices." <http://www.neresc.ac.uk/ws-gaf>, 2003.
- [2] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "WS-GAF: A Grid Application Framework based on Web Services Specifications and Practices." Submitted for publication, 2004.
- [3] "OASIS Web Services Business Process Execution Language." <http://www.oasis-open.org/committees/wsbpel>.
- [4] OASIS(WS-CAF), "Web Services Context (WS-CTX)." <http://www.iona.com/devcenter/standards/WS-CAF/WSCTX.pdf>.
- [5] "Web Services Addressing (WS-Addressing)." <http://msdn.microsoft.com/ws/2003/03/ws-addressing>.
- [6] OASIS, "OASIS Web Services Business Process Execution Language." <http://www.oasis-open.org/committees/wsbpel>.
- [7] Apache, "Web Services Functionality Extensions (WS-FX)." <http://ws.apache.org/ws-fx/>.
- [8] OMG, "CORBA/IIOP Specifications." [http://www.omg.org/technology/documents/corba\\_spec\\_catalog.htm](http://www.omg.org/technology/documents/corba_spec_catalog.htm).
- [9] W3C, "Extensible Markup Language (XML)." <http://www.w3.org/XML/>.
- [10] W3C, "XML Schema." <http://www.w3.org/XML/Schema>.
- [11] W3C, "SOAP Version 1.2 Part 1: Messaging Framework." <http://www.w3.org/TR/soap12-part1>.
- [12] W3C, "Web Services Architecture." <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, 2004.
- [13] "Service-Oriented Architecture (SOA) Definition." [http://www.service-architecture.com/web-services/articles/service-oriented\\_architecture\\_soa\\_definition.html](http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html).
- [14] H. He, "What is Service-Oriented Architecture." <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>, 2003.
- [15] W. Vogels, "Web Services Are Not Distributed Objects," *IEEE Internet Computing*, vol. 7, pp. 59-66, 2003.

- [16] D. Box, "Service-Oriented Architecture and Programming (SOAP) - Part 1 & Part 2 talks from the MSDN TV archive  
<http://msdn.microsoft.com/msdntv/episode.aspx?xml=episodes/en/20030827SOAPDB/manifest.xml>,  
<http://msdn.microsoft.com/msdntv/episode.aspx?xml=episodes/en/20030902SOAPDB/manifest.xml>,"  
 2003.
- [17] W3C, "Web Services Description Language (WSDL)." <http://www.w3.org/2002/ws/desc>.
- [18] WS-I, "Web Services Interoperability (WS-I) Interoperability Profile 1.0a." <http://www.ws-i.org>.
- [19] S. Baker, "Web Services and CORBA," presented at On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE, Irvine, California, USA, 2002.
- [20] "WS-Coordination." <http://msdn.microsoft.com/ws/2002/08/WSCoor>.
- [21] "WS-Transaction." <http://msdn.microsoft.com/ws/2002/08/wstx>.
- [22] "OASIS Web Services Reliable Messaging." <http://www.oasis-open.org/committees/wsrn>.
- [23] "Web Services Composite Application Framework (WS-CAF)." <http://www.iona.com/devcenter/standards/WS-CAF>.
- [24] I. Foster and D. Gannon, "Open Grid Services Architecture Platform (OGSA)," <https://forge.gridforum.org/projects/ogsa-wg>, 2003.
- [25] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1998.
- [26] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum 22 June 2002.
- [27] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, pp. 42-47, 2002.
- [28] F. Berman, G. Fox, and T. Hey, "Grid Computing: Making The Global Infrastructure a Reality," John Wiley & Sons, 2003.
- [29] GGF, "OGSA Working Group," <https://forge.gridforum.org/projects/ogsa-wg>.
- [30] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.
- [31] R. Moats, "RFC 2141: URN Syntax." <http://www.ietf.org/rfc/rfc2141.txt>: IETF, 1997.
- [32] "I3C Life Sciences Identifiers (LSIDs)." <http://www.i3c.org/wgr/ta/resources/lsid/docs/index.htm>.
- [33] "Resource Description Framework." <http://www.w3.org/RDF/>.
- [34] <http://dublincore.org/>, Dublin Core Metadata Initiative.
- [35] S. Parastatidis, P. Watson, and J. Webber, "Grid Resource Specification." <http://www.neresc.ac.uk/ws-gaf>, 2003.
- [36] "Global Grid Forum." <http://www.gridforum.org>.
- [37] OASIS, "OASIS Web Services Security." <http://www.oasis-open.org/committees/wss>.
- [38] "OASIS Web Services Security." <http://www.oasis-open.org/committees/wss>.
- [39] *Data Management: Structured Query Language (SQL), Version 2*: The Open Group, 1996.