GWD-I (draft-ggf-ogsi-gridserviceprimer-1)
Open Grid Services Infrastructure (OGSI)
http://www.ggf.org/ogsi-wg

Editors:
Tim Banks

June 5th, 2003

Open Grid Service Infrastructure Primer (Draft)

Abstract

This is a working draft of the OGSI Primer. The aim is to provide an introduction to the OGSI specification with progressive examples, but without relying on any particular implementation.

The Primer is a non-normative document, which means that it is not a definitive (from the GGF's point of view) specification of OGSI, intended to provide an easily readable description of OGSI. It is intended towards quickly understanding the basic fundamentals for creating OGSI-based grid services. The examples and other explanatory material in this document are here to help you understand OGSI, but they may not always provide definitive or fully-complete answers. In such cases, you should refer to the relevant normative parts of the Grid Service Specification which can be found on the OGSI working group **web site**.

Making comments and contributions.

The OGSI Primer is a being produced by the OGSI working group of the GGF. The **web site**[1] of the working group lists its charter, mailing list and previous products, including the draft of the OGSI specification for which the Primer will act as an introduction.

The Primer will be developed during 2003 and during its development comments from the OGSI working group are encouraged. You should join the working group by introducing yourself via the mailing list. All that's needed is a note to say 'hello' and explain your interests and background. If you want to make comments on the Primer, the mailing list is the way to gain the authors' attention.

You can put you comments directly on the mailing list, or use the GridForge web site.

[1] The GridForge web site at https://forge.gridforum.org/projects/ogsi-wg is being developed – it may soon be a better starting point to make comments on the Primer.

Contributions to the document are coordinated via the mailing list and associated conference calls.

Current Status and future schedule.

*This is **work in progress!*** This draft was produced in preparation for discussion at the GGF8 meeting in Seattle.

Ideas and comments on the structure of the document are encouraged. Chapters 1-3 and parts of chapters 4-6 are in a reviewable state and an 'incremental improvement' process based on review and comments is appropriate. However, beyond that, the text consists of material taken from the OGSI specification, together with questions and conversations from sources such as the OGSI-WG mailing list. It has not yet been reworked into a coherent form.

Input from the GGF meeting, together with comments from the mailing list and working draft will be incorporated during the remainder of 2003 with the target of producing the final document at the end of 2003.

Contents

Table of Figures

1 About this Primer

This chapter introduces the structure of this document its intended audience and other material which might be relevant.

1.1 Who should read this Document?

This is an introductory document to the Open Grid Services Infrastructure (OGSI) specification which is aimed at a wide audience of architects and developers, implementers and users.

No prior knowledge of the Grid or Web Services is assumed, though an awareness of distributed computing will help. If you need extensive background material on grids, *The Anatomy of the Grid* (see [1] in the 'References' section) should help. However,

- Chapter 2 summarizes the background of grid computing. Anyone interested in gaining an overview of the concepts and major features of services which conform to the OGSI specification might start at chapter 3.
- Section 3.1 gives an overview of Web Services whose specifications and technology form the basis for OGSI. It describes some major mechanisms underlying web services and introduces terminology that will be used elsewhere in the primer. It also contains a simple example of a web service. If you are already familiar with Web Services you may want only to look at the example.
- Section 3.2 and chapter 4 describe the concepts which are introduced by OGSI. The functions required in the Grid which are additional to Web Services.
- Chapter 5 explains how these services can be constructed in real implementations.
- Chapter 6 introduces large scale features which are enabled by the form of the individual grid services. These are the features necessary to create large-scale applications and systems.

More detailed information is needed for implementers of Grid-based services or client systems which need to call them. They might start with this document before going to read documentation for platform-specific tools and interfaces.

- From chapter 7 onwards, the details of Grid services interfaces are explained in a way which parallels the Grid specification. This makes it easy to correlate the information in the two documents.

Examples are used throughout the Primer to illustrate the features which the Specification requires. The examples are deliberately simple, to avoid any need for knowledge of more realistic, but complex, Grid applications. Also, though they contain correct interface definitions for the functions described, implementations are not provided. This is because the Specification aims to define a standard for interoperability while leaving freedom to implement Grid services in a wide variety of ways. For details of implementation, you

should investigate one of the Grid service toolkits such as GT3[10] and identify sample services which correspond to the ones described here.

1.2 Related Documents

This document is a companion document to the Open Grid Service Infrastructure (OGSI) specification [3]. The Specification is the complete and authoritative description of Grid Services and should always be used to resolve questions of detail or ambiguity.

This document is, as far as possible, a self-contained introduction to the concepts of Grid Services and their main features which should quickly provide insight into the way Grid Services can be used and operated. Some of its sections are aligned to corresponding topics in the OGSI specification, but it has more introductory material and uses illustrative examples which are not constrained by the need to be complete in detail, though this is also a goal wherever possible.

2 Grids and their Requirements

2.1 What are Grids?

Grid computing is way of organizing computing resources so that they can be flexibly and dynamically allocated and accessed, often to solve problems requiring many organizations' resources. The resources can include central processors, storage, network bandwidth, databases, applications, sensors and so on. The objective of grid computing is to share information and processing capacity so that it can be more efficiently exploited. This advantage is clearest when the need for resources is unpredictable, short-term, or changes quickly or where it is simply larger than any single organization's capability to provide it. Some kinds of problem which might take days to solve on a single installation's resources can be reduced to a few minutes with the right kind of parallelization and distribution of the task. This reduction in turn-around time has opened up areas and styles for computing applications which have previously been impractical.

2.2 Why a Service-Oriented view of grids?

Inevitably, the use of widely distributed, resources requires a connecting architecture which is universal; it must allow the use of heterogeneous resources which have the minimum of constraints on their hardware and software platforms. For this reason, the connectivity is defined in terms of the sequence network messages that an application or resource and its consumers must exchange. There is no need to say what operating system, programming language or interfaces should be used to create the messages, and none of these things is part of the OGSI specification. Any hardware platform or application container will do, provided the messages are correctly constructed.

The term **service** is used to describe a network-enabled entity that provides a specific capability, for example, the ability to move files, create and execute processes, or verify access rights. A service is defined in terms of the messages one uses to interact with it and the behavior expected in response. This behavior may depend on the **state** of resources such as files, databases or sensors which are encompassed by the service and the state may change in response to messages from one or more clients, or on internally generated events such as timers or external physical events. The messages, state and other behavior must be described by a **service definition.**

A good service definition permits a variety of **implementations.** For example, an FTP server speaks the FTP Protocol and supports remote read and write access to a collection of files. One FTP server implementation may simply write to and read from the server's local disk, while another may write to and read from a mass storage system, automatically compressing and uncompressing files in the process. If variants are

possible, then discovery mechanisms that allow a client to determine the properties of a particular implementation of a service are important.

From the client's point of view, an important aspect of the implementation of a service is the **protocol** which is used to communicate requests. A well-constructed server side implementation may permit several protocols. For example, http and https can both be used to retrieve Web pages from Web servers, but the latter is preferable if security is important.

The term *Web Services* describes an important distributed computing paradigm that focuses on simple, Internet-based standards (e.g., eXtensible Markup Language: XML) to enable heterogeneous distributed computing. Web Services define a technique for describing definitions of software components to be accessed, protocols for accessing these components, and discovery methods that enable the identification of relevant service implementations. Web services are programming language-, programming model-, and system software-neutral.

2.3 The Scope of OGSI

OGSI describes the services of grid services using Web Services as a basis. This means exploiting

- The mechanisms for encoding message transmission protocols which are described as **bindings** in Web Services. There are many ways of encoding messages and Web Services separates these concerns from the XML definitions of application interfaces: OGSI standardizes interfaces at the application level, independent of the bindings.
- The conventions used in Web Services to separate the primary application interfaces (function calls and their parameters) from functions often managed by standardized, generic middleware. This includes issues as authentication and access control for an operation or isolation of an operation from the effects of concurrent calls from other clients.
- The techniques used in Web Services to separate service- and network-management functions from the application interface. The management issues include Workload balancing, performance monitoring, and problem diagnosis.

Grids may require techniques which are developments of those used in Web Services because of their large scale, scope and dynamic nature. These techniques are described in documents which build on the basic service definitions of OGSI and the assumed standards and techniques of Web Services.

[Need a reference to the Physiology paper and OGSA]

2.4 What is needed to use Grid Services

[Introduction to what kind of server platforms exist, and what do they provide, what does the user provide.]

[What's needed on the client side. Tools and what they do]

3 Background Technologies

3.1 Web Services Introduction

This chapter provides a high-level overview of Web Services. It necessarily leaves out much detail but includes descriptions of those aspects of Web Services which are extended by OGSI and illustrates them via a simple example.

[Need a reference to more detailed material for those that are interested.]

3.1.1 The Case for Web Services

Web services are centered on the service definitions and the messages, encodings and protocols used between clients and servers rather than the application interfaces (APIs) which clients and servers use to manipulate them; this was a difficulty with older technologies such as CORBA [reference?] even though they recognized the importance of protocols, these systems were built "APIs first" and the interoperability goal remained elusive.

Web services build on a set of well established technologies and protocols. For example, service descriptions are written in XML and data interchange formats can be both expressed in and transmitted as XML so tools for describing the format of XML documents, for creating them and parsing them can all be exploited by Web services. Also, http is used as a transport protocol which is well understood, has many implementations and is widely deployed with trusted security features and configurations such as firewalls. All this has made the adoption of Web services much easier and consequently more widespread.

The standards that define Web services are structured and extensible. The ability to use simple, widely accepted SOAP and http, where appropriate, is preserved while enabling replacement, enhancement and optimization if necessary. This lowers the barrier for adoption for many users who might view the more complex systems as too difficult. This also lowers the barrier to implementation for providers and developers. All of this contributes to the popularity of Web services.

A consequence of the focus on heterogeneity and web-wide scope (as opposed to enterprise-wide scope) of Web services is that investment in development of services can be exploited more easily and more widely. This aspect is also important for investment in resource which can be exploited via a computing Grid.

3.1.2 Web Services Overview

The term Web service is evolving as new standards and technologies are being defined. Many different vendors tout new Web service technologies etc. For the purposes of this primer we will define a Web service as something that can be described using the Web Services Description Language (WSDL)[4]. The next section describes the content and structure of WSDL documents.

The use of this language enables exploitation of other [references?] standards and their implementations which provide services for publication of the existence of a service. Such publication enables the services to be discovered and subsequently invoked by a client using a binding (protocol) that is mutually acceptable. Publication, dynamic discovery and invocation of services is a essential feature of grid computing. Figure 3-1 summarises these fundamental aspects of Web services: here the services are shown as applications, data and the resources needed to process and store them.

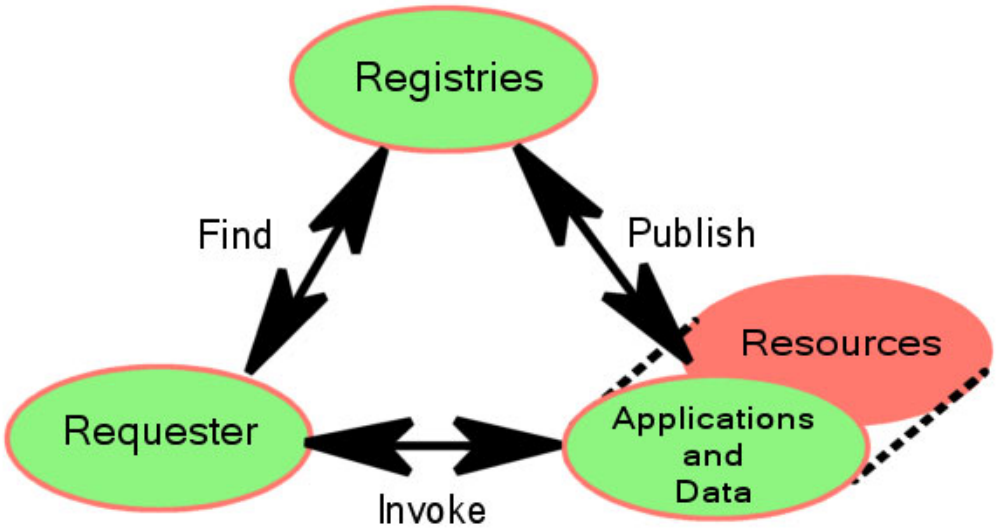

Figure 3-1: The basis of Web service computing

The mechanisms needed by the client to use services and by the server to implement them are the same as in other distributed architectures: a WSDL compiler takes the service description and generates code that translates WSDL concepts into concepts specific to the client's and server's implementation. These mechanisms are described in section 5 (Implementing Web and Grid Services).

3.1.3 Web Services Description Language (WSDL)

WSDL is an XML document style for describing service definitions. Figure 3-2 summarizes the markup language for Web Services description. The root XML element called the wsdl:definitions is on the left of the diagram. It may contain subsiduary elements, shown via the 'contains' linkage, such as wsdl:import and wsdl:types. Subsidiary elements can be repeated as shown by the range indicator; for example all subsidiary elements of wsdl:definitions can appear any number of times (0..*).

The diagram is divided into three vertical sections.

- The elements in the top section allow imports from external files and definitions of datatypes for use in subsequent elements.
- The middle section is the abstract definition of the service interface its operations and their parameters. These definitions determine what a client of the service can do.
- The bottom section is defines the binding(s) of the abstract definition to concrete message formats, protocols and endpoint addresses through which the service can be invoked.

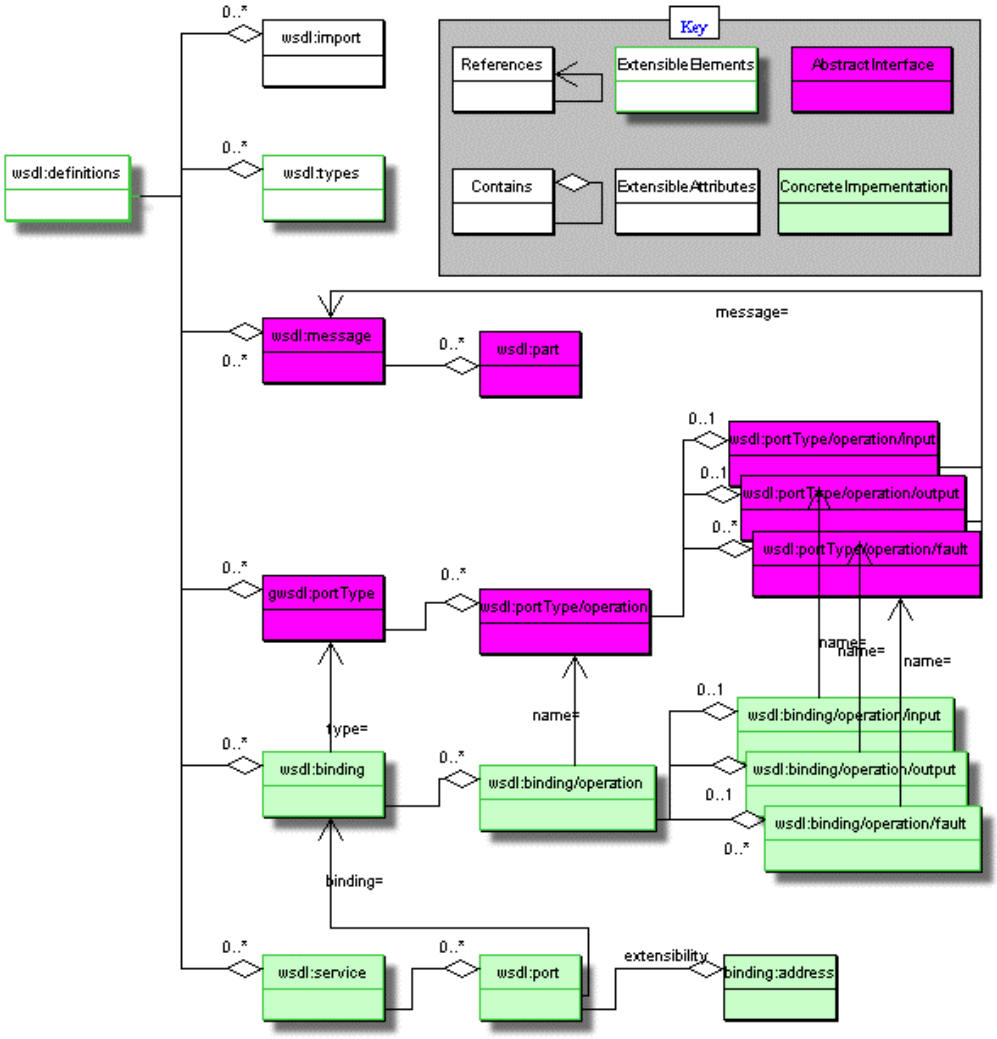

Figure 3-2: WSDL markup elements

At the heart of a WSDL description is the **portType** definition. It is a concept similar to a Java interface or a C++ class. A portType may contain several operations (similar to Java or C++ method signatures) which reference (shown in the diagram as arrows) messages to describe their inputs, outputs and faults (exceptions). A message is composed of many parts where each part can be of different types. The message parts can be thought of as input and output parameters; a message simply combines them.

The types of message parts are defined within the `types` component of `WSDL:definition`. This element is *extensible,* meaning that it can contain arbitrary subsidiary elements to allow general data types to be constructed. The default type system in WSDL is XML Schema, which we won't discuss here in detail. In order to understand our example it is sufficient to know that at the end of the definition we link the type names of message parts to their XML definitions. WSDL allows the other elements to contain subsidiary extension elements means of allowing description of very al interface types; these extension points are shown in the diagram. The purpose of other extensions is decribed in appendix A.3 of WSDL[4].

Another important part of the WSDL definition is the binding. It describes the concrete implementation of messages: that is a data encoding, messaging protocol, and underlying communication protocol. The XML elements of a binding are operations which reference the corresponding abstract messages by name. An important aspect of WSDL bindings is their capacity for extension with new subsidiary element definitions. WSDL allows any and all data encoding, message and communications protocols to be described. The usefulness of a particular description depends on having client and server systems which are able to interpret the WSDL descriptions to encode/decode the messages.

[There was a discussion of bindings on the OGSI-WG mailing list with the subject line '[ogsi-wg] OGSI (spec draft 29) comments' around May 15th with suggestions for documenting bindings. (see the note from David Snelling on 19/05/2003 at 11:18). References to samples of bindings would be useful at this point.]

One important piece of information in the binding is where the service lives. The port element associates a protocol-specific address (endpoint) hosting the web service to a binding element.

3.2 The Counter Example

This section introduces the terms used by the OGSI specification and their relation to Web Services. The Counter Service example is used as a vehicle for the discussion. The WSDL for a simple Counter Service follows (the bindings and service sections are not presented):

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter"
                  targetNamespace="http://www.gridforum.org/namespaces/
```

```
2003/05/ogsiprimer/counter">
   <wsdl:types>
      <xs:schema/>
   </wsdl:types>
   <wsdl:message name="increaseMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <wsdl:message name="getValueMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <wsdl:portType name="counterPortType">
      <wsdl:operation name="increase">
         <wsdl:input message="increaseMsg"/>
      </wsdl:operation>
      <wsdl:operation name="getValue">
         <wsdl:output message="getValueMsg"/>
      </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

Two operations are available: increase and getValue.

3.2.1 Counter Web Service

In the Web Services world, the Counter Web Service is a software component whose interface is described in WSDL and it is the logical receiver of operations.

The Counter Web Service of Figure 3-3 implements the WSDL interface presented in Section 3.2. The interface does not say anything about the state that must be maintained by the implementation of the Web Service. However, for the Counter Web Service to be useful, state has to be maintained in an implementation-specific way (e.g., in memory, in a database, in a file, etc.), making it a stateful Web Service. Multiple consumers, that have discovered the Counter Web Service through a registry, can use its operations to change the counter's state (any security-related issues are orthogonal to this discussion and are not considered). In Figure 1, a consumer of the service submits a request for an increase operation.

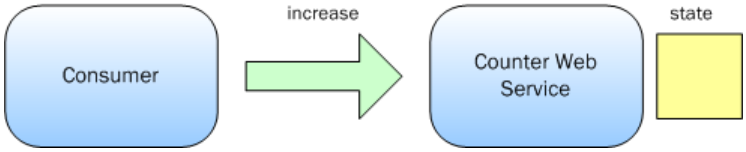

Figure 3-3: An increase operation to the Counter Web Service

If a Counter Web Service was to be extended so that multiple counters were supported, an ID would have to be introduced. The ID would identify the particular counter on which an operation was to be executed. Although the interface of the Counter Web Service would have to change in order to accept the ID, the same Counter Web Service is still going to be the logical receiver of the operations (Figure 3-4).

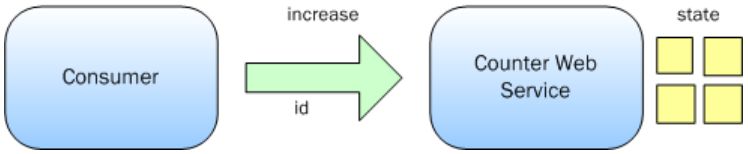

Figure 3-4: An increase operation to a Counter Web Service that supports multiple counters

Following, is the modified WSDL of the Counter Web Service that can support multiple counters.

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter"

targetNamespace="http://www.gridforum.org/namespaces/2003/05/ogsiprimer/
counter">
   <wsdl:types>
      <xs:schema/>
   </wsdl:types>
   <wsdl:message name="increaseMsg">
      <wsdl:part name="counterId" type="xs:positiveInteger"/>
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <wsdl:message name="getValueMsg">
      <wsdl:part name="counterId" type="xs:positiveInteger"/>
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <wsdl:portType name="counterPortType">
      <wsdl:operation name="increase">
         <wsdl:input message="increaseMsg"/>
      </wsdl:operation>
      <wsdl:operation name="getValue">
         <wsdl:output message="getValueMsg"/>
      </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

3.2.2 Counter Grid Service

In OGSI-terms, a Grid Service is a general term referring to a template, a contract, an interface described in GWSDL and the Grid Service Instances that must adhere to it. The term Grid Service does not characterise a component that can be part of a Grid application (i.e., it does not refer to a software component that can execute operations). Instead, it is Grid Service Instances that are the logical recipients of operations.

In the OGSI world, a Grid Service Instance (GSI) has to be explicitly created (by a factory or by the hosting environment) and then registered with a handle resolver. An example of the steps that may be required for a Grid Service Instance to be discovered, created, and consumed is described in section 4.2

Once a Grid Service Reference (GSR) to a Grid Service Instance is available to a consumer, the GSI can be used to offer equivalent functionality with its Web Service counterpart. However, the OGSI-specific characteristics of a GSI can be used to provide richer solutions.

The Counter Grid Service Instance of Figure 3-5 is equivalent to the Counter Web Service presented in Figure 1. Different service consumers can call operations on the same Counter GSI in order to change or access the state of the counter. This example differs from the one of Figure 1 in that the state is logically attached to the instance. The semantics of a Grid Service Instance, as defined by OGSI, specify that state has to be maintained between consumer-service interactions. This is done in an implementation specific way (i.e., the OGSI standard does not talk about how state is to be maintained).

Figure 3-5: An increase operation to a Counter Grid Service Instance

In contrast to the Counter Web Service, each Counter Grid Service Instance is associated with lifetime properties that can be used to manage the lifecycle of a counter. Also note that the OGSI specification does not deal with the semantics of concurrent access to the same Counter Grid Service Instance by multiple clients. Hence, the consistency semantics of state related information are application-specific.

The GWSDL interface of the Counter Grid Service looks very similar to its Web Service equivalent. Apart from the obvious namespace change, the only other difference is the introduction of portType extension using the "extends" attribute. This indicates to GWSDL processors that the counterPortType extends the functionality defined by the ogsi:GridService portType.

```
<wsdl:definitions
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/OGSI"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter"
                  targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter">
   <wsdl:types>
      <xs:schema/>
   </wsdl:types>
   <wsdl:message name="increaseMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <wsdl:message name="getValueMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
```

```
   <gwsdl:portType name="counterPortType" extends="ogsi:GridService">
      <wsdl:operation name="increase">
         <wsdl:input message="increaseMsg"/>
      </wsdl:operation>
      <wsdl:operation name="getValue">
         <wsdl:output message="getValueMsg"/>
      </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

Once a Grid Service Instance that adheres to the above GWSDL is created, it behaves in exactly the same way as the Counter Web Service but it also offers the additional functionality defined by OGSI. The GWSDL of a Grid Service Instance will, of course, contain the necessary bindings and service elements, as required by WSDL.

The implicit association between a Grid Service Instance and state, allows us to replicate the functionality of the multiple Counter Web Service without any change to the interface and implementation. All that is required is a new Grid Service Instance. One could argue that a new Web Service identical to the one of Figure 1 could be deployed if multiple counters were required while keeping the same interface. However, it is the way in which OGSI facilitates the creation of new transient instances and the management of their lifetime, identity and state that are of great value when compared to traditional Web Services.

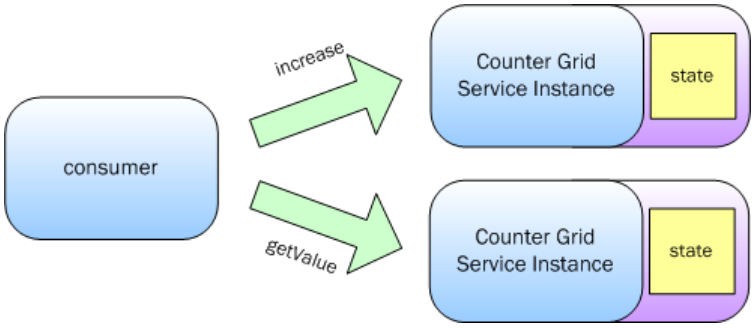

Figure 3-6: Operations to two different Counter GSIs representing a counter each

In addition to state management, Grid Service Instances can provide access to information through attribute-like constructs, called Service Data Elements (SDEs). An SDE is declared in the Grid Service's interface and accessed through operations defined by OGSI's GridService portType.

The Counter Grid Service may expose the value of the counter through an SDE. The getValue operation will no longer be necessary. The GWSDL of the Counter Grid Service will now be:

```
<wsdl:definitions
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/OGSI"
                  xmlns:sd="http://www.gridforum.org/namespaces
/2003/03/serviceData"
```

```
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter"
                  targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter">
   <wsdl:types>
      <xs:schema/>
   </wsdl:types>
   <wsdl:message name="increaseMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
      <wsdl:message name="getValueMsg">
      <wsdl:part name="value" type="xs:positiveInteger"/>
   </wsdl:message>
   <gwsdl:portType name="counterPortType" extends="ogsi:GridService">
      <wsdl:operation name="increase">
         <wsdl:input message="increaseMsg"/>
      </wsdl:operation>
      <sd:serviceData name="counterValue"
                      type="xs:positiveInteger"
                      minOccurs="1"
                      maxOccurs="1"
                      mutability="mutable">
         <sd:documentation>
            The value of the counter.
         </sd:documentation>
      </sd:serviceData>
   </wsdl:portType>
</wsdl:definitions>
```

It is now possible to access the counterValue SDE through operations specified by the Grid Service portType (Figure 3-7).

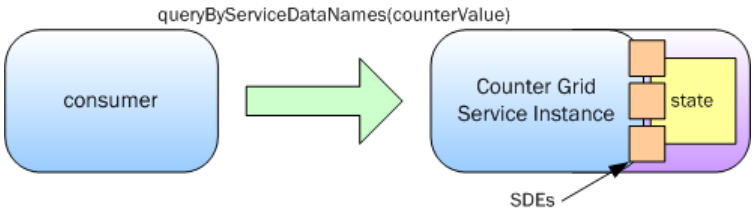

Figure 3-7: Consumer requesting the value of the counter through the counterValue SDE

Note that OGSI says nothing about how the value of an SDE is maintained, if it is maintained at all.

3.3 Terminology

Web Service: A software component identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by

Internet protocols. (Web Services Glossary, WS-Arch W3C Working Group, Draft 14 May 2003)

Web Service Consumer: An software component that sends messages to a Web Service.

Stateful Web Service: A Web Service that maintains some state between different operation invocations issued by the same or different Web Service Consumers.

Grid Service(s): A general term used to refer to all aspects of OGSI. The term "Grid Service" is sometimes used to refer to a Grid Service Description document and/or a Grid Service Instance for a particular service.

Grid Service Description: A WSDL(-like) document that defines the interface of Grid Service Instances. The defined interface must extend the OGSI GridService portType.

Grid Service Instance: A stateful Web service whose interface adheres to that defined by a Grid Service Description and whose lifetime management properties are well defined.

Service Data Element: An attribute-like construct exposing state information through operations defined by the GridService portType.

Grid Service Handle: A URI that permanently identifies a Grid Service Instance.

Grid Service Reference: A temporal, binding-specific endpoint that provides access to a Grid Service Instance.

4 The Concepts of Open Grid Services

This chapter introduces the ideas that are essential to Grid Services. These are:

- The use of 'extension' to create complex portTypes from simpler, ones, how this is enabled by Grid service WSDL and the behaviours every grid service must have as a result of extending the basic OGSI gridService portType.
- The roles of portTypes other defined by OGSI: the Factory, HandleResolver and registration.
- The Identity of a Grid Service Instance
- Faults
- Extensibility of Operations
- Service Data
- Notification

The next chapters (chapters 5 and 6) describes how Grid Services can be implemented and what large scale features are needed to create and/or use services based in a grid. More detailed explanations of the standard portTypes can be found in subsequent chapters starting with the core GridService portType in chapter 7.

4.1 The relationship of Grid service WSDL to Web service WSDL

Extra features of OGSI are introduced to basic Web Services by redefinition of the portType element of WSDL 1.1 which describes a service interface. The differences are marked in Figure 4-1 and can be summarised as follows:

- While a WSDL portType contains only operations, an OGSI portType can contain ServiceData descriptions.

- In OGSI, new portTypes can be constructed by reference to existing ones whose definitions it can extend. This enables standard behaviour to be defined authoritatively, and referenced by service definitions that need to incorporate it.

These additions to the WSDL document schema are prefixed with *gwsdl* to identify their definition by OGSI.

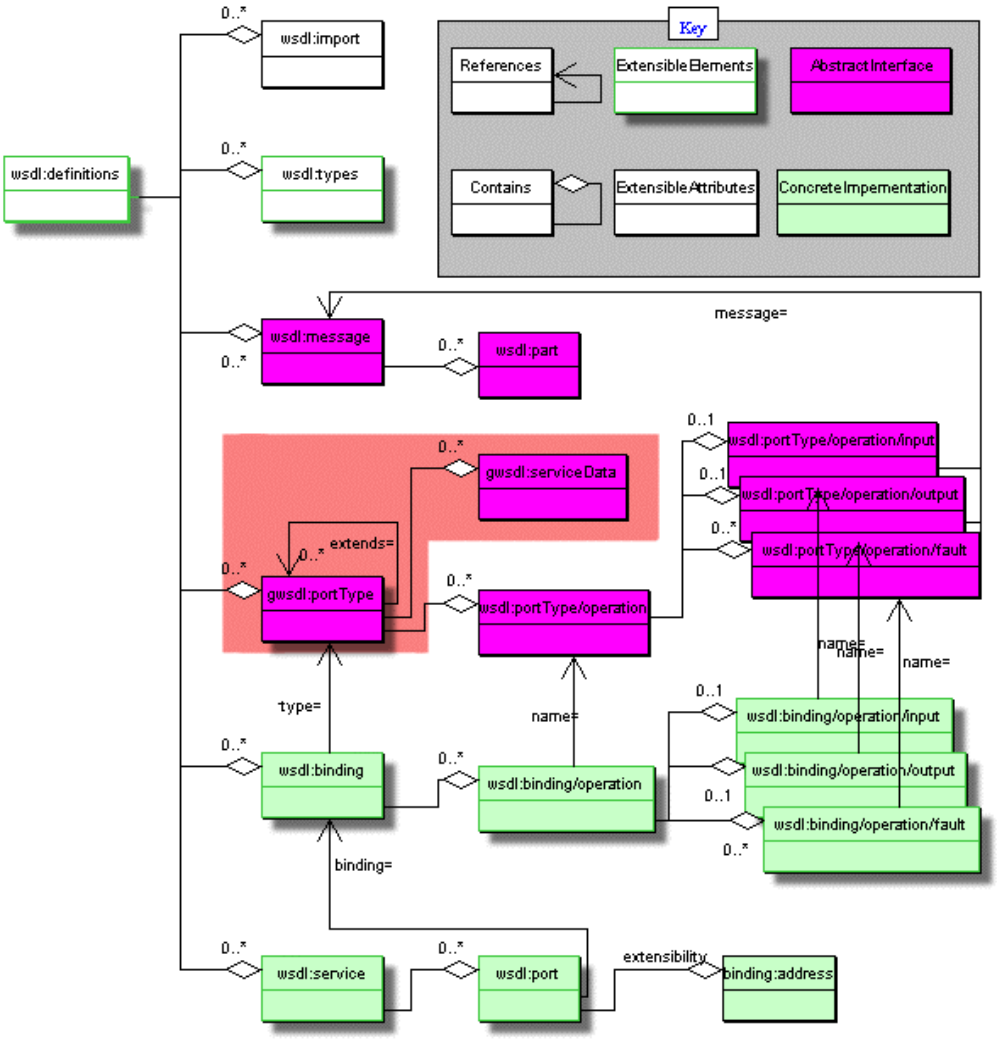

Figure 4-1: Markup language for Grid Services

OGSI is based on Web services, and in particular uses WSDL as the mechanism to describe the public interfaces of Grid services. However, WSDL 1.1 is deficient in two critical areas: lack of interface (portType) extension and the inability to describe additional information elements on a portType which is needed to enable the description of Service Data. These deficiencies have been addressed by the current "work in progress" draft of WSDL 1.2 in the W3C Web Services Description Working Group [WSDL 1.2 DRAFT]. Because WSDL 1.2 is currently "work in progress", OGSI cannot directly incorporate the entire WSDL 1.2 body of work.

Instead, OGSI defines an extension to WSDL 1.1, isolated to the wsdl:portType element, that provides the minimal required extensions to WSDL 1.1. These extensions to WSDL 1.1 match equivalent functionality agreed to by the W3C Web Services Description

Working Group. **The Global Grid Forum commits to updating OGSI when WSDL 1.2 [WSDL 1.2] is published as a draft specification by the W3C**.

4.1.1 Using portType extension

Grid Service Description can use portType extension to define standard operations by the use of the 'extends' attribute. For example, the Counter grid service (Section 3.2.2) begins:

```
<gwsdl:portType name="counterPortType" extends="ogsi:GridService">
```

This declaration causes the inclusion of the definitions for ServiceData from the **ogsi:GridService** portType (described fully in chapter 7). All Grid Service Descriptions include these definitions. In summary, they are:

- The identity (known as the Grid Service Handle) of any Grid Service Instance created to implement the description.
- The names of all the portTypes which the Instance implements.
- The identity of the factory which created the Instance.
- The termination time of the Instance.

In addition, the GridService portType provides operations which allow a client to query and modify this and other ServiceData.

Finally, the GridService portType provides an operation to destroy the Instance.

4.2 Service Factories and Instances

Figure 4-2 shows an interaction between components defined by OGSI which is a typical pattern for the use of Grid Services. The client uses a well-known service called the Registry to discover a factory. Many factories may exist in the registry, each capable of creating different types of Grid Service Instance. The factory establishes a globally unique identity for the instance and creates it. The handle and references are returned to the client (for use in invoking the service instance) and also published in the handleResolver.

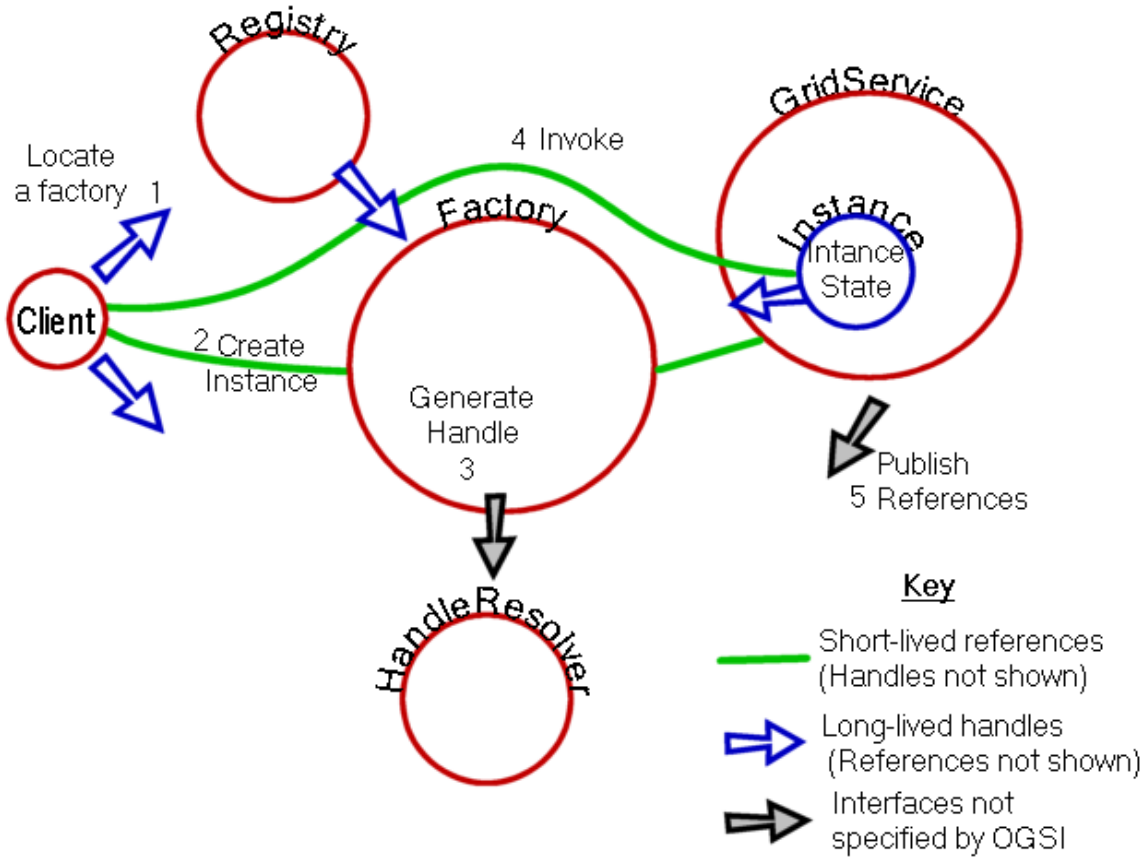

Figure 4-2: Main Components Of OGSI Services

4.3 Identity, Handles, References and Locators

4.3.1 Relative longevity/fragility of Handle and Reference

A client gains access to a Grid Service Instance through Grid Service Handles and Grid Service References. A Grid Service Handle (GSH) can be thought of as a permanent network pointer to a particular Grid Service Instance, but the GSH does not provide sufficient information to allow a client to access the service. The client needs to "resolve" a GSH into a Grid Service Reference (GSR). which contains all the necessary information to access the service. The GSR is not a "permanent" network pointer to the Grid Service Instance because a GSR may become invalid for various reasons; for example, the Grid Service Instance may be moved to a different server.

4.3.2 Passing Handles and references: the Locator

Handles and References may be passed as input and/or output parameters, and it is often convenient to pass a collection of References and Handles so that the eventual user (a client application or its supporting middleware) of the target instance can choose the most appropriate way of calling it. For example, a client which is local to the Instance may use a GSR representing a binding describing unencrypted transmission. Conversely, if the

client is remote, and the Locator has been stored for some time before use and its references are no longer valid, the client may choose one of the Handles, for example a handle described by a secure scheme, and resolve it to a GSR which describes an encrypted protocol.

A Locator may also contain the name (as a fully qualified QName) of the interface for the target service. This can be used by the recipient of the Locator discover the interface description, check that the handles and refrences refer to implementations which are suitable for the client, and allow client infrastructure to enable access to the service. In some cases, References themselves may contain interface references or descriptions, but this is not always the case.

Many handle schemes are possible; the OGSI specification does not restrict or mandate any particular one and allows multiple schemes to be used in parallel. In this Primer, for the purposes of examples, we use a simple one based on the http protocol. In this scheme, a handle is a globally unique URI in the http scheme (that is, beginning "http://") described further in. Handles may also be based on https to provide encryption of responses from the resolver. Other schemes may be specified and developed in with characteristics such as improved caching or trustworthiness. See [5] for an example.

4.3.3 The Identity of Grid Service Instances

Each Grid Service Instance is distinguished from all others by means of its Handles and References. Use of the same Handle to call (via resolution) operations on multiple occcasions, even by different clients, must mean the operations act on the same Instance.

However, a grid service can have multiple Handles and/or References and a client may receive different versions from several sources as Locators are passed around. Comparing versions of these recieved from different sources is not useful as a way of identifying the target Grid service Instance as being the same one. If a Service requires to Clients to be able to establish equivalence of several Locators, Handles or References, it must provide an operation to do this and the details of the comparison are dependent on the service.

For example, a counter service may provide access to each counter Instance by multiple clients. One way of implementing the service is to establish multiple copies of the same Instance with multiple (different) references to those copies as a way of providing workload distribution. Nevertheless, each reference is equivalent to the others provided the copies are correctly synchronised during updates. A service may also place additional information in its Handles to distinguish copies of the service from each other. The Counter Instance may provide different Qualities of Service (such as fast or slow response times) to different clients based on information in the Handle, yet still preserve the semantics of the Counter service across all clients of the Instance.

The semantics of the service description may, like the counter, require that the service move through a series of well-defined states in response to a particular sequence of messages, thus requiring state coherence regardless of how Handles are resolved to

References. However, other service descriptions may be defined that allow for looser consistency between the various members of the distributed service implementation. The copies of the Service Instance, identified by different Handles and References may respond differently in some respects, yet still be the same Instance.

4.4 Operation Faults

A common procedure is used for handling operation faults. This simplifies problem determination by having a common base set of information that all fault messages contain. It also allows for "chaining" of fault information up through a service invocation stack, so that a recipient of a fault can drill down through the causes to understand more detail about the reason for the fault. OGSI defines a base XSD type (ogsi:FaultType) for all fault messages that Grid services must return. All faults from a Grid service should either use the ogsi:FaultType directly, or extensions of it. All Grid service operations must return the ogsi:fault (which is of type ogsi:FaultType) in addition to any operation-specific faults. For example

```
   <wsdl:definitions ...>
      <types>
         <xsd:schema ...>
            <xsd:complexType name="MyFaultType">
               <xsd:complexContent>
                  <xsd:extension base="ogsi:FaultType"/>
               </xsd:complexContent>
            </xsd:complexType>
         <xsd:element name="myFault" type="tns:MyFaultType"/>
      </xsd:schema>
   </types>

   <message name="myFaultMessage">
      <part name="fault" element="tns:MyFaultType"/>
   </message>

   <gwsdl:portType ...>
      <wsdl:operation ...>
         <input ...>
         <output ...>
         <fault name="myFault" message="tns:myFaultMessage"/>
         <fault name="fault" message="ogsi:faultMessage"/>
      </wsdl:operation>
   </gwsdl:portType>
</wsdl:definitions>
```

4.5 Extensible of Operations

[from the spec section 7.8]

Several OGSI operations accept an input argument that is an untyped extensibility element, which allows for common patterns of behavior to be expressed in an extensible manner. In order to allow a client to discover the valid extensions that are supported by such an

operation, we define a common approach for expressing extensible operation capabilities via static service data values.

For example, the NotificationSource::subscribe operation allows a client to ask a service for notification messages whenever portions of that service's serviceDataValues changes. The specific portions of service data upon which to send notifications are defined by a subscription expression. This argument is not fully typed, but is instead an extensible argument. One simple subscription expression is defined by the NotificationSource portType that can be passed in this argument to any service that implements NotificationSource. However, services that implement a portType that inherits from NotificationSource can extend the capabilities of subscription by defining new query expressions that are more powerful, and/or more customized to a specific problem domain. This section defines the means by which a client can determine what.subscription expressions are supported by services that implement extensions to the NotificationSource portType.

We define a single XSD type that is the base for all SDEs that describe extensible operations:

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"
<xsd:complexType name="OperationExtensibilityType">
<xsd:attribute name="inputElement" type="QName" use="optional"/>
</xsd:complexType>
```

For each extensible operation in a portType, that portType SHOULD have a serviceData declaration of type OperationExtensibilityType, and mutability="static". Static values of this SDE define the valid extensions of the operation.

For example, suppose we have portType named myPT with an operation named myOperation, such that one of the myOperation's input parameters is extensible. Further suppose there are two standard input elements called myop:myOption1 and myop:myOption2 that can be passed into myOperation's extensible input parameter. The portType definition for myOperation would be:

```
<gwsdl:portType name="myPT">
…
<sd:serviceData name="myOperationExtensiblity"
type="ogsi:OperationExtensiblityType"
minOccurs=0 maxOccurs="unbounded"
mutability="static"
modifiability="false"
nillable="false" />
…
<sd:staticServiceDataValues>
<myOperationExtensibility inputElement="m1:myOption1"/>
<myOperationExtensibility inputElement="m1:myOption2"/>
</sd:staticServiceDataValues>
…
</gwsdl:portType>
```

The inputElement attribute of the SDE MUST be a QName that uniquely implies the types and behavior of a particular extension to the operation. The inputElement, if present, SHOULD be the QName of an XSD element decla ration that is a valid element that can be

passed to an operation as an extensible input argument. All other properties of the extensible operation are implied by the inputElement, unless they are explicitly defined in an extension of OperationExtensibilityType. For example, the inputElement MAY imply the type of output parameters from the operation, and MAY imply the semantics of how the operation when it receives this inputElement.

If inputElement is omitted from the SDE value, then it MUST be valid to omit the extensible input argument when invoking the operation.

Operation extensibility SDE values MAY be included in portTypes that extend a portType containing operation extensibility serviceData declarations. For example, a second portType named myPT2 that extends myPT could define additional valid input arguments to myOperation:

```
<gwsdl:portType name="myPT2" extends="m1:myPT>
…
<sd:staticServiceDataValues>
<myOperationExtensibility inputElement="m2:myOption3"/>
<myOperationExtensibility/>
</sd:staticServiceDataValues>
…
</gwsdl:portType>
```

In this example, a service that implements myPT2 would support four options for the extensible input argument to myOperation: m1:myOption1, m1:myOption2, m2:myOption3, and no element at all. Each of these options may further imply output argument types, semantics of the operation related to the inputElement, etc.

In some situations it is useful to extend ogsi:OperationExtensibilityType to include additional attributes or elements. In this case, a new type should be defined that is an xsd:extension of ogsi:OperationExtensibilityType, along with a serviceData element defined with this extended type. See the Factory portType (§12) for an example of this.

4.5.1 Extensible ServiceData elements

4.6 Introducing ServiceData

[ServiceData Lifetimes and the concept of soft state]

[ServiceData as the external view of the Service, including introspection and its implications for registration, inquiry, monitoring, configuration and notification]

In order to support discovery, introspection, and monitoring of Grid service instances, we introduce the concept of service data, which refers to descriptive information about a Grid service instance, including both meta-data (information about the service instance) and state data (runtime properties of the service instance). We describe the components of the service data concept and their relationships.

Each Grid service instance has an associated set of service data elements (SDEs). Each SDE is represented in XML by a serviceData element (see Section 4.4.1). Each SDE can be used to model a separate property of the service.

ServiceData elements are roughly analogous to instance variables in a class definition in some object-oriented programming language. However, service data elements represent read-only state data.

(what follows is from section 6 of the spec)
The approach to *stateful* Web services introduced in OGSI identified the need for a common
mechanism to expose a service instance's state data to service requestors for query, update and change notification. The term used is "serviceData". Since this concept is applicable to any Web service including those used outside the context of Grid applications, we propose a common approach to exposing Web service state data called serviceData. We are endeavoring to introduce this concept to the broader Web services community.
In order to provide a complete description of the interface of a stateful Web service (i.e., a Grid service), it is necessary to provide a description of the elements of its state that are externally observable. By externally observable, we mean to say that the state of the service is exposed to clients making use of the declared service interface, where those clients are outside of what would be considered the internal implementation of the service itself. The need to declare service data as part of the service's external interface is roughly equivalent to the idea of declaring attributes as part of an object-oriented interface described in an object-oriented interface definition language (IDL). Service data can be exposed for read, update, or subscription purposes.

Since WSDL defines operations and messages for portTypes,, the declared state of a service MUST only be externally accessed through service operations defined as part of the service interface. To avoid the need to define serviceData specific operations for each ServiceData element, the Grid service portType (§9) provides base operations for manipulating ServiceData elements by name.

Consider an example. Interface foo introduces operations op1, op2, and op3. Also assume that the foo interface consists of publicly accessible data elements of de1, de2, and de3. We use WSDL to describe foo and its operations. The OGSI serviceData construct extends WSDL so that the designer can further define the interface to foo by declaring the public accessibility of certain parts of its state de1, de2 and de3. This declaration then facilitates the execution of operations against the service data of a stateful service instance implementing the foo interface.

[The Abstract Structure of Service Data]

4.7 Introducing Notification

The notification framework allows for asynchronous, one-way delivery of interesting messages from a source to a subscribed sink. Any service that wishes to support subscription of notification messages, must support the *NotificationSource* interface.

OGSI allows for notification on the service data elements of a grid service instance. As part of its service Data, the *NotificationSource* maintains a set of service data elements to which a requestor may subscribe for notification of changes. If notification on implementation-specific internal state is desired, then additional servide data elements may be defined for that purpose. For example in the Counter example from section 3.2 where state is maintained internally as an integer, a service data element (CounterStatus? in the example below) may be defined for the purpose of notification.

```
<wsdl:definitions
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/OGSI"

xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:counter-
notifcation="http://www.gridforum.org/namespaces/2003/05/ogsiprimer/coun
ter/notification"

targetNamespace="http://www.gridforum.org/namespaces/2003/05/ogsiprimer/
counter/notification">
   <wsdl:types>
      <schema
targetNamespace="http://www.globus.org/namespaces/2003/05/ogsiprimer/cou
nter/notification"
        attributeFormDefault="qualified"
         elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">
         <complexType name="CounterStatusType">
            <sequence>
               <element name="status" type="string"/>
            </sequence>
         </complexType>
      </schema>
   </wsdl:types>

   <wsdl:message name="increaseMsg">
      <wsdl:part name="value" type="xsd:positiveInteger"/>
   </wsdl:message>
   <wsdl:message name="getValueMsg">
      <wsdl:part name="value" type="xsd:positiveInteger"/>
   </wsdl:message>
   <gwsdl:portType name="NotificationCounterPortType"
extends="ogsi:NotificationSource">
      <wsdl:operation name="increase">
         <wsdl:input message="increaseMsg"/>
       </wsdl:operation>
      <sd:serviceData name="CounterStatus" type="counter-
notification:CounterStatusType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false">
        <documentation>Sample Counter Status  Type as
SDE</documentation>
```

```
      </sd:serviceData>
    </gwsdl:portType>
</wsdl:definitions>
```

In the implementation of the Grid service, the CounterStatus SDE should be updated whenever a change to the counter state occurs.

To start notification from a particular service one has to invoke the *subscribe* operation on the notification source interface, giving it the GSH of the notification sink. A subscription request also contains a subscription expression which is an XML element that describes what messages should be sent from the source to the sink, as well as when messages should be sent, based on changes to values within a service instance's serviceData values.

A subscription request causes the creation of a *subscription* Grid service instance which can be used by the client to manage the (soft-state) lifetime of the subscription, and to discover properties of the subscription (This is possible because the *NotificationSource* portType is a factory of *subscription* Grid service instances). A locator to this subscription instance is returned as an part of the output as well the currently planned termination time.

Following a successful subscription, a stream of notification messages then flow from the source to the sink, whenever a change to the subscribed SDE's occurs, until the subscription is either explicitly destroyed or is timed out.

Any client that is interested in receiving notification, only has to support the *NotificationSink* interface. That is a Web service is not required to also implement the *GridService* portType in order to act as a notification sink.

The OGSI Notification framework defines how to subscribe to SDEs by name only, but allows for more advanced subscription expressions through the subscribe operation extensibility declarations. This will be investigated in chapter 11.

5 Implementing Web and Grid Services

In this section, we examine the relationship between OGSI and the existing and developing Web services framework (description languages, tools and implementation platforms) on which it builds and depends. We examine both the client-side programming patterns for Grid services and several conceptual hosting environments for Grid services. The patterns described in this section are enabled but not *required* by OGSI. We discuss these patterns in this section to help put into context concepts and details described in the other parts of the Primer and in the Specification. In particular, the discussion of implementing robust services provides more information on the relationship between handles and references, and the discussion of replicated service instances answers some questions about the issue of identity.

First, however, we discuss the client side.

5.1 Client-Side Programming Patterns

An important issue, that requires some explanation, particularly for those not familiar with Web services, is how OGSI interfaces are likely to be invoked from client applications.

OGSI exploits an important component of the Web services framework: the use of WSDL to describe abstract interfaces and, separately, multiple protocol bindings, encoding styles, messaging styles (RPC vs. document-oriented), security tokens and so on, for a given Web service. The *Web Services Invocation Framework* [WSIF] and *Java API for XML RPC* [JAXRPC] are examples of infrastructure software that provide the capability to separate interface from bindings and provide multiple bindings. There are many other examples. In each case, the basic structure is similar, and it provides a client-side architecture which can also be used to access OGSI services.

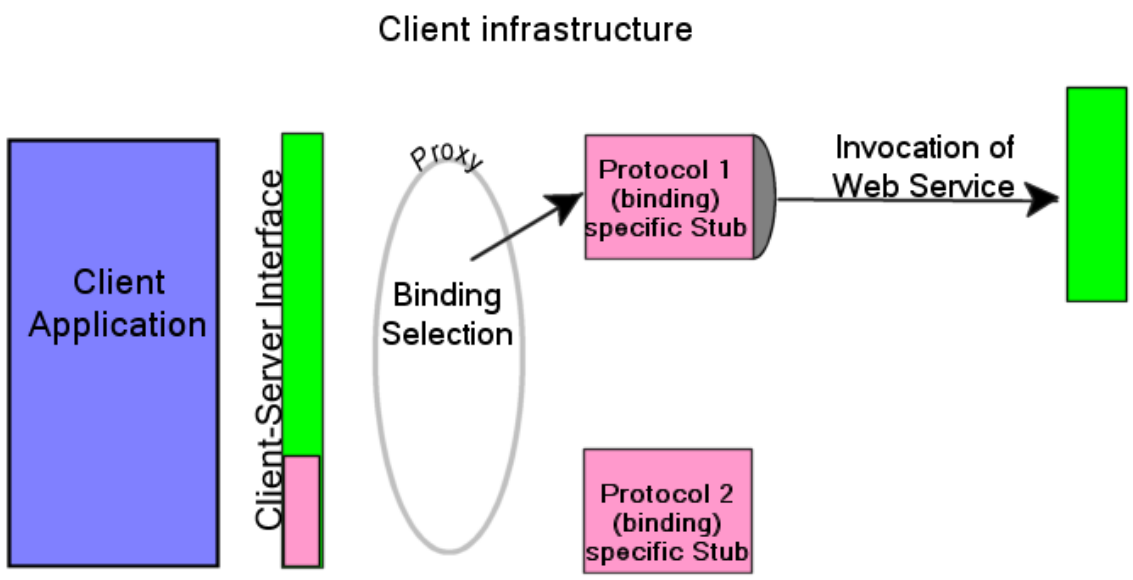

Figure 5-1: A client-side runtime architecture

In figure 5, the client application accesses a client-side representation of the Web service, sometime called a *proxy*. The proxy calls, or uses components (shown as protocol-specific *stubs*) which marshal parameters for the invocation of the Web service over a chosen binding, and adding other necessary information such as protocol headers or security tokens obtained from the client's runtime environment.

There is a clear separation between the client application and the stub by the client-side interface which consists of the methods, procedure calls, parameters, exceptions and other interface features needed to invoke the service and receive responses and handle faults generated by the service. The client interface also defines exceptions which describe, in a generic way, errors arising from the binding mechanisms. These are dependent on the particular client infrastructure.

The client's interface to the service can be generated by taking the WSDL description of the Web service interface and transforming it into interface definitions in a programming language specific way (e.g. Java interfaces or C#) suitable for the particular client-side architecture. The parameter marshalling and message routing stubs are generated from the various binding options provided by the WSDL. Generation of the interface and stubs can take place at various stages of development, deployment and execution of the client application: dynamic Web service invocation leaves the generation of the binding and service address until execution time.

This approach allows certain efficiencies, for example, detecting that the client and the Web service exist on the same network host, and therefore avoiding the overhead of preparing for and executing the invocation using network protocols.

It is possible, but not recommended, for developers to build customized code that directly couples client applications to fixed bindings of a particular service. Although certain circumstances demand potential efficiencies gained this style of customization, this approach introduces significant inflexibility into a system and therefore should only be used under extraordinary circumstances.

We expect the stub and client side infrastructure model that we describe to be a common approach to enabling client access to Grid service instances where each remote instance is represented by a proxy. This includes both application specific services as well as common infrastructure services that are defined by the OSGI Specification and described in this Primer. So, for most developers using Grid services, the infrastructure and application level services appear in the form of a class library or programming language interface that is natural to the caller.

The WSDL and the GWSDL extensions required by OGSI can be support by heterogeneous tools and enabling infrastructure software. The techniques for doing this are discussed in section 5.3.

Further discussion of the Client-side, including its use of registries, grid Service handles and resolution of bindings and References can be found in section 6.3.

5.2 Server-Side Programming Patterns

The OGSI Specification does not dictate any particular service-side implementation architecture. A variety of approaches are possible, ranging from implementing the Grid service instance directly as an operating system process to a sophisticated server-side component model such as J2EE. In the former case, most or even all, support for standard Grid service behaviors (invocation, lifetime management, registration, etc.) is encapsulated within the user process, for example via linking with a standard library; in the latter case, many of these behaviors will be supported by the hosting environment.

To illustrate a range of possibilities for implementation, and the features of the Specification which enable this range we describe different hosting patterns of increasing complexity in the following sections.

5.2.1 Monolithic Service Implementation

In Figure 5-2, we depict a scenario where the entire behavior of the Grid service, including the demarshalling/decoding of the network message, has been encapsulated within a single executable. The *protocol termination* represents a communications component such as an http server with the Grid service implemented directly using the servlet interface. Although this approach may have some efficiency advantages, it provides little opportunity for reuse of functionality between Grid service implementations.

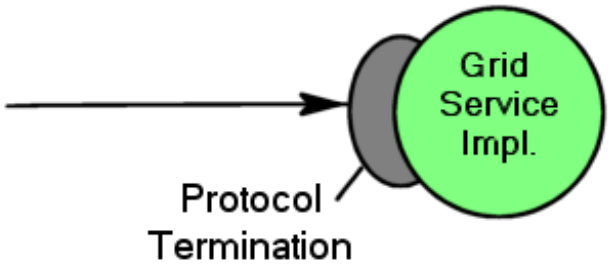

Figure 5-2: Simple monolithic Grid service

5.2.2 Implementation within a Container

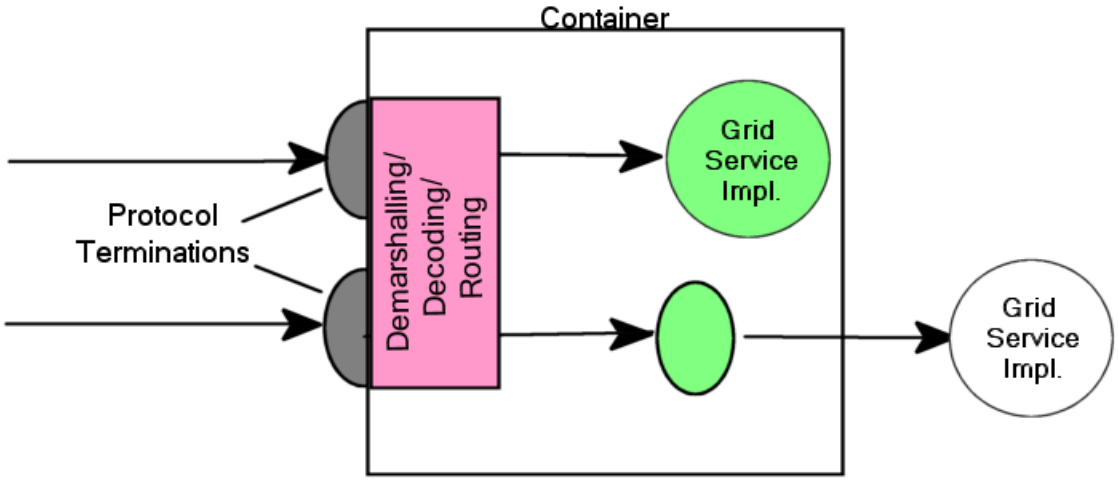

Figure 5-3: Container approach to the implementation of argument demarshalling functions.

In Figure 5-3, the invocation message is received at a network protocol termination point (e.g., an http server, as is the case for many Grid services.) This converts the data in the invocation message into a format consumable by the hosting environment. We illustrate two Grid service instances (the circles) implemented as container-managed components (for example EJBs within a J2EE container). Here, the message is dispatched to these components, with the container frequently providing facilities for demarshalling and decoding the incoming message from a format (such as an XML/SOAP message) into an invocation of the component in native programming language. Demarshalling from protocol terminations to the native language is accomplished by generated components called *skeletons* which correspond to the stubs on the client-side. In some circumstances (the upper circle), the entire behavior of a Grid service is completely encapsulated within the component. In other cases (the lower circle), a component will collaborate with other server-side executables, to delegate certain operations to standard components or perhaps through an adapter layer designed to translate language invocation syntax, to complete the implementation of the Grid service behavior, or

5.2.3 Container-managed State

A container implementation may provide a range of functionality beyond simple argument demarshalling. For example, the container implementation may provide lifetime management functions, automatic support for authorization and authentication, request logging, intercepting lifetime management functions and terminating service instances when a service lifetime expires or an explicit destruction request is received. Thus, we avoid the need to re-implement these common behaviors in different Grid service implementations.

Figure 5-4 shows a container which consists of multiple execution environments which can be selected to host a Grid service instance and is a possible implementation of the J2EE container described in the EJB specification [9]. In this implementation the state of the instance can be saved by the container at operation boundaries and restored in case a service instance fails due to software or hardware errors.

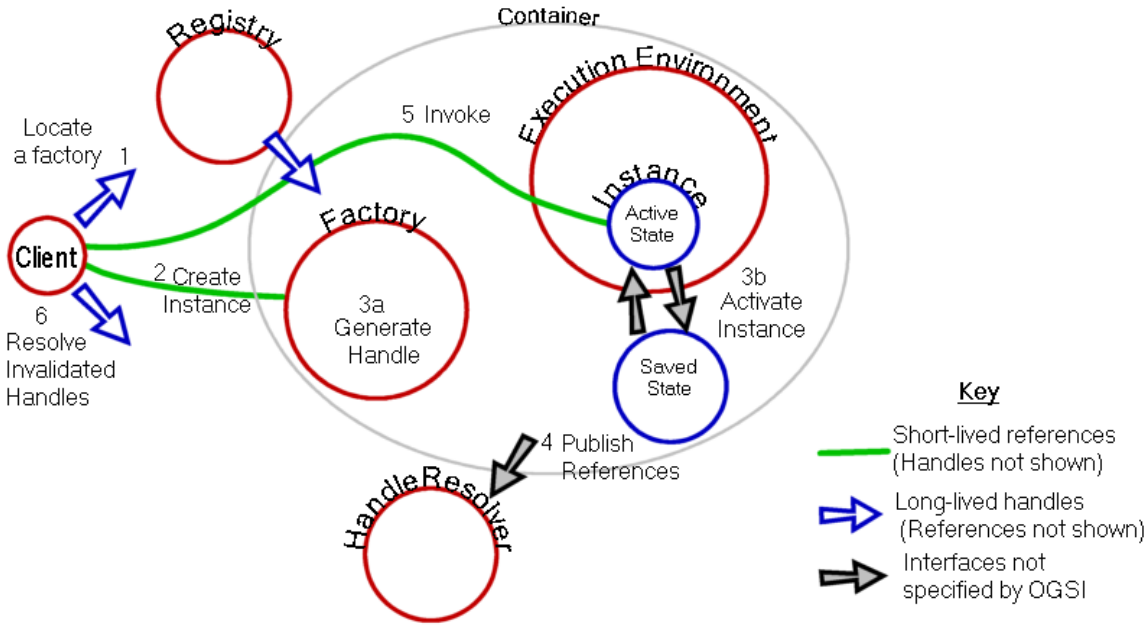

Figure 5-4: Container with state management

Figure 5-4 includes Grid HandleResolver and Registry services so that the order of events required to create and access the service can be seen. The client has a permanent reference to the HandleResolver and uses it to resolve service Handles that become invalid. The client also has a handle to the registry service which it can use (1) to locate a suitable factory and, via the factory (2), create a Grid service Instance (3a). The container is responsible for allocating the execution environment for the instance, establishing the service Reference and supplying this to the HandleResolver (4), and initialising the service state and managing the store/load operations. The factory returns to the client a Locator containing a Handle and a Reference to the new service Instance which can be used to invoke service operations (5). If the execution environment fails, the container may allocate a new one with a new service Reference, load the state and supply the new reference to the HandleResolver. The client uses the Handle to obtain the updated Reference (6).

5.2.4 Replicated Copies of a Service Instance

[This illustrates the use of multiple references simultaneously by different clients to access the same service Instance. See OpFAQHandleResolution for input material.]

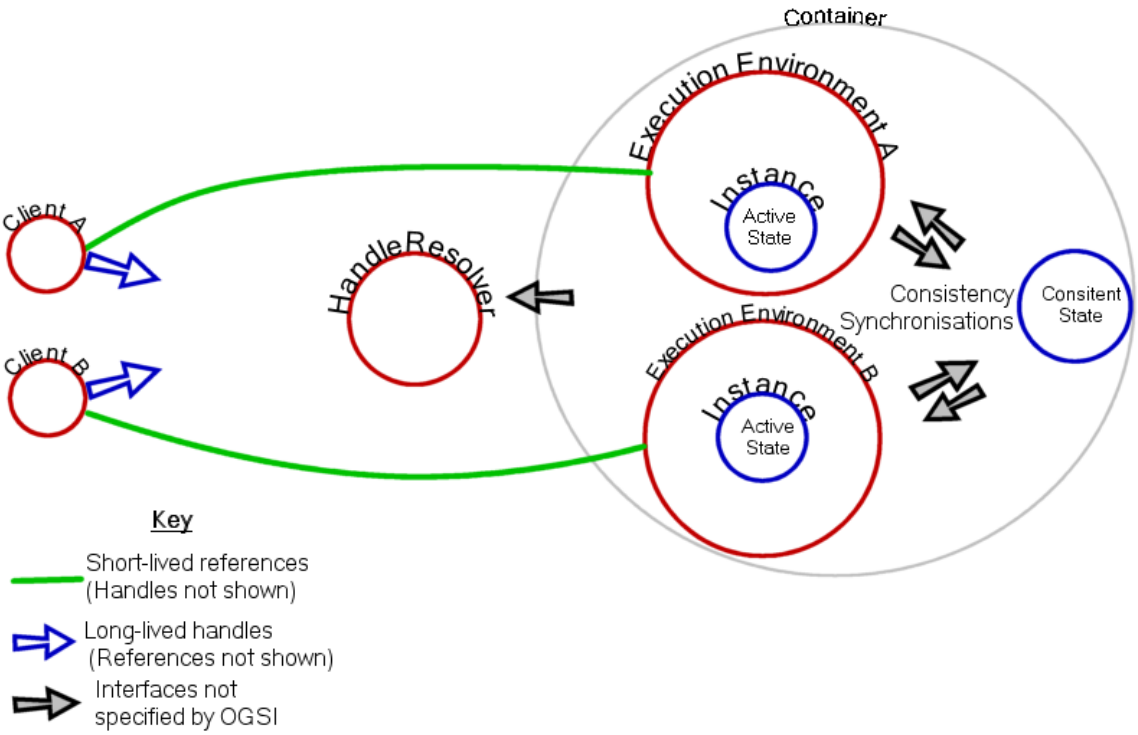

Figure 5-5: Use of multiple references for the same Grid service Instance

5.3 The Relationship of Grid service WSDL to Web service WSDL

[An explanation of the interpretation, in tooling, of gwsdl constructs in WSDL 1.1 is needed here, namely:

- *portType extension*
- *ServiceData description?*

What tool and extensions are needed?

The exchanges on WGSI-WG around 11/12th May provide input]

6 Using Grid Services

This chapter introduces the concepts that are needed to use create and use the service instances described by the previous chapter. The issues are

- Locating services and communicating with them
- Ways in which collective information is used.

6.1 Using Registries

6.1.1 The purpose of registries in OGSA

A registry is the very general concept. In computing, we may define it as a catalogue of objects, described and indexed by the objects' metadata. The objects may be either inside the computing system (i.e. the objects consist entirely in data); or they may be real-world items, with only metadata inside the computing system; or they may be computing-system proxies for real-world objects. In all cases, the essence of the registry pattern is that a registry stores some part of an object's metadata without storing either the object itself or (for a real-world object) its proxy.

Given a registry, clients may inspect the metadata and use them to reason about the catalogued objects. Typically, clients then use the metadata to locate and access some of the catalogued objects. Often, the only use of the registry is to locate the objects.

OGSI is concerned with registries of grid services. Grid services can be used to build other kinds of registries, but OGSI has specific port-types that can be used to build registries where the entries describe grid services.

The essential requirement of a service registry is that the metadata include a GSH for each catalogued service. The registry itself does not resolve a GSH to a GSH, although the service running the registry may also include a *HandleResolver* port for this purpose. From OGSA's point of view, "locating" a service instance through a registry means finding its persistent handle, not finding its reference, the latter function being the role of a handle resolver.

6.1.2 OGSI support for service registries

In OGSA, service registries have two uses:

- finding service factories from which to make new instances;
- finding existing instances.

The same OGSI port-types cover both uses:

- *serviceGroup*

- *serviceGroupEntry*
- *serviceGroupRegistration*

From the OGSI specification:

> The *ServiceGroup* port-type provides an interface for representing a service group comprising zero or more member services. [...] The *ServiceGroup* port-type extends the *GridService* port-type.
>
> [The serviceGroupEntry] port-type defines the interface through which individual entries in a service group should be managed. Each *ServiceGroupEntry* service refers to a grid-service instance that is a member of the *ServiceGroup*
>
> [The *ServiceGroupRegistration*] port-type provides a management interface (add and remove operations) for a service group. The *ServiceGroupRegistration* port-type extends the *ServiceGroup* port-type.

Thus, in OGSI, a service group may be used to create a service registry.

The service implementing the registry provides either a *ServiceGroup* or a *ServiceGroupRegistration* port. If the set of registered services is updated only through local, non-grid interfaces, then the registry service has a *ServiceGroup* port; call this a "locally-managed" registry. If the set of registered services can be altered by clients working over the grid, then the registry service has instead a *ServiceGroupRegistration* port; call this a "grid-managed" registry. Since *ServiceGroupRegistration* extends *ServiceGroup*, the registry service needs only one of these two port types.

The registry facility also has one *ServiceGroupEntry* port for each registered service. Since any given grid-service can only have one copy of a given port-type, this means that the registry has one subsidiary service, providing *ServiceGroupEntry*, as a "registration proxy" for each registered service. In fact, the intention is to have the *appearance* of one proxy service-instance per registered service in order to use the lifetime-management facilities of the *GridService* port. In practice, a registry may implement these proxy services in some light-weight form in order to save resources. I.e., a registry for a trivial number of services may use conventional service-instances for its proxy services, using ordinary service containers, while a registry intended for thousands of registrants may be built as a specialized service-container with a more-scalable implementation of the proxies.

A service instance is registered by

1. creating a new proxy service-instance with the *ServiceGroupEntry* port-type;
2. adding an element of type *ogsi:EntryType* (see below) to the service data of the *ServiceGroup* or *ServiceGroupRegistration* port.

Deregistration is the reverse process. A registration proxy can be set to terminate at a certain time, in the normal manner of OGSI services, thus limiting the lifetime of a registration.

In a grid-managed registry, clients can call the *add* operation of the *ServiceGroupRegistration* port to register a service and the *remove* operation of that port to deregister services. Each call to *add* registers one service. Each call to *remove* deregisters a set of related services. In a locally-managed registry, the registration and deregistration mechanisms are outwith OGSI.

The *ogsi:Entry* contains:

- an *ogsi:Locator* element pointing to the registration proxy for the registered service;
- an *ogsi:Locator* element pointing to the registered service itself;
- an *ogsi:EntryContent* element that contains zero or more unconstrained metadata describing the registered service.

Thus a client of the registry reads the metadata of registered services by reading the *ogsi:EntryContentType* elements from the *ServiceGroup* port.

[Some words here or hereabouts concerning how the registry limits what comes in. _MembershipRule and all that.]

6.1.3 Example 1: a simple registry of service factories

This registry records the service factories currently running in one service container; it's a way of keeping track of current activity on one server. Any kind of factory can be registered. The registered services need have nothing in common except what is required by the OGSI standard. The main metadata of interest are the sets of port types provided by each service instance created by the factories.

This is a locally-managed registry. The registry contents are set by the service operator as part of the registry-service's configuration and are not mutable via the grid. Hence, this registry is based on a *ServiceGroup* port, not a ServiceGroupRegistration_ port.

The metadata in the registry come from the standard service-data prescribed by OGSI for the *Factory* port-type. The *ogsi:CreateServiceExtensibilityType* elements in a *factory* port list as QNames all the port-types of the service instances that the factory can create.

For this example, suppose that there are just three factories, all for variants of a counter service.

- The basic counter has a *Counter* port and a *GridService* port.
- The "private" counter, which enforces access control for a resticted set of users, has a *SecureCounter* port and a *GridService* port.

- The "shared" counter, which is intended for concurrent use by more than one client, has a *Counter* port, a *NotificationSource* port and a *GridService* port.

Suppose further that this set is very stable, such that the author of the registry chooses to hard-code the membership and metadata of the registry. Therefore, the membership and member-descriptions are all carried in the GWSDL for the registry as "static" SDEs:

```
<gwsdl:portType name="ServiceGroup">

   <!-- Definition of the SD as per OGSI standard. -->
   <sd:serviceData name="entry"
                   type="ogsi:EntryType"
                   minOccurs="0"
                   maxOccurs="unbounded"
                   mutability="mutable"
                   modifiable="false"
                   nillable="false"/>

   <!-- Specific values for this registry. -->
   <sd:staticServiceDataValues>

      <!-- Description of basic-counter factory. -->
      <ogsi:EntryType>
         <serviceGroupEntryLocator nil="true"/>
         <memberServiceLocator>

<ogsi:handle>http://exemplar.org/ogsa/services/CounterFactory</ogsi:hand
le>
         </memberServiceLocator>
         <content>
            <ogsi:CreateServiceExtensibilityType>
               <createsInterface>Counter</createsInterface>
               <createsInterface>GridService</createsInterface>
            </ogsi:CreateServiceExtensibilityType>
         </content>
      </ogsi:EntryType>

      <!-- Description of private-counter factory. -->
      <ogsi:EntryType>
         <serviceGroupEntryLocator nil="true"/>
         <ogsi:memberServiceLocator>

<ogsi:handle>http://exemplar.org/ogsa/services/PrivateCounterFactory</og
si:handle>
         </memberServiceLocator>
         <content>
            <ogsi:CreateServiceExtensibilityType>
               <createsInterface>SecureCounter</createsInterface>
               <createsInterface>GridService</createsInterface>
            </ogsi:CreateServiceExtensibilityType>
         </content>
      </ogsi:EntryType>

      <!-- Description of shared-counter factory. -->
      <ogsi:EntryType>
```

```
         <serviceGroupEntryLocator nil="true"/>
         <memberServiceLocator>

<ogsi:handle>http://exemplar.org/ogsa/services/SharedCounterFactory</ogs
i:handle>
         </memberServiceLocator>
         <content>
            <ogsi:CreateServiceExtensibilityType>
               <createsInterface>Counter</createsInterface>
               <createsInterface>GridService</createsInterface>
               <createsInterface>NotificationSource</createsInterface>
            </ogsi:CreateServiceExtensibilityType>
         </content>
      </ogsi:EntryType>

  </sd:staticServiceDataValues>

</gwsdl:portType>
```

Notes on this WSDL:

1. This is GWSDL, the extended form of WSDL defined by OGSI. Basic WSDL won't do as we need to add elements of various namespaces into the *portType* element.
2. Elements with the *ogsi* prefix are defined in the OGSI standard.
3. The *sd:serviceData* element defines the form of the service data. The GWSDL for this element is fixed by OGSI standard.
4. The serviceGroupEntryLocator elements are all annulled. This registry is locally managed and doesn't use service instances with *ServiceGroupEntry* ports.
5. The *memberServiceLocator* elements contain handle elements which give the GSHs of the registered factories. The given GSHs follow the HTTP scheme for GSHs used by Globus Toolkit 3; other forms for handles are possible.

[To do: describe how a client finds a particular type of service using the registry.]

A client can use the registry to select factories for particular kinds of counter as follows.

Client invokes *findServiceData* on the registry's *GridService* port. Client includes this as the *queryExpression* parameter of the invoked operation:

```
<ogsi:queryByServiceDataNames>
   <name>ogsi:entry</name>
</ogsi:queryByServiceDataNames>
```

Service returns this as the *result* parameter of the operation:

```
<sd:serviceDataValues>
   <ogsi:EntryType>
      <serviceGroupEntryLocator nil="true"/>
      <memberServiceLocator>
```

```
<ogsi:handle>http://exemplar.org/ogsa/services/CounterFactory</ogsi:hand
le>
      </memberServiceLocator>
      <content>
         <ogsi:CreateServiceExtensibilityType>
            <createsInterface>Counter</createsInterface>
            <createsInterface>GridService</createsInterface>
         </ogsi:CreateServiceExtensibilityType>
      </content>
   </ogsi:EntryType>
   <ogsi:EntryType>
      <serviceGroupEntryLocator nil="true"/>
      <ogsi:memberServiceLocator>

<ogsi:handle>http://exemplar.org/ogsa/services/PrivateCounterFactory</og
si:handle>
      </memberServiceLocator>
      <content>
         <ogsi:CreateServiceExtensibilityType>
            <createsInterface>SecureCounter</createsInterface>
            <createsInterface>GridService</createsInterface>
         </ogsi:CreateServiceExtensibilityType>
      </content>
   </ogsi:EntryType>
   <ogsi:EntryType>
      <serviceGroupEntryLocator nil="true"/>
      <memberServiceLocator>

<ogsi:handle>http://exemplar.org/ogsa/services/SharedCounterFactory</ogs
i:handle>
      </memberServiceLocator>
      <content>
         <ogsi:CreateServiceExtensibilityType>
            <createsInterface>Counter</createsInterface>
            <createsInterface>GridService</createsInterface>
            <createsInterface>NotificationSource</createsInterface>
         </ogsi:CreateServiceExtensibilityType>
      </content>
   </ogsi:EntryType>
</sd:serviceDataValues>
```

I.e., the result of the operation is a dump of all the metadata for all the services.

Client runs these metadata through an XPath serach-engine (an XSLT processor is a likely implementation of this) using this query:

```
//ogsi:EntryType[content/ogsi:CreateServiceExtensibilityType/createsInte
rface="NotificationSource"]
```

to find all the factories for counters that do notification. Client this then left with

```
<ogsi:EntryType>
   <serviceGroupEntryLocator nil="true"/>
   <memberServiceLocator>
```

```
<ogsi:handle>http://exemplar.org/ogsa/services/SharedCounterFactory</ogs
i:handle>
   </memberServiceLocator>
   <content>
      <ogsi:CreateServiceExtensibilityType>
         <createsInterface>Counter</createsInterface>
         <createsInterface>GridService</createsInterface>
         <createsInterface>NotificationSource</createsInterface>
      </ogsi:CreateServiceExtensibilityType>
   </content>
</ogsi:EntryType>
```

Client runs a further XPath search using query

```
//ogsi:handle
```

to extract the GSH of the selected service.

6.1.4 Example 2: grid-managed registry of service instances

[To do: all of example]

6.2 Abstract ServiceGroups

[Explain the abstract concept, and illustrate with concrete examples]

6.3 Binding to a Service

[bootstrapping to find information about resources and factories]

[**Resolvers** this needs an example]

[Spec section 3.3 Client Use of Grid Service Handles and References]

A client gains access to a Grid service instance through Grid Service Handles and Grid Service References. A Grid Service Handle (GSH) can be thought of as a permanent network pointer to a particular Grid service instance. The GSH does not provide sufficient information to allow a client to access the service; the client needs to "resolve" a GSH into a Grid Service Reference (GSR). The GSR contains all the necessary information to access the service. The GSR is not a "permanent" network pointer to the Grid service instance because a GSR may become invalid for various reasons; for example, the Grid service instance may be moved to a different server.

OGSI provides a mechanism, the HandleResolver (see §10) to support client resolution of a Grid Service Handle into a Grid Service Reference. In Figure 6-1, a client application needs to resolve a GSH into a GSR.

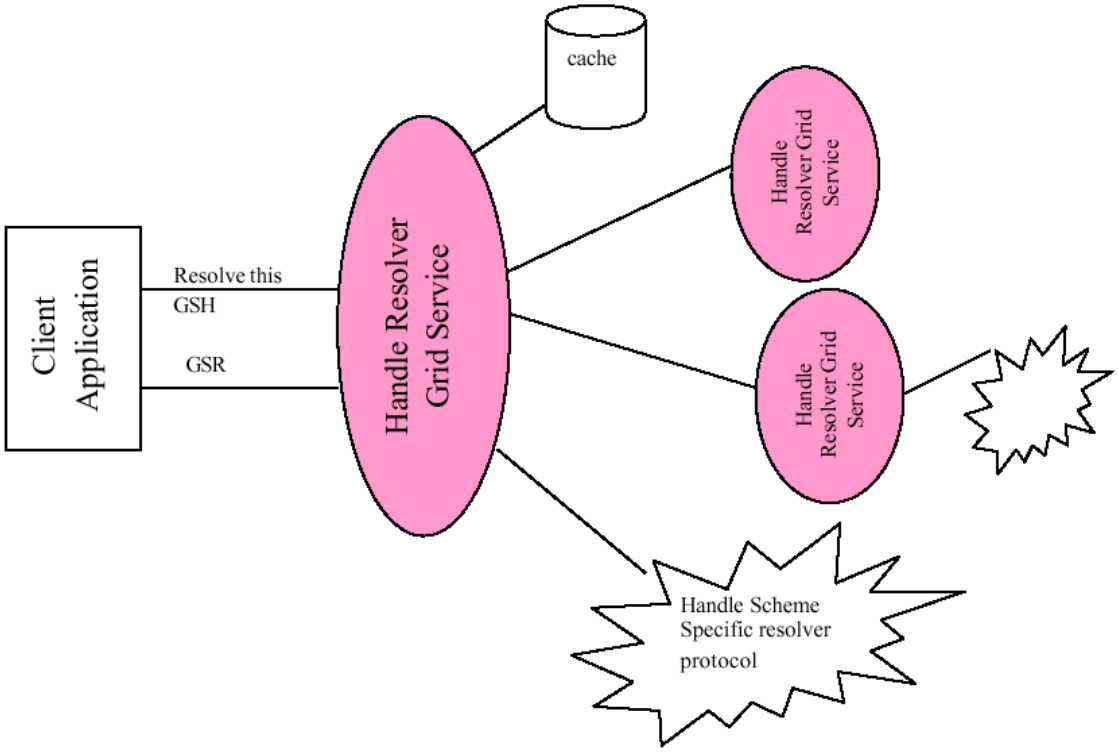

Figure 6-1: Resolving a GSH

The client resolves a GSH into a GSR by invoking a HandleResolver Grid service identif ied by some out-of-band mechanism. The HandleResolver can use various means to do the resolution; some of these means are depicted in Figure 2. The HandleResolver may have the GSR stored in a local cache. The HandleResolver may need to invoke another HandleResolver to resolve the GSH. The HandleResolver may use a handle resolution protocol, specified by the particular kind (or scheme) of the GSH to resolve to a GSR. The Handle resolver protocol is specific to the kind of GSH being resolved. For example, one kind of handle may suggest the use of HTTP GET to a URL encoded in the GSH order to resolve to a GSR.

7 The GridService portType

To be a Grid service a service must implement the interface and behavior that enables clients to manage it like other Grid Services. In summary, the behaviors allow clients to locate or create service instances, call them and arrange for their destruction when they are no longer needed. These behaviors are defined in the Grid Service Specification [3] as portTypes, beginning with the basic GridService which is common to all OGSI services.

7.1 The basic requirement: GridServiceportType

[This is based on section 9 of the spec].

We start with the GridService portType, which must be implemented by all Grid services and thus serves as the base interface definition in OGSA. This portType is analogous to the base Object class within object -oriented programming languages such as Smalltalk or Java, in that it encapsulates the root behavior of the component model. The behavior encapsulated by the GridService portType is that of

- The required elements the serviceDataSet and the semantics associated with these elements.
- Managing the termination of the instance

7.1.1 Terminology: Service Description and Service Instance

(This is from the spec, section 7.1)

We distinguish in OGSA between the description of a Grid service and an instance of a Grid service:

- A Grid service description describes how a client interacts with service instances. This description is independent of any particular instance. Within a WSDL document, the Grid service description is embodied in the most derivedportType (i.e. the portType referenced by the wsdl:service element describing the service) of the instance, along with its associated portTypes, serviceDataDescriptions, messages, and types definitions.
- A Grid service description may be simultaneously used by any number of Grid service instances, each of which:
    - embodies some state with which the service description describes how to interact;
    - has one or more Grid Service Handles;

- and has one or more Grid Service References to it.

A common form of Grid Service Reference (defined in section ?) is a WSDL document comprising a serviceelement, which carries an attribute that refers to a most derived portType defined by the service description of that instance.

A service description is primarily used for two purposes. First, as a description of a service interface, it can be used by tooling to automatically generate client interface proxies, server skeletons, etc. Second, it can be used for discovery, for example, to find a service instance that implements a particular service description, or to find a factory that can create instances with a particular service description.

The service description is meant to capture both interface syntax, as well as (in a very rudimentary, non -normative fashion) semantics. Interface syntax is, of course, described by portTypes.

Semantics may be inferred through the name assigned to the portType. For example, when defining a Grid service, one defines zero or more uniquely named portTypes, and then collects a set of portTypes defined from a variety of sources into a final or most derived portType. Concise semantics can be associated with each of these names in specification documents – and perhaps in the future through Semantic Web or other formal descriptions. These names can then be used by clients to discover services with the sought-after semantics, by searching for service instances and factories with the appropriate names. Of course, the use of namespaces to define these names provides a vehicle for assuring globally unique names.

7.2 GridService Service Data

[Enumerate the servicedata elements and describe their purpose.]

[This is from section 6.2 of the spec]

For example, the following portType declares two serviceData elements, with qualified names “tns:sd1” and “tns:sd2”. Any service that implements this portType MUST have as part of its state these two ServiceData
elements.

```
<wsdl:definitions xmlns:tns=”xxx” targetNamespace=”xxx”>
<gwsdl:portType name="exampleSDUse"> *
<wsdl:operation name=…>
…
<sd:serviceData name="sd1" type=”xsd:String”
mutability=”static”/>
<sd:serviceData name="sd2" type=”tns:SomeComplexType”/>
…
<sd:staticServiceDataValues>
<tns:sdl>initValue</tns:sd1>
</sd:staticServiceDataValues>
</gwsdl:portType>
…
</wsdl:definitions>
```

7.2.1 Using serviceData, an Example from GridService portType

[From section 6.2.2 of the Spec]

Let's examine how serviceData can be used by reviewing an example, namely the GridService portType, described in §9. The (non-normative) serviceData elements declared for the Grid Service portType are as follows.

```
<wsdl:definitions …
<gwsdl:portType name="GridService" …>
<wsdl:operation name= …>
…
<sd:serviceData name="interface" type="xsd:QName"
    minOccurs="1" maxOccurs="unbounded"
    mutability="constant"/>
<sd:serviceData name="serviceDataName" type="xsd:QName"
    minOccurs="0" maxOccurs="unbounded"
    mutability="mutable" nillable="false"/>
<sd:serviceData name="factoryHandle"
    type="ogsi:HandleType"
    minOccurs="1" maxOccurs="1
    mutability="constant" nillable="true"/>
<sd:serviceData name="gridServiceHandle"
    type="ogsi:HandleType"
    minOccurs="0" maxOccurs="unbounded"
    mutability="extendable"/>
<sd:serviceData name="gridServiceReference"
    type="ogsi:ReferenceType"
    minOccurs="0" maxOccurs="unbounded"
    mutability="mutable"/>
<sd:serviceData name="findServiceDataExtensibility"
    type="ogsi:OperationExtensibilityType"
    minOccurs="1" maxOccurs="unbounded"
    mutability="static"/>
<sd:serviceData name="terminationTime" type="ogsi:terminationTime"
    minOccurs="1" maxOccurs="1"
```

The normative description of the individual serviceData elements are in (§9.1).

The following is an example set of serviceData element values for a Grid service.

```
…
xmlns:crm="http://gridforum.org/namespaces/2002/11/crm"
xmlns:tns="http://example.com/exampleNS"
xnlns="http://example.com/exampleNS">
<sd:serviceDataValues>
<ogsi:interface>crm:GenericOSPT</ogsi:interface>
<ogsi:interface>ogsi:GridService</ogsi:interface>
<ogsi:serviceDataName>ogsi:interface
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:serviceDataName
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:factoryHandle
```

```
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:gridServiceHandle
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:gridServiceReference
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:findServiceDataExtensibility
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:terminationTime
</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:setServiceDataExtensibility
</ogsi:serviceDataName>
<ogsi:factoryHandle>someURI</ogsi:factoryHandle>
<ogsi:gridServiceHandle>someURI</ogsi:gridServiceHandle>
<ogsi:gridServiceHandle>someOtherURI</ogsi:gridServiceHandle>
<ogsi:gridServiceReference>…</ogsi:gridServiceReference>
<ogsi:gridServiceReference>…</ogsi:gridServiceReference>
<ogsi:findServiceDataExtensibility
inputElement="ogsi:queryByServiceDataNames" />
<ogsi:terminationTime after="2002-11-01T11:22:33"
before="2002-12-09T11:22:33"/>
<ogsi:setServiceDataExtensibility
inputElement="ogsi:setByServiceDataNames" />
```

7.2.2 ServiceData Initial values

See example text in the spec (section 6.3.1)

7.2.3 ServiceData and portType Inheritance

See example text in the spec (sections 6.4 and 6.4.1)

7.2.4 ServiceData Bindings

[From the spec section 6.2, last para]

The wsdl:binding associated with various operations manipulating serviceData elements will indicate the encoding of that data between service requestor and service provider. For example, a binding might indicate that the serviceData element values are encoded as serialized Java objects.

7.3 GridService Operations

[Summary of Querying and modyfing servicedata, termination time. Destrying a service. Sufficient to inform the following example]

7.3.1 Querying ServiceData

[From section 9.2.1.1 of the spec)

For example, a findServiceData invocation with this QueryExpression:

```
<ogsi:queryByServiceDataNames>
<name>ogsi:findServiceDataExtensibility</name>
<name>ogsi:setServiceDataExtensibility</name>
```

```
</ogsi:queryByServiceDataNames>
```

might return this Result:.

```
<sd:serviceDataValues>

<ogsi:findServiceDataExtensibility
inputElement="ogsi:queryByServiceDataNames"/>
<ogsi:setServiceDataExtensibility
inputElement="ogsi:setByServiceDataNames"/>
<ogsi:setServiceDataExtensibility>
inputElement="ogsi:deleteByServiceDataNames"/>
</sd:serviceDataValues>
```

7.3.2 GridService Examples

[Develop the example of the Counter with Servicedata defined]

8 Referencing and Handle Resolution

[Details on the The handle resolver portType – concepts are introduced earlier]

(This text came from an earlier version of the spec)

A handle resolver is a Grid service instance that implements the HandleResolver portType. Each GSH scheme defines a particular resolver protocol for resolving a GSH of that scheme to a GSR. Some schemes, such as the http and https, MAY not require the use of a HandleResolver service, as they are based on some other resolver protocol. However, there are two situations where a Grid service based resolver protocol MAY be used, and which therefore motivates the definition of a standard HandleResolver portType. First, a GSH scheme MAY be defined that uses the HandleResolver as a fundamental part of its resolver protocol, where the GSH carries information about to which HandleResolver service instance a client should send resolution requests. Second, in order to avoid placing undo burden on a client by requiring it to directly speak various resolver protocols, a client instead MAY be configured to outsource any GSH resolutions to a third party HandleResolver service. This outsourced handle resolver MAY in turn speak the scheme-specific resolver protocols directly. Both of these situations are addressed through the definition of the HandleResolver portType.

Various handle resolvers may have different approaches as to how they are populated with GSH to GSR mappings. Some handle resolvers may be tied directly into a hosting environment's lifetime management services, such that creation and destruction of instances will automatically add and remove mappings, through some out-of-band, hosting-environment-specific means. Other handle resolver services may implement the Registration portType, such that whenever a service instance registers its existence with the resolver, that resolver queries the GridServiceHandles and GridServiceReferences service data elements of that instance to construct its mapping database. Other handle resolver services may implement a custom registration protocol via a custom portType. But in all of these cases, the HandleResolver portType MAY be used to query the resolver service for GSH to GSR mappings.

[Suggestion: this is the place to answer questions about the relationship between a service implementation and the HandleResolver?. There is some input in OpFAQHandleResolution]

[Suggestion: is this the place to explain solutions to the bootstrap problem - how to get a reference to the handle resolver]

[Suggestion: explain the rationale behind the GSRExclusionSet? input to findByHandle]

[Question: In section 7.5.1.1. the spec recommends only including minumal info in the WSDL version of a GSR. What other information is possible? Why is it discouraged]

9 Finding Services: ServiceGroups and Registries

[This needs a major update following changes to the spec]

9.1 Reasons for Registries

Registries are sources of information about services (including resources) in a Grid. They are themselves Grid Services, and their function is to provide references to other services, perhaps based on some selection criteria such as the serviceType, resource capacity, availability or current load. The criteria aren't determined by the Grid Specification and depend on the registry and the kind of information it provides. Thes following examples give a flavor of how registries and service information can be organized.

- Central directories are the starting point for enquiries about Grid services and resources. Their locations may be well-known to potential clients and may keep lists of more local registries.
- Partner Catalog UDDI registry: Web services to be used by a particular company can be published to a Partner Catalog (rolodex like) UDDI registry. A Partner Catalog UDDI registry sits behind the firewall. This kind of private UDDI registry contains only approved, tested, and valid web service descriptions from legitimate business partners. The business context and metadata for these web services can be targeted to the specific requestor.
- Local directories can contain references to factories capable of creating services instances (including resource allocations, for example). Local information can contain more detailed, precise and/or dynamic information.
- Factories may keep a list of the service instances they have created.
- Instances keep information about their own current status which registries (or clients can use) to refresh information got from other sources.

The factory (penultimate) example shows how a registry can be only one of several functions provided by a service. The Grid Service Specification describes a Registration portType containing few basic operations that a registry must provide, but this can be combined with other portTypes or extended with additional operations and message types. Figure (2?) shows some of the alternatives described above.

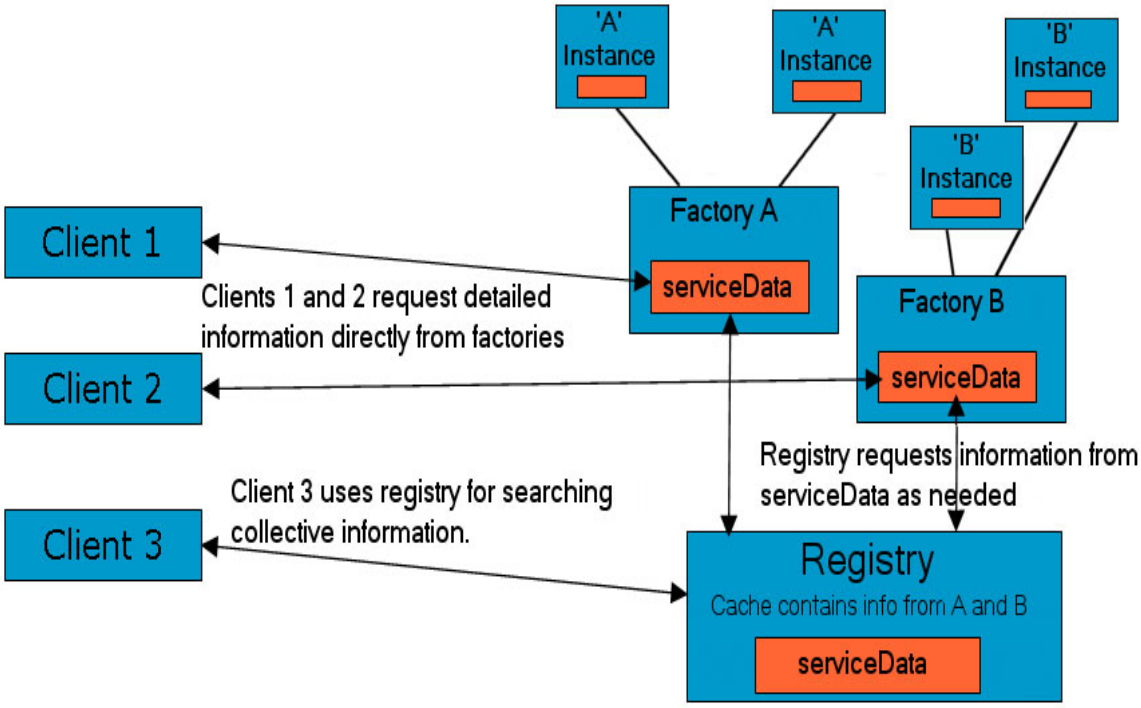

Figure 9-1: Factories and a dedicated Registry as information sources

9.2 The Registry Interfaces

There are two components to a registry – Registration and Discovery. Registration places a services reference in the registry, Discovery allows a client to retrieve it. Though it's logical to think of these as two parts of the same service, this is not necessarily the best way to organize the information. The Discovery service can use the registered references to find out more information (or more dynamic information, such as the current load) about the services, or pass this on to a third party which provides selection among a group of similar services, or qualities of its own such as scalability.

9.2.1 Registration PortType

Below is a diagram of the WSDL markup elements which describe the Service Group portType.

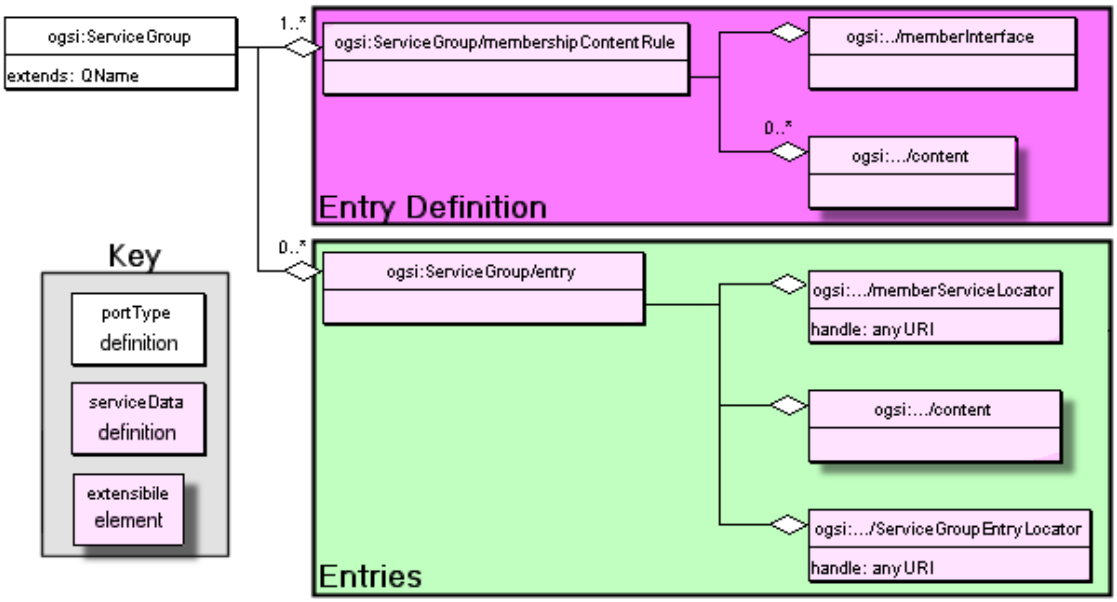

Figure 9-2: GWSDL Description of Service Group.

A registry service must implement the Registration portType which has the following operations

- Registration :: RegisterService

  Add or atomically update an entry in the registry. This operation takes as input

  - A **Locator** (Containing one or more handles and maybe references to the service) being registered.
  - Descriptive information about the service. The format of the information depends on the kind of registry and is described by ServiceDataElement values of the type gsdl:registrationExtensibilityType. An example might be a UDDI description of the service.
  - If the operation is successful, (no fault is generated) the newly registered service will subsequently be returned

- Registration :: UnregisterService

  Remove a Grid Service Handle from the registry.

9.2.2 Making Discoveries

Registration information is made available as a WS-Inspection (WSIL) document. – other forms of information may be available too, depending on the implementation of the discovery service. The WSIL document contains references to all of the services contained in the registry and can be retrieved from the registry as the serviceData

element GridServiceRegistryWSInspection using the normal operations for querying service Data.

9.2.3 Lifetime of Registration

As with other stateful interfaces in OGSA, GSH registration is a soft state operation and must be periodically refreshed, thus allowing discovery services to deal naturally with dynamic service availability.

This soft-state registration requires the service to periodically refresh is

9.3 An Example Registry

We need something realistic which illustrates sub-setting via attributes, eg selection of CPUs, based on ServiceData elements describing speed in Mhz?

.

9.4 Service Discovery and Invocation

What is the Bootstrap sequence for a client?

9.5 Service Registration

What does an instance have to do to be registered? Should it rely on the Factory to do it?

What does a factory have to do to be registered?

10 Creating Transient Services: The Factory

[The Factory as a pattern]

From a programming model perspective, a factory is an abstract concept or pattern. A factory is used by a client to create an instance of a Grid service. A client invokes a create operation on a factory and receives as response a serviceLocator for the newly created service. This specification defines one approach to realizing the factory pattern as a Grid service. OGSA uses a document-centric approach to define the operations of the basic factory. Service providers can, if they wish, define their own factories with specifically typed operation signatures.

In OGSA terms, a factory is a Grid service that MUST implement the Factory portType, which provides a standard WSDL operation for creation of Grid service instances. A factory MAY of course also implement other portTypes (in addition to the required GridService portType), such as:

• Registration (Section ??), which allows clients to inquire of the factory as to what Grid service instances created by the factory are in existence.

Upon creation by a factory, the Grid service instance MUST be registered with, and receive a GSH from, a handle resolution service (see Section 7). The method by which this registration is accomplished is specific to the hosting environment, and is therefore outside the scope of this specification.

10.1 The Factory Interface

11 GridService Notification

(This is from Section 11 in the spec)

The purpose of notification is to deliver interesting messages from a notification source to a notification sink, where:

- A notification source is a Grid service instance that implements the NotificationSource portType, and is the sender of notification messages. A source MAY be able to send notification messages to any number of sinks.
- A notification sink is a Grid service instance that receives notification messages from any number of sources. A sink MAY implement the DeliverNotification operation of the NotificationSink portType, which allows it to receive notification messages of any type. Alternatively, a sink MAY implement a specialized notification delivery operation from a different portType, where that operation is a specialization of the DeliverNotification operation. A specialized delivery operation MAY only accept a subset of the types of messages that the general DeliverNotification operation can accept, and like DeliverNotification is an input-only operation (i.e. it does not return a response).
- A notification message is an XML element sent from a notification source to a notification sink. The XML type of that element is determined by the subscription expression.
- A subscription expression is an XML element that describes what messages should be sent from the notification source to the notification sink. The subscription express also describes when messages should be sent, based on changes to values within a service instance's serviceDataSet.
- In order to establish what and where notification messages are to be delivered, a subscription request is issued to a source, containing a subscription expression, the serviceLocator of the notification sink to which notification messages are to be sent, the portType and operation name of the specialized notification delivery operation to which notification messages should be sent, and an initial lifetime for the subscription.
- A subscription request causes the creation of a Grid service instance, called a subscription, which implements the NotificationSubscription portType. This portType MAY be used by clients to manage the (soft-state) lifetime of the subscription, and to discover properties of the subscription.

This notification framework allows for either direct service-to-service notification message delivery, or for the ability to integrate various intermediary delivery services. Intermediary delivery services might include: messaging service products commonly used in the commercial world, message filtering services, message archival and replay services, etc.

11.1 Notification Interfaces

11.1.1 ServiceData for Notification

11.1.2 Notification Operations

12 Grid Services Security

Depending on the domain and the nature of the Grid services, a service implementor can and would implement layers of security - from transport level to message level to service level. In this context most probably one would require an interoperable security layer that spans multiple administrative domains.

12.1 Approach & Scope

This section addresses Grid Services security in a descriptive way (as opposed to prescriptive directions) and touches security only with respect to the primitives in OGSI/OGSA. This section :

1. summarizes the important ideas, topics and concepts (as resulted from discussions and specifications from the various grid security wgs)
2. provides links to detailed work in each of the above areas (thus neither duplicating nor reiterating the rest of the grid security work)
3. provides examples (abstract examples and patterns) how they could be used
4. will not have implementation details

The most important is to explain how security is factored out of the application (and is orthogonal to OGSI) and provide references to introductory material which explains (in a generic fashion) how security protocols can be plugged in to Client and Server infrastructure. What those infrastructures do with the security is not the business of OGSI, but some simple examples would be reassuring to the Primer audience.

12.2 List of Topics to address

1. ServiceData access, visibility and exposure controls - how to express, exchange and process serviceData security artifacts
2. WSDL security - i.e. security of the exposed interfaces including restricting operations
3. Standards for identity and access control policies
4. Examples of how to handle resolution protocols
5. Explain VOs and how they are achieved using Kerberos/X.509
6. X.509 and Kerberos usage in a grid scenario
7. Infosec issues - firewall traversal, any ACL and other requirements for grid installation
   a. Examples/explanation of how gateways can be constructed for Web services as part of a firewall.
8. Trust mechanisms - establish, bootstrap and use
   a. Some simple examples of authorization for operations, and the importance of propagation of the authorization would be reassuring to the Primer audience.

13 Advanced Topics

13.1 Advanced Registries [?]

13.2 Recommendations for Change Management

< Change management from the gss spec>

13.3 Describing Operation semantics

[How operations may interact with each other, where to document this]

13.4 Monitoring Execution

[nice example – Monitoring a Datamining operation?]

14 Glossary of Terminology

Thse terms and abbreviations are used in the text and are defined where they are first used. The list below may help you if you dip into the text without reading from the beginning.

TBD – To Be Done.

15 Comparison with other Distributed Architectures

[Comparison to Corba, EJB, BPEL?]

This was 3.1. Relationship to Distributed Object Systems in the spec.

As we describe in much more detail below, a given Grid service implementation is an addressable, and potentially stateful, instance that implements one or more interfaces described by WSDL portTypes. Grid service factories (§12) can be used to create instances implementing a given set of portType(s). Each Grid service instance has a notion of identity with respect to the other instances in the distributed Grid, (§7.5.2.1). Each instance can be characterized as state coupled with behavior published through type-specific operations. The architecture also supports introspection in that a client application can ask a Grid service instance to return information describing itself, such as the collection of portTypes that it implements.

Grid service instances are made accessible to (potentially remote) client applications through the use of a Grid Service Handle (§7.5.2) and a Grid Service Reference (§7.5.1). These constructs are basically network-wide pointers to specific Grid service instances hosted in (potentially remote) execution environments. A client application can use a Grid Service Reference to send requests (represented by the operations defined in the portType(s) of the target service) directly to the specific instance at the specified network-attached service endpoint identified by the Grid Service Reference.

We expect that in many situations, client stubs and helper classes isolate application programmers from the details of using Grid Service References. Some client side infrastructure software assumes responsibility for directing an operation to a specific instance that the GSR identifies.

Each of the characteristics introduced above (stateful instances, typed interfaces, global names, etc.) is frequently also cited as a fundamental characteristic of so-called *distributed object-based system*s. However, there are also various other aspects of distributed object models (as traditionally defined) that are specifically *not* required or prescribed by OGSI. For this reason, we do not adopt the term distributed object model or distributed object system then describing this work, but instead use the term Open Grid Services Infrastructure, thus emphasizing the connections that we establish with both Web services and Grid technologies.

Among the object-related issues that are not addressed within OGSI are implementation inheritance, service mobility, development approach, and hosting technology. The Grid service specification does not require, nor does it prevent, implementations based upon object technologies that support inheritance at either the interface or the implementation level. There is no requirement in the architecture to expose the notion of implementation inheritance either at the client side or the service provider side of the usage contract. In addition, the Grid service specification does not prescribe, dictate, or prevent the use of any particular development approach or hosting technology for the Grid service. Grid service providers are free to implement the semantic contract of the service in any technology and hosting architecture of their choosing.

We envision implementations in J2EE, .NET, traditional commercial transaction management servers, traditional procedural UNIX servers, etc. We also envision service implementations in a wide variety of programming languages that would include both object-oriented and non-object-oriented alternatives.

16 Editor Information

Tim Banks
IBM
Hursley Park, Winchester, UK. SO21 2JN.
Email: tim_banks@uk.ibm.com

17 Contributors

We gratefully acknowledge the contributions made to this document by the following people: Adbeslem Djaoui, Kate Keahey, Guy Rixon, Savas Parastatidis

Also, the authors of the OGSI Specification which provided source material: Steven Tuecke, Karl Czajkowski, Ian Foster, Jeffrey Frey, Steve Graham, Carl Kesselman, Tom Maguire, Thomas Sandholm, Dr. David Snelling and Peter Vanderbilt.

18 Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we've missed):

Marc Brooks, Krishna Sankar.

This work was supported in part by the North-East UK Regional e-Science, IBM, Rutherford Appleton Laboratory, UK

19 Document References

[1] *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, I. Foster, C. Kesselman, S. Tuecke, Authors. International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.
Available at http://www.globus.org/research/papers/anatomy.pdf.

[2] *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, I. Foster, C. Kesselman, J. Nick, S. Tuecke, Authors. Globus Project, 2002.
Available at http://www.globus.org/research/papers/ogsa.pdf

[3] *Open Grid Services Infrastructure (OGSI) (draft). February 17, 2003.* S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, P. Vanderbilt. Global Grid Forum.
Available at: http://www.ggf.org/ogsi-wg

[4] *Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001*
Available at http://www.w3.org/TR/wsdl

[5] *Secure Grid Naming Protocol (SGNP): Draft Specification for Review and Comment. GGF4 Submission May 3, 2002*
Available at http://sourceforge.net/projects/sgnp/

[6] *Java (TM) API for XML-Based RPC (JAX-RPC).*
Available at http://java.sun.com/xml/jaxrpc/docs.html

[7] *Welcome to WSIF: Web Services Inocation Framework*
Available at http://www.apache.org/wsif

[8] *OGSA. Ref from OpImplementingWebandGridServices*
Available at ?

[9] *Enterprise JavaBeans TM Specification, Version 2.1. Sun Microsystems*

Available at ?

20 Copyright Notice

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

GLOBAL GRID FORUM
office@gridforum.org
www.ggf.org

21 The Index