Open Grid Service Infrastructure (OGSI)

(prefinal)

Status of this Memo

This document provides information to the community regarding the specification of the Open Grid Service Infrastructure (OGSI). Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

Building on both Grid and Web services technologies, the Open Grid Services Infrastructure (OGSI) defines mechanisms for creating, managing, and exchanging information among entities called Grid services. Succinctly, a Grid service is a Web service that conforms to a set of conventions (interfaces and behaviors) that define how a client interacts with a Grid service. These conventions, and other OGSI mechanisms associated with Grid service creation and discovery, provide for the controlled, fault resilient, and secure management of the distributed and often long-lived state that is commonly required in advanced distributed applications. In a separate document, we have presented in detail the motivation, requirements, structure, and applications that underlie OGSI. Here we focus on technical details, providing a full specification of the behaviors and Web Service Definition Language (WSDL) interfaces that define a Grid service.
Full Copyright Notice

Copyright © Global Grid Forum (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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).
## Contents

1. Introduction ..................................................................................................................6  
2. Notational Conventions ...............................................................................................6  
3. Setting the Context .......................................................................................................7  
   3.1 Relationship to Distributed Object Systems ........................................................7  
   3.2 Client-Side Programming Patterns ......................................................................8  
   3.3 Relationship to Hosting Environment ..................................................................9  
4. The Grid Service ........................................................................................................11  
5. WSDL Extensions and Conventions ..........................................................................11  
6. Service Data ...............................................................................................................12  
   6.1 Motivation ..........................................................................................................12  
   6.2 Extending PortType with ServiceData...............................................................14  
      6.2.1 Structure of the ServiceData Declaration...................................................15  
      6.2.2 Using ServiceData, an Example from GridService portType .................16  
      6.2.3 Interpretation of the ServiceData Declaration Element .............................18  
      6.2.4 Mutability ...................................................................................................19  
   6.3 ServiceData Values ............................................................................................19  
      6.3.1 Defining Initial SDE Values within the PortType .................................20  
   6.4 ServiceData Element Aggregation within a PortType Inheritance Hierarchy ..21  
      6.4.1 Initial Values of Static ServiceData Elements within a PortType Inheritance Hierarchy ...........................................................................................................21  
   6.5 Dynamic ServiceData Elements ........................................................................23  
   7. Core Grid Service Properties .....................................................................................23  
   7.1 Service Description and Service Instance..........................................................24  
   7.2 Modeling Time in OGSA ...................................................................................24  
   7.3 XML Element Lifetime Declaration Properties.................................................25  
   7.4 Interface Naming and Change Management ......................................................28  
      7.4.1 The Change Management Problem............................................................28  
      7.4.2 Naming Conventions for Grid Service Descriptions .................................28  
   7.5 Naming Grid Service Instances: Handles and References .................................29  
      7.5.1 Grid Service Reference (GSR) ....................................................................29  
      7.5.1.1 WSDL Encoding of a GSR ....................................................................31  
      7.5.2 Grid Service Handle (GSH) .......................................................................31  
      7.5.2.1 Service Instance Sameness .....................................................................32  
      7.5.3 Locators ......................................................................................................32  
   7.6 Grid Service Lifecycle .......................................................................................33  
   7.7 Common Handling of Operation Faults .............................................................34  
8. Grid Service Interfaces ...............................................................................................36  
9. The GridService PortType .........................................................................................36  
   9.1 GridService PortType: Service Data Declarations and Elements ......................37  
   9.2 GridService PortType: Operations and Messages .............................................39  
      9.2.1 GridService :: FindServiceData .................................................................39  
      9.2.1.1 queryByMultipleServiceDataNames .....................................................39  
      9.2.2 GridService :: SetServiceData .................................................................40  
      9.2.2.1 setByServiceDataName .........................................................................41  
      9.2.2.2 deleteByServiceDataName ....................................................................41
9.2.2.3 setByMultipleServiceDataNames (optional) .........................................42
9.2.2.4 deleteByMultipleServiceDataNames (optional) ....................................42
9.2.3 GridService :: RequestTerminationAfter ...................................................43
9.2.4 GridService :: RequestTerminationBefore .................................................43
9.2.5 GridService :: Destroy ...............................................................................44

10 The HandleResolver PortType...............................................................................44
10.1 HandleResolver PortType: Service Data Descriptions ......................................45
10.2 HandleResolver PortType: Operations and Messages .......................................45
10.2.1 HandleResolver :: FindByHandle ..............................................................45

11 Notification ............................................................................................................46
11.1 The NotificationSource PortType ....................................................................47
11.1.1 NotificationSource PortType: Service Data Descriptions and Elements...47
11.1.2 NotificationSource PortType: Operations and Messages ............................47
11.1.2.1 NotificationSource :: Subscribe ............................................................47
11.2 The NotificationSubscription PortType .............................................................49
11.2.1 NotificationSubscription PortType: Service Data Descriptions .................49
11.2.2 NotificationSubscription PortType: Operations and Messages .................49
11.3 The NotificationSink PortType .........................................................................49
11.3.1 NotificationSink PortType: Service Data Descriptions .............................49
11.3.2 NotificationSink PortType: Operations and Messages ................................49
11.3.2.1 NotificationSink :: DeliverNotification ..............................................49

12 The Factory PortType ............................................................................................50
12.1 Factory PortType: Service Data Descriptions....................................................50
12.2 Factory PortType: Operations and Messages ..................................................50
12.2.1 Factory :: CreateService.............................................................................50

13 ServiceGroup .........................................................................................................51
13.1 The ServiceGroupEntry PortType ....................................................................51
13.1.1 ServiceGroupEntry: Service Data Declarations.......................................52
13.1.2 ServiceGroupEntry: Operations .................................................................52
13.2 The ServiceGroup PortType ...............................................................................52
13.2.1 ServiceGroup: Service Data Descriptions ..............................................53
13.2.2 ServiceGroup: Operations and Messages ..............................................55
13.2.2.1 ServiceGroup :: Add ...........................................................................55
13.2.2.2 ServiceGroup :: Remove ....................................................................56

14 Security Considerations .........................................................................................56

15 Change Log .............................................................................................................57
15.5 Draft 5 (11/04/2002) → Draft 6 (1/10/2003) ...................................................60
15.6 Draft 6 (1/10/2003) → Draft 7 (1/xx/2003) ....................................................60

ogsi-wg@gridforum.org
1 Introduction

The Open Grid Services Architecture (OGSA) integrates key Grid technologies with Web services mechanisms to create a distributed system framework based around the Open Grid Service Infrastructure (OGSI). A Grid service instance is a (potentially transient) service that conforms to a set of conventions (expressed as WSDL interfaces, extensions, and behaviors) for such purposes as lifetime management, discovery of characteristics, notification, and so forth. Grid services provide for the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications. OGSI also introduces standard factory and registration interfaces for creating and discovering Grid services.

In this document, we propose detailed specifications for the conventions that govern how clients create, discover, and interact with a Grid service. That is, we specify (a) how Grid service instances are named and referenced, (b) the interfaces (and associated behaviors) that define any Grid service and (c) the additional (optional) interfaces and behaviors associated with factories and service groups. We do not address how Grid services are created, managed, and destroyed within any particular hosting environment. Thus, services that conform to this specification are not necessarily portable to various hosting environments, but they can be invoked by any client that conforms to this specification (of course, subject to policy and compatible protocol bindings).

Our presentation here is deliberately terse, in order to avoid overlap with [Grid Physiology]. The reader is referred to [Grid Physiology] for discussion of motivation, requirements, architecture, relationship to Grid and Web services technologies, other related work, and applications.

This document has 4 major parts:

- Sections 1 thru 3 are introductory in nature, non-normative and add detail to the context set by [Grid Physiology] for this work.
- Sections 4, thru 7 introduce how the Grid services work uses the Web Services Description Language (WSDL), including a gwsdl extension to WSDL as a temporary measure until WSDL 1.2 is complete, the notion of serviceData to expose state data of a service, and other core Grid services concepts.
- Sections 8 thru 13 define various required and optional portTypes including GridService, HandleResolver, Notification, Factory, and Registration.
- Sections 14 thru 21 are miscellaneous concluding matter.

2 Notational Conventions

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC-2119 [RFC 2119].

This specification uses namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:ogsi-wg@gridforum.org">ogsi-wg@gridforum.org</a></td>
<td>6</td>
</tr>
</tbody>
</table>
Note: the ogsi-wg is discussing the need for a new temporary namespace vs the use of the WSDL 1.2 namespace: "http://www.w3.org/2003/01/wsdl" (see Section 5)

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URLs [RFC 2396].

The following abbreviations and terms are used in this document:

- **GSH**: Grid Service Handle, as defined in Section 7.5.
- **GSR**: Grid Service Reference, as defined in Section 7.5.
- **SDE**: Service Data Element, as defined in Section 7.2.
- The terms Web services, XML, SOAP, and WSDL are as defined in [Grid Physiology].

The term hosting environment is used in this document to denote the server in which one or more Grid service implementations run. Such servers are typically language and/or platform specific. Examples include native Unix and Windows processes, J2EE application servers, and Microsoft .NET.

### 3 Setting the Context

Although [Grid Physiology] describes overall motivation for the Open Grid Services Architecture (OGSA), this document describes its architecture at a more detailed level. We call the base for OGSA the Open Grid Services Infrastructure (OGSI). Correspondingly, there are several details we examine in this section that help put the remainder of the document in context. Specifically, we discuss the relationship between OGSI and distributed object systems, and also the relationship that we expect to exist between OGSI and the existing Web services framework, examining both the client-side programming patterns and a conceptual hosting environment for Grid services.

We emphasize that 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 certain details described in the other parts of this document.

#### 3.1 Relationship to Distributed Object Systems

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 (Section 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 system, (Section 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 (Section 7.5.2) and a Grid Service Reference (Section 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.

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 systems. 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 when 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.

### 3.2 Client-Side Programming Patterns

Another important issue that we feel requires some explanation, particularly for readers 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 multiple protocol bindings, encoding styles, messaging styles (RPC vs. document-oriented), and so on, for a given Web service.
Figure 1 depicts a possible (but not required) client-side architecture for OGSI. In this approach, there is a clear separation between the client application and the client-side representation of the Web service (proxy), including components for marshalling the invocation of a Web service over a chosen binding. In particular, the client application is insulated from the details of the Web service invocation by a higher-level abstraction: the client-side interface. Various runtime tools can take the WSDL description of the Web service and generate interface definitions in a wide-range of programming language specific constructs (e.g. Java interfaces). This interface is a front-end to specific parameter marshalling and message routing that can incorporate various binding options provided by the WSDL. Further, 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.

One example of this approach to Web services is Java API for XML-Based RPC [JAX-RPC]. Within the client application runtime, a proxy provides a client-side representation of remote service instance’s interface. Proxy behaviors specific to a particular encoding and network protocol (binding in Web services terminology) are encapsulated in a protocol (binding) -specific stub. Details related to the binding-specific access to the Grid service, such as correct formatting and authentication mechanics, happen here; thus, the application is not required to handle these details itself.

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

3.3 Relationship to Hosting Environment

OGSA does not dictate a particular service provider-side implementation architecture. A variety of approaches are possible, ranging from implementing the Grid service 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.

Figure 2: Two alternative approaches to the implementation of argument demarshalling functions in a Grid Service hosting environment

Figure 2 illustrates these differences by showing two different approaches to the implementation of argument demarshalling functions. We assume that, as is the case for many Grid services, the invocation message is received at a network protocol termination point (e.g., an HTTP servlet engine), which converts the data in the invocation message into a format consumable by the hosting environment. At the top of Figure 2, we illustrate two Grid services (the ovals) associated with 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. In some circumstances (the lower oval), the entire behavior of a Grid service is completely encapsulated within the component. In other cases (the upper oval), a component will collaborate with other server-side executables, perhaps through an adapter layer, to complete the implementation of the Grid services behavior. At the bottom of Figure 2, we depict another scenario wherein the entire behavior of the Grid service, including the demarshalling/decoding of the network message, has been encapsulated within a single executable. Although this approach may have some efficiency advantages, it provides little opportunity for reuse of functionality between Grid service implementations.
A container implementation may provide a range of functionality beyond simple argument
demarshalling. For example, the container implementation may provide lifetime management
functions, 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.

4 The Grid Service

The purpose of this document is to specify the interfaces and behaviors that define a Grid service.
In brief, a Grid service is a WSDL-defined service that conforms to a set of conventions relating
to its interface definitions and behaviors. Thus, every Grid service is a Web service, though the
converse of this statement is not true. In the following sections, we expand upon this brief
statement by:

• Introducing a set of WSDL conventions that we make use of in our Grid service
  specification; these conventions have been incorporated in WSDL 1.2 (W3C working
group document)
• Defining service data, which provides a standard way for representing and querying
  meta-data and state data from a service instance.
• Introducing a series of core properties of Grid service including:
  o Defining Grid service description and Grid service instance, as organizing
    principles for their extension and their use;
  o Defining how time is modeled in OGSI;
  o Defining the Grid Service Handle and Grid Service Reference constructs, which
    we use to refer to Grid service instances;
  o Defining a common approach for conveying fault information from operations;
  o Defining the lifecycle of a Grid service instance.

In the remainder of the document, we introduce various portTypes, starting with the GridService
portType that must be supported by any Grid service, and then proceeding with the remainder of
the optional portTypes that describe fundamental behaviors of Grid services.

5 WSDL Extensions and Conventions

~rewrite this to reflect:

• The conclusion to use WSDL 1.2 namespace directly, or describe a particular subset of
  WSDL 1.2 extensions that we exploit in OGSI (so called gswdl TEMPORARY
  namespace)
• gwsdl namespace as it documents how this work extends WSDL 1.1 in the direction set
  by WSDL 1.2. Intention is that gwsdl is totally replaced by WSDL 1.2 when WSDL 1.2
  is finished

This draft is based on extensions to the WSDL language proposed by the W3C Web Services
Description Working Group [WSDL 1.2]. In particular, we are relying upon the following new
constructs proposed for WSDL 1.2 (draft):

• open content model (extensibility elements appearing in each WSDL element)
• portType inheritance

ogsi-wg@gridforum.org
• operation name uniqueness that includes namespace qualifier.

6 Service Data

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 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 will endeavor 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. And, as is the case in object systems where encapsulation is observed, the declared state of a service may be accessed externally only through service operations defined as part of the service interface.

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 serviceData work extends WSDL allowing the designer to 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 queries against the service data of a stateful service instance implementing the foo interface.

Put simply, the service data declaration is the mechanism used to express the elements of publicly available state exposed by the service as part of its service interface. ServiceData elements are accessible through operations of the service interfaces such as those defined in this specification. Private internal state of the service is not part of the service interface and is therefore not represented through a service data declaration.

6.1 Motivation

The serviceData concept was introduced to provide a flexible, properties-style approach to access state data of a Web service. The serviceData concept is similar to the notion of an instance variable in object-oriented programming languages such as Java\textsuperscript{TM}, Smalltalk and C++ and most closely associated with a Java Bean property. However, scenarios from systems management and Grid computing exhibit entities that have a potentially large number of properties, with a need to operate across combinations of those properties. If we wish to model these entities using Web services, we need an approach that models this complex state effectively.

An alternative approach, of providing "get operation per property" was considered, but not adopted for several reasons:

1. If we use a "get" operation per property, then the set of messages to get multiple properties is chatty, does not allow atomicity to be addressed, and does not allow queries across multiple properties. Example: A Web service modeling an operatingSystem
object contains (among other properties) the following four properties:
\texttt{totalVirtualMemorySize, freeVirtualMemory, freePhysicalMemory,}
\texttt{totalVisibleMemorySize}. Two of the properties, \texttt{totalVirtualMemorySize} and \texttt{totalVisibleMemorySize}, are constant and set at boot time. The other two properties, \texttt{freeVirtualMemory} and \texttt{freePhysicalMemory}, change over time as memory is allocated and de-allocated to a process. Consider the case wherein a management application is monitoring (and displaying) memory usage, \texttt{totalVirtualMemorySize} and \texttt{totalVisibleMemorySize} can be read individually since both are constant in a running system. Since \texttt{freeVirtualMemory} and \texttt{freePhysicalMemory} do change and change with respect to each other, these two properties need to be obtained together atomically in a single get operation; otherwise, the usage displayed of physical and virtual memory for a given point in time is incorrect. We could just define an operation that does a get on both \texttt{freeVirtualMemory} and \texttt{freePhysicalMemory}, but this approach does not scale with the number of properties (see next point).

2. If we then compel portType designers to define additional operations to get all the combinations of the serviceData, the number of operations is voluminous. Although the above example is simple, in reality there are many other properties in the \texttt{operatingSystem} object, many of which have exactly the same behavior described in the example in 1, and depending on exactly what you monitor from the complete \texttt{operatingSystem} object would require all combinations of dynamically changing properties, e.g. 10 properties (from \texttt{operatingSystem}) would yield 10! operations.

By supporting a serviceData approach, we provide the mechanism for requestors (e.g. management applications) to build query expressions (e.g. XPath and XQuery) to express queries involving combinations of service data elements. Example: The \texttt{operatingSystem} resource contains (among other properties) the following three properties: \texttt{totalSwapSpaceSize, sizeStoredInPagingFiles, and freeSpaceInPagingFiles}. The \texttt{filesystem} resource contains (among other properties) the following two properties: \texttt{fileSystemSize} and \texttt{availableSpace}. These two resources are related since \texttt{filesystem} is contained in or booted from \texttt{operatingSystem}. The properties \texttt{fileSystemSize} and \texttt{totalSwapSpaceSize} are typically constant, even between system boots. The other properties in this example change over time and with respect to each other, e.g. as \texttt{sizeStoredInPagingFiles} increases, \texttt{freeSpaceInPagingFiles} and \texttt{availableSpace} decreases. To monitor filesystem usage and display that usage, the query spans instances of both objects (\texttt{operatingSystem} and \texttt{filesystem}) and multiple properties within those instances. Single 'get' operations do not solve the problem of retrieving data across multiple object instances atomically.

There is also a requirement to support dynamic addition of "properties" at runtime. The notion is that the interface (portType) defines the majority of the serviceData elements (properties), however it is possible that at runtime, perhaps associated with a particular lifecycle state, to add serviceData elements to the Web service instance. Example: A Web service contains its lifecycle state in a serviceData element where the possible lifecycle values are 'exists', 'running', 'maintenance', and 'failed'. There is additional and different serviceData associated with each of these lifecycle states. In the failed lifecycle state, there is an additional serviceData element that contains the 'debug stack'. In the maintenance lifecycle state, the additional serviceData element is the list of data to help process the maintenance state. The 'exists' and 'running' lifecycle states
have no additional serviceData elements. In Web services, when serviceData elements exist in a
portType, they are expected to be valid. In this example where the information associated with a
lifecycle state is variable, use of dynamic serviceData for access to that additional information is
a good solution because the right needed information is returned based on lifecycle state. A
counter argument would be to define an operation `getLifecycleState` to accomplish the
same, but it is not the state itself that that operation would get, but rather the five (or however
many) new serviceData elements associated with that new state: you wouldn't want to model the
aggregation of those five properties into a `getLifecycleState` given the variability of
serviceData information associated with each lifecycle state.

### 6.2 Extending PortType with ServiceData

ServiceData defines a new child element of a portType named serviceData. This element defines
serviceData elements associated with that portType. Optionally, initial values for those
serviceData elements (marked as “static” serviceData elements) can be specified using the
`staticServiceDataValues` element within portType.

```xml
<gsdl:portType name="ncname"> *
  <wsdl:documentation .... /> ?
  <wsdl:operation name="ncname"> *
  ...
  <sd:serviceData name="ncname" ... /> *
  <sd:staticServiceDataValues>? 
  <some_element>*
  </sd:staticServiceDataValues>
  ...
</wsdl:portType>
```

The addition of serviceData elements in a portType declares the set of serviceData elements
associated with the portType. These serviceData elements are referred to as serviceData
declarations, or SDDs. Note the use of the `gsdl:portType`, which allows the appearance of
elements from other namespaces.

For example, a portType declares a set of serviceData declarations such as shown below:

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

The addition of serviceData elements in a portType declares the set of serviceData elements
associated with the portType. The serviceData element children of a portType element are
referred to as serviceData declarations, or SDDs. Any service that implements the portType
named `exampleSDUse` MUST have as part of its state the serviceData elements with qualified
names “tns:sd1” and “tns:sd2”. These components of a service instance’s state are called serviceData elements or SDEs. The value(s) of any serviceData element, whether declared statically in the portType or assigned during the life of the Web service instance are called serviceData element values or SDE values.

6.2.1 Structure of the ServiceData Declaration

The definition for serviceData element is:

```xml
<element name="serviceData" type="ServiceDataType"/>
```

```xml
<xsd:complexType name="ClosedServiceDataElementType">
  <xsd:complexContent>
    <xsd:restriction base="xsd:element">
      <xsd:sequence>
        <xsd:element ref="xsd:annotation" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:NCName"/>
      <xsd:attribute name="type" type="xsd:QName" />
      <xsd:attributeGroup ref="xsd:occurs" />
      <xsd:attribute name="nillable" type="xsd:boolean" use="optional" default="false" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

```xml
<xsd:complexType name="ServiceDataType">
  <xsd:complexContent>
    <xsd:extension base="ClosedServiceDataElementType">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="mutability" default="extend">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="static"/>
            <xsd:enumeration value="constant"/>
            <xsd:enumeration value="extend"/>
            <xsd:enumeration value="mutable"/>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="modifiable" type="xsd:boolean" default="false" />
      </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

```xml
<element name="serviceData" type="ServiceDataType"/>
```

```xml
<xsd:complexType name="ServiceDataValuesType"/>
```
A serviceData element is restricted to contain only these properties from xsd:element:

- annotation
- name
- type
- minOccurs
- maxOccurs
- nillable
- open attribute content model

A serviceData element extends this restriction of xsd:element to allow open element content model, and to define the mutability and modifiable attributes (6.2.4).

### 6.2.2 Using ServiceData, an Example from GridService portType

Let's examine how serviceData can be used by reviewing an example portType: the GridService portType, described in detail in Section 9. The serviceData elements declared for the Grid Service portType are shown below:

```xml
<wsdl:definitions ...>
  <gwsdl:portType name="GridService" ...>
    <wsdl:operation name="...">
      ...
      <sd:serviceData name="PortType" type="ogsi:portTypeNameType" minOccurs="1" maxOccurs="unbounded" mutability="constant"/>
      <sd:serviceData name="serviceDataName" type="xsd:QName" minOccurs="0" maxOccurs="unbounded" mutability="mutable"/>
      <sd:serviceData name="FactoryHandle" type="ogsi:gridServiceHandleType" minOccurs="0" mutability="constant" nillable="true"/>
      <sd:serviceData name="GridServiceHandle" type="ogsi:gridServiceHandleType" minOccurs="0" maxOccurs="unbounded" mutability="extend"/>
      <sd:serviceData name="GridServiceReferences" type="ogsi:gridServiceReferenceType" minOccurs="0" maxOccurs="unbounded" mutability="mutable"/>
      <sd:serviceData name="QueryExpressionType" type="xsd:QName"/>
    </wsdl:operation>
  </gwsdl:portType>
</wsdl:definitions>
```
minOccurs="1" maxOccurs="unbounded"
mutability="extend"/>
</sd:serviceData>
<sd:serviceData name="TerminationTime" type="ogsi:terminationTime"
minOccurs="1" maxOccurs="1"
mutability="mutable"/>
</sd:serviceData>
<sd:serviceData name="CurrentTime" type="ogs:currentTime"
minOccurs="1" maxOccurs="1"
mutability="mutable"/>
</sd:serviceData>
<sd:serviceData name="UpdateExpressionType"

And an example set of serviceData element values for some Grid service might look like:

... xmlns:crm="http://gridforum.org/namespaces/2002/11/crm"
xmlns:tns="http://example.com/exampleNS"
xlns="http://example.com/exampleNS">
<sd:serviceDataValues>
<ogsi:PortType>crm:GenericOSPT</ogsi:PortType>
<ogsi:PortType>ogsi:GridService</ogsi:PortType>
<ogsi:serviceDataName>ogsi:PortType</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:ogsiserviceDataName</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:FactoryHandle</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:ogsiGridServiceHandle</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:GridServiceReferences</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:QueryExpressionType</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:TerminationTime</ogsi:serviceDataName>
<ogsi:serviceDataName>ogsi:CurrentTime</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:QueryExpressionType>
  ogsi:queryByMultipleServiceDataNames
</ogsi:QueryExpressionType>
<ogsi:TerminationTime after="2002-11-01T11:22:33"
  before="2002-12-09T11:22:33"/>
6.2.3 Interpretation of the ServiceData Declaration Element

The serviceData declaration element is very similar to the declaration of an element in XML Schema. Therefore we use a restriction of the xsd:element declaration from XML Schema to declare serviceData elements.

- **maxOccurs = (nonNegativeInteger | unbounded) : default to 1**
  - This value indicates the maximum number of serviceData element values that can appear in the service instance’s serviceDataValues or the portType staticServiceDataValues.

- **minOccurs = nonNegativeInteger : default to 1**
  - This value indicates the minimum number of serviceData element values that can appear in the service instance’s serviceDataValues or the portType staticServiceDataValues.
  - If the value is 0, then the serviceData element is optional

- **name = NCName and {target namespace}**
  - The name of the serviceData element must be unique amongst all sd:serviceData and xsd:element declarations in the target namespace of the wsdl:definitions element.
  - The combination of the name of the serviceData element and the target namespace of the wsdl:definitions element’s targetNamespace attribute forms a QName, allowing a unique reference to this serviceData element.

- **nillable = boolean : default to false**
  - Indicates whether the serviceData element can have a nil value (that is a value that has an attribute xsi:nil with value="true")
    - For example a serviceData declaration
      `<serviceDataElement name="foo" type="xsd:string" nillable=true" />
    - can have a valid SDE value
      `<foo xsi:nil="true"/>

- **type = QName**
  - Defines the XML schema type of the serviceData element

- **modifiable = “boolean” : default to false**
  - If true, it is legal for requestors to directly update the serviceData value through the SetServiceData operation (see Section 9.2.2), subject to constraints on cardinality (minOccurs, maxOccurs) and mutability. If false, the serviceData element should be regarded as “read only” by the requestor, though its values may change as a result of other operations on the service’s interface.

- **mutability = “static” | “constant” | “extend” | “mutable” : default to extend**
An indication on how the values of a serviceData element can change. See (Section 6.2.4)

- {any attributes with non-schema namespace}
  - Open content on the attributes of serviceData declaration will be allowed.

- Content
  - annotation
    - This element allows documentation elements to appear as children of a serviceData declaration
  - Open content element model, meaning elements from any other namespace (besides XML Schema) may appear as child elements of the serviceData element.

### 6.2.4 Mutability

We provide a mutability attribute on the serviceData element declaration. This attribute indicates how a serviceData element’s values may change over the lifetime of the instance

- mutability="static": this implies that the SDE value is assigned in the WSDL declaration (staticServiceDataValues) and remains that value for any instance of that portType. A “static” SDE is analogous to a class member variable in programming languages.
- mutability="constant": this implies that the SDE value is assigned upon creation of the Grid service instance and MUST not change during the lifetime of the Grid service instance once it is set to a value.
- mutability="extend": this implies that once elements are in the SDE value, they are guaranteed to be part of the SDE value for the lifetime of the Grid service. New elements can be added to the SDE value, but once these elements are added, they cannot be removed.
- mutability="mutable": this implies any of the elements in the SDE value MAY be removed at anytime, and others MAY be added.

Note: the functionality described here is different than the fixed and default attributes on the element definition in XML Schema. Fixed could be used to suggest a static value, but append and mutable would have to be modeled by a mutability attribute. The case where mutability = “constant” would be used is to specify a property that does not change after a value is assigned (but the value is not assigned by the service description, but rather, it must be initialized at runtime).

### 6.3 ServiceData Values

Each service instance is associated with a collection of serviceData elements. The set of serviceData elements are defined within the various portTypes that form the interface of the service and may also be dynamically extended to include additional serviceData elements (see 6.5). We call the set of serviceData elements associated with a service instance a “serviceData set”. A serviceData set may also refer to the set of serviceData elements aggregated from all of the serviceData elements declared in a portType extension hierarchy (see 6.4).

Each service MUST convey a “logical” XML document, with a root element of serviceDataValues that contains the serviceData element values. An example of a serviceDataValues element is shown above. The implementation of the service is free to choose how the SDE values are stored; for example, it is not obliged to store the SDE values as XML, but instead as instance variables that are converted into XML or other encodings as necessary.
Furthermore, 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.

### 6.3.1 Defining Initial SDE Values within the PortType

A portType MAY declare initial values for any serviceData element with mutability marked as “static” in its serviceData set, regardless of whether the serviceData element was declared locally, or in one of the portTypes it extends. Further, initial values MAY be declared in multiple portTypes within the extension hierarchy, so long as the total number of initial values does not exceed that element’s maxOccurs constraint. In this case, the initial values for this serviceData element is the collection of all of the initial values.

For example, the following is legal:

```xml
<wsdl:definitions xmlns:tns="xxx" targetNamespace="xxx">
  <gwsl:portType name="otherPT">
    <wsdl:operation name=...>
      ...
      <sd:serviceData name="otherSD" type="xsd:String"
                      mutability="static" maxOccurs="unbounded"/>
    </wsdl:operation>
    <sd:staticServiceDataValues>
      <xxx:otherSD>initial value 1</xxx:otherSD>
    </sd:staticServiceDataValues>
  </gwsl:portType>
  ...
  <gwsl:portType name="exampleSDUse" extends "xxx:otherPT">
    <wsdl:operation name=...>
      ...
      <sd:serviceData name="sd1" type="xsd:String"
                      mutability="static"/>
      <sd:serviceData name="sd2" type="tns:SomeComplexType"/>
    </wsdl:operation>
    <sd:staticServiceDataValues>
      <xxx:sd1>an initial value</xxx:sd1>
      <xxx:otherSD>initial value 2</xxx:otherSD>
    </sd:staticServiceDataValues>
  </gwsl:portType>
  ...
</wsdl:definitions>
```

In this example, there are two initial values for xxx:otherSD: “initial value 1” and “initial value 2”.

Initial values MUST NOT be declared for a serviceData element with mutability other than “static”.

ogsi-wg@gridforum.org
6.4 ServiceData Element Aggregation within a PortType Inheritance Hierarchy

WSDL 1.2 has introduced the notion of multiple portType extension; we have modeled that construct within the gwsdl namespace. A portType can extend 0 or more other portTypes. There is no direct relationship between a wsdl:service and the portTypes supported by the service modeled in the WSDL syntax. Rather, the set of portTypes implemented by the service is derived through the port element children of the service element and binding elements referred to from those port elements. This set of portTypes, and all portTypes they extend, defines the complete interface to the service.

The serviceData set defined by the service’s interface is the set union of the serviceData elements declared in each portType in the complete interface implemented by the Web service. Because serviceData elements are uniquely identified by QName, the set union semantic implies that a serviceData element can appear only once in the set of serviceData elements. For example if a portType named “pt1” and portType named “pt2” both declare a serviceData named “tns:sd1”, and a portType named “pt3” extends both “pt1 and “pt2” then it has one (not two) serviceData elements named “tns:sd1”.

Consider the following example:

```xml
<gwsdl:portType name="pt1">
   <sd:serviceData name="sd1" …/>
</gwsdl:portType>

<gwsdl:portType name="pt2" extends "pt1">
   <sd:serviceData name="sd2" …/>
</gwsdl:portType>

<gwsdl:portType name="pt3" extends "pt1">
   <sd:serviceData name="sd3" …/>
</gwsdl:portType>

<gwsdl:portType name="pt4" extends "pt2 pt3">
   <sd:serviceData name="sd4" …/>
</gwsdl:portType>
```

The serviceData sets defined by each portType is summarized as follows:

<table>
<thead>
<tr>
<th>if a service implements…</th>
<th>its serviceData set contains…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt1</td>
<td>sd1</td>
</tr>
<tr>
<td>Pt2</td>
<td>sd1, sd2</td>
</tr>
<tr>
<td>Pt3</td>
<td>sd1, sd3</td>
</tr>
<tr>
<td>Pt4</td>
<td>sd1, sd2, sd3, sd4</td>
</tr>
</tbody>
</table>

6.4.1 Initial Values of Static ServiceData Elements within a PortType Inheritance Hierarchy

Initial values of static SDEs can be aggregated down a portType extension hierarchy. However, the cardinality requirements (minOccurs and maxOccurs) MUST be preserved.

For example:
A Web service instance that implements pt1 would have the value &lt;sd1&gt;1&lt;/sd1&gt; for SDE named sd1.

A Web service instance that implements pt2 would inherit the value &lt;sd1&gt;1&lt;/sd1&gt; for SDE named sd1 and would have the value &lt;sd2&gt;2&lt;/sd2&gt; for the SDE named sd2.

A Web service instance that implements pt3 would have two values &lt;sd3&gt;3a&lt;/sd3&gt; and &lt;sd3&gt;3b&lt;/sd3&gt; for the SDE named sd3. It would of course inherit the value for the SDE named sd1.

A Web service instance that implements pt4 would inherit the value for sd1 defined in pt1, but the absence of a staticServiceDataValues element implies that there is no initial value for sd4 (although it is most likely that one would be defined in a portType which extends pt4).

A Web service instance that implements pt5 could not be created. Since there is no initial value for sd5, and the minOccurs value is greater than zero, an error is generated when the instance is created. PortTypes of this sort can be encountered if it is the intention of the designer to declare
an “abstract” portType, wherein portTypes extending the abstract portType define concrete values for SDEs with minOccurs greater than zero.

```xml
<gwSDL:portType name="pt6" extends "pt1">
  <sd:staticServiceDataValues>
    <sd1>6</sd1>
  </sd:staticServiceDataValues>
</gwSDL:portType>
```

A Web service instance that implements pt6 could not be created. Since this portType declares an additional value for the SDE named sd1 (recall there is a value inherited from pt1) which exceeds the maxOccurs value for the SDE named sd1, an error is generated when the instance is created. PortTypes of this sort are in error and the designer should be ridiculed.

```xml
<gwSDL:portType name="pt7" extends "pt2 pt3">
  <sd:serviceData name="sd7" minOccurs="1" maxOccurs="1" mutability="static"/>
  <sd:staticServiceDataValues>
    <sd7>7</sd7>
    <sd3>7</sd3>
  </sd:staticServiceDataValues>
</gwSDL:portType>
```

A Web service instance that implements pt7 has a very interesting set of serviceData element values. First, it has a single value <sd1>1</sd1> for the SDE named sd1. Despite inheriting pt1 via pt2 and pt3, the initial values for sd1 are not repeated. The value <sd2>2</sd2> is the only value for the SDE named sd2, this is inherited from pt2. The SDE named pt3 has 3 values: <sd3>3a</sd3>, <sd3>3b</sd3> (inherited from pt3) and <sd3>7</sd3> locally defined. And finally, there is a locally defined value for the SDE named sd7 (<sd7>7</sd7>).

In general, values for static SDEs are aggregated down a portType extension hierarchy. If the resulting set of SDE values violates the cardinality of the SDE (the number of values is either less than the value of minOccurs, or greater than the value of maxOccurs), an error is reported when a Web service instance is created.

### 6.5 Dynamic ServiceData Elements

Although many serviceData elements are defined in the Web service’s interface definition, there are situations that surface where serviceData elements can be dynamically added to or removed from an instance. The means by which the serviceData set of an instance may be updated is implementation specific. Note, in GridService portType (Section 9), there is a serviceData element named “serviceData” that lists the serviceData elements currently defined, allowing the requestor to use the subscribe operation (Section 11.1.2.1) if this serviceDataSet changes, and findServiceData (Section 9.2.1) operation to determine the current serviceDataSet value.

### 7 Core Grid Service Properties

In this section we discuss a collection of properties or concepts common to all Grid services.
7.1 Service Description and Service Instance

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 derived portType (i.e. the portType referenced by the wsdl:service element’s port children (via referenced binding elements) describing the service) of the instance, along with its associated portTypes (including serviceData declarations), bindings, 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 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 WSDL 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. 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 Modeling Time in OGSA

Issue 55: WS-Timestamp is a component of WS-Security that is being standardized in OASIS. We need to examine how OGSI is modelling time and reconcile.

Throughout this specification there is the need to represent time that is meaningful to multiple parties in the distributed Grid. For example: information may be tagged by a producer with timestamps in order to convey that information’s useful lifetime to consumers; clients need to negotiate service instance lifetimes with services; and multiple services may need a common understanding of time in order for clients to be able to manage their simultaneous use and interaction.

The GMT global time standard is assumed for Grid services, allowing operations to refer unambiguously to absolute times. However, assuming the GMT time standard to represent time does not imply any particular level of clock synchronization between clients and services in the Grid. In fact, no specific accuracy of synchronization is specified or expected by this specification, as this is a service-quality issue.
Grid service hosting environments and clients SHOULD utilize the Network Time Protocol (NTP) or equivalent function to synchronize their clocks to the global standard GMT time. However, clients and services MUST accept and act appropriately on messages containing time values that might be out of range due to inadequate synchronization, where “appropriately” MAY include refusing the use the information associated with those time values. Furthermore, clients and services requiring global ordering or synchronization at a finer granularity than their clock accuracies or resolutions allow for MUST coordinate through the use of additional synchronization service interfaces, such as through transactions or synthesized global clocks.

In some cases it is required to model both zero time and infinite time (for examples, see Sections 7.3 and 9.2). To model zero time, a time in the past or ‘now’ should be used. However, infinite time requires an extended notion of time. We therefore introduce the following type in the ogsi namespace.

```xml
... targetNamespace = "http://www.gridforum.org/namespaces/2003/OGSI"
<xsd:simpleType name="extendedDateTime">
    <xsd:union memberTypes="xsd:dateTime">
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMToken">
                <xsd:enumeration value="infinity"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

### 7.3 XML Element Lifetime Declaration Properties

Since serviceData elements may represent point-in-time observations of dynamic state of a service instance, it is critical that consumers of serviceData be able to understand the valid lifetimes of these observations. The client MAY use this time-related information to reason about the validity and availability of the serviceData element and its value, though the client is free to ignore the information at its own discretion.

We define three XML attributes, which together describe the lifetimes associated with an XML element and its sub-elements. These attributes MAY be used in any XML element that allows for extensibility attributes, including the serviceData element.

The three lifetime declaration properties are:

- **ogsi:goodFrom**: Declares the time from which the content of the element is said to be valid. This is typically the time at which the value was created.
- **ogsi:goodUntil**: Declares the time until which the content of the element is said to be valid. This property MUST be greater than or equal to the goodFrom time.
- **ogsi:availableUntil**: Declares the time until which this element itself is expected to be available, perhaps with updated values. Prior to this time, a client SHOULD be able to obtain an updated copy of this element. After this time, a client MAY no longer be able to get a copy of this element (while still observing cardinality and mutability constraints on this element). This property MUST be greater than or equal to the goodFrom time.

The XML definitions for these attributes in the ogsi namespace are as follows:

```xml
... targetNamespace = "http://www.gridforum.org/namespaces/2003/OGSI"
```
<xsd:attribute name="goodFrom" type="ogsi:extendedDateTime"/>
<xsd:attribute name="goodUntil" type="ogsi:extendedDateTime"/>
<xsd:attribute name="availableUntil" type="ogsi:extendedDateTime"/>

The definition of extendedDateTime is given in section 7.2.

We use the following serviceData element example to illustrate and further define these lifetime declaration attributes:

```xml
<wsdl:definitions
    targetNamespace="http://example.com/ns"
    xmlns:n1="http://example.com/ns"
>
    <wsdl:types>
        <xsd:schema ...>
            "targetNamespace=http://example.com/ns"
            ...
        </xsd:schema>
    </wsdl:types>

    <ogsi:serviceDataDescription
        name="MySDE"
        ...
    </ogsi:serviceDataDescription>

    <gwsdl:portType name="MyPortType">
        ...
        <sd:serviceData name="MySDE" type= "n1:MyType"
            minOccurs="1" maxOccurs="1"
            mutability="mutable"/>
        ...
    </gwsdl:portType>

    <gwsdl:portType>
        ...
    </gwsdl:portType>
</wsdl:definitions>
```

And within the service instance’s serviceDataValues:

```xml
<sd:serviceDataValues
    <n1:MySDE
        goodFrom="2002-04-27T10:20:00.000-06:00"
        goodUntil="2002-04-27T11:20:00.000-06:00"
        availableUntil="2002-04-28T10:20:00.000-06:00">
        <n1:e1>
            abc
        </n1:e1>
        <n1:e2 ogsi:goodUntil="2002-04-27T10:30:00.000-06:00">
            def
        </n1:e2>
        <n1:e3 ogsi:availableUntil="2002-04-27T20:20:00.000-06:00">
```

ogsi-wg@gridforum.org
The goodFrom and goodUntil attributes of the n1:MySDE element refer to all of its child elements. These attributes declare to the consumer of this SDE what the expected lifetime is for this element’s value, which in this example is from 10:20am until 11:20am EST on 27 April 2002. In other words, the consumer of the SDE is being advised that after 1 hour the service data value is likely to no longer be valid, and therefore the client should query the service again for the SDE with the same name (n1:MySDE) to obtain a newer value.

The availableUntil does not refer to the service data value of the SDE, but rather to the availability of this named serviceData element itself. Prior to the declared availableUntil time, a client SHOULD be able to query the same Grid service instance for an updated value of this named SDE. In this example, a client should be able to query the same service until 28 April 2002 10:20am EDT for the serviceData element named n1:MySDE, and receive a response with an updated copy of its value. However, after that time, such a query MAY result in a response indicating that no such service data element exists. In other words, the consumer of the SDE is being advised that it can expect to be able to obtain an updated value of this named SDE for 1 day, but after that time the service may no longer have an SDE with the name n1:MySDE.

It is sometimes not sufficient for lifetime information of a SDE to refer only to the complete service data value. Rather, the value of a SDE may contain child elements with different lifetimes than those declared in the parent element. Any XML element contained within a serviceData element value MAY use any combination of the goodFrom, goodUntil, and availableUntil attributes, assuming that the schema for that element allows for these extensibility attributes. Such attributes on child elements override the default values specified on parent elements. There are no constraints on the values of these attributes in the child elements, relative to those specified in the parent elements, except that the ordering constraints between the effective goodFrom, goodUntil and availableUntil values for any element must be maintained.

In the above example, the lifetime attributes carried in the parent element (n1:MySDE) provide default values for all children of that element. For example, the n1:e1 element uses these default values, as described above. However, the n1:e2 element overrides the goodUntil attribute, thus stating that its value (“def”) is only expected to be valid for 10 minutes, instead of 1 hour as is declared in the parent element. Such a situation might arise if a portion of a complex element changes more quickly than other portions of the element. Likewise, the n1:e3 element overrides the availableUntil, thus stating that the n1:e3 element may no longer exist within n1:MySDE after 10 hours. In other words, after 10 hours, a requestor that queries for the value of this serviceData element MAY be returned a n1:MySDE element that does not contain a n1:e3 child element. This example, of course, assumes that the n1:MyType schema allows for n1:e3 to be an optional element, and thus be omitted from n1:MyType.

It is RECOMMENDED that the XML schema for elements that are intended to be used as types in service data declarations allow all elements within their schema to be extended with these lifetime declaration properties, in order to allow for fine-grained lifetime declarations.

If these attributes do not appear on an element, then the goodFrom, goodUntil and availableUntil properties are unknown.

ogsi-wg@gridforum.org
7.4 Interface Naming and Change Management

A critical issue in distributed systems is enabling the upgrade of services over time. This implies in turn that clients need to be able to determine when services have changed their interface and/or implementation. Here, we discuss this issue and some of the OGSA mechanisms, requirements, and recommendations that are used to address it.

7.4.1 The Change Management Problem

The semantics of a particular Grid service instance are defined by the combination of two things:

1. Its interface specification. Syntactically, a Grid service’s interface is defined by its service description, comprising its portTypes, operations, serviceData declarations, messages, and types. Semantically, the interface typically is defined in specification documents such as this one, though it may also be defined through other formal approaches.

2. The implementation of the interface. While expected implementation semantics may be implied from interface specifications, ultimately it is the implementation that truly defines the semantics of any given Grid service instance. Implementation decisions and errors may result in a service having behaviors that are ill-defined and/or at odds with the interface specification. Nonetheless, in such an implementation semantics may come to be relied upon by clients of that service interface, whether by accident or by design.

In order for a client to be able to reliably discover and use a Grid service instance, the client must be able to determine whether it is compatible with both of these two definitions of the service. In other words, does the Grid service support the portType(s) that the client requires? And does the implementation have the semantics that the client requires, such as a particular patch level containing a critical bug fix?

Further, Grid service descriptions will necessarily evolve over time. If a Grid service description is extended in a backward compatible manner, then clients that require the previous definition of the Grid service should be able to use a Grid service that supports the new extended description. Such backward compatible extensions might occur to the interface definition, such as through the addition of a new operation or service data description to the interface, or the addition of optional extensions to existing operations. Or, backward compatible extensions might occur through implementation changes, such as a patch that fixes a bug. For example, a new implementation that corrects an error that previously caused an operation to fail would generally be viewed as being backwards compatible.

However, if a Grid service description is changed in a way that is not backward compatible, a client MUST be able to recognize this as well. Again, this could be the result of incompatible changes to the interface or implementation of a Grid service. A bug fix that “fixes” an “erroneous” behavior that users have learned to take advantage of might not be considered backward compatible.

This discussion points to the need to be able to provide concise descriptions of both the interface and implementation of a Grid service, as well as to make unambiguous compatibility statements about Grid services that support different interfaces or implementations.

7.4.2 Naming Conventions for Grid Service Descriptions

In WSDL, each portType is globally and uniquely named via its qualified name—that is, the combination of the namespace containing the portType definition and the locally unique name of the portType element within that namespace. In OGSI, our concern with change management
leads us to require that all elements of a Grid service description MUST be immutable. The
qname of a Grid service portType, operation, message, serviceData declaration and underlying
type definitions MAY be assumed to refer to one and only one WSDL/XSD specification. If a
change is needed, a new portType MUST be defined with a new qname—that is, defined with a
new local name, and/or in a new namespace.

Note that during development, a Grid Service may change frequently at both the interface,
portType, and implementation levels. Because of the strong immutability requirement above,
developers should choose their release schedules carefully. Once an interface or implementation
is in use outside of the total control of the developer, no further changes are permitted without the
introduction of a new portType.

Changes in the interface that extend the functionality of a Grid Service, without altering its
existing behavior, SHOULD be modeled with portType extension, allowing existing clients to use
the new service without modification. Such interface extensions MAY also be modeled with
totally new portTypes, but this is not recommended.

Changes to the implementation only MUST always be backward compatible, so that existing
clients can continue to function as before. Significant changes (i.e. not backwards compatible) in
the implementation MUST be assigned a new portType, even if the interface has not changed.

7.5 Naming Grid Service Instances: Handles and References

Each Grid service instance is named, globally and for all time, by one or more Grid Service
Handles (GSH). However, a GSH is just a minimal name in the form of a URI, and does not carry
enough information to allow a client to communicate directly with the service instance. Instead, a
GSH must be resolved to a Grid Service Reference (GSR). A GSR contains all information that a
client requires to communicate with the service via one or more network protocol bindings.

Like any URI, a GSH consists of a scheme, followed by a string containing information that is
specific to the scheme. The scheme indicates how one interprets the scheme-specific data to
resolve the GSH into a GSR, within the bounds of the requirements defined below. A client MAY
choose to resolve GSHs itself, or it MAY choose to outsource all resolution, for example, to a
pre-configured service that implements the HandleResolver portType (see Section 10).

The format of the GSR is specific to the binding mechanism used by the client to communicate
with the Grid service instance. For example, if an RMI/IIOP binding is used, the GSR would take
the format of an IOR. If a SOAP binding is used, the GSR would take the form of a properly
annotated WSDL document.

While a GSH is valid for the entire lifetime of the Grid service instance, a GSR may become
invalid, therefore requiring a client to resolve the GSH into a new, valid GSR.

7.5.1 Grid Service Reference (GSR)

Grid service instances are made accessible to (potentially remote) client applications through the
use of a Grid Service Reference (GSR). A GSR is typically a network-wide pointer to a specific
Grid service instance that is hosted in an environment responsible for its execution. A client
application can use a GSR to send requests (represented by the operations defined in the WSDL
portType(s) of the target service) directly to the specific instance at the specified (potentially
network-attached) service endpoint identified by the GSR. In other words, the GSR supports the
programmatic notion of passing Grid service instances "by reference". The GSR contains all of
the information required to access the Grid service instance resident in its hosting environment
over one or more communication protocol bindings. However, a GSR may be localized to a given
client context or hosting environment, and the scope of portability for a GSR is determined by the
binding mechanism(s) it supports.

The encoding of a Grid Service Reference may take many forms in the system. Like any other
operation message part, the actual encoded format of the GSR "on the wire" is specific to the
Web service binding mechanism used by the client to communicate with the Grid service
instance. Below we define a WSDL encoding of a GSR that MAY be used by some bindings, but
the use of any particular encoding is defined in binding specifications, and is therefore outside of
the scope of this specification.

However, it is useful to elaborate further on this point here. For example, if an RMI/IIOP binding
were used, the GSR would be encoded as a CORBA compliant IOR. If a SOAP binding were
used, the GSR may take the form of the WSDL encoding defined below. This "on the wire" form
of the Grid Service Reference is created both in the Grid service hosting environment, when
references are returned as reply parameters of a WSDL defined operation, and by the client
application or its designated execution environment when references are passed as input
parameters of a WSDL defined operation. This "on the wire" form of the Grid Service Reference,
passed as a parameter of a WSDL defined operation request message, SHOULD include all of the
service endpoint binding address information required to communicate with the associated
service instance over any of the communication protocols supported by the designated service
instance, regardless of the Web service binding protocol used to carry the WSDL defined
operation request message.

Any number of Grid Service References to a given Grid service instance MAY exist in the
system. The lifecycle of a GSR MAY be independent of the lifecycle of the associated Grid
service instance. A GSR is valid when the associated Grid service instance exists and can be
accessed through use of the Grid Service Reference, but validity MAY only be detected by the
client attempting to utilize the GSR. A GSR MAY become invalid during the lifetime of the Grid
service instance. Typically this occurs because of changes introduced at the Grid service hosting
environment. These changes MAY include modifications to the Web service binding protocols
supported at the hosting environment, or of course, the destruction of the Grid service instance
itself. Use of an invalid Grid Service Reference by a client SHOULD result in an exception being
detected by or presented to the client.

When a Grid Service Reference is found to be invalid and the designated Grid service instance
exists, a client MAY obtain a new GSR using the Grid Service Handle of the associated Grid
service instance, as defined in Section 7.5.2. For convenience, the Grid Service Handle MAY be
contained within a binding-specific implementation of the Grid Service Reference.

A binding-specific implementation of a Grid Service Reference MAY include an expiration time,
which is a declaration to clients holding that GSR that the GSR SHOULD be valid prior to that
time, and it MAY NOT be valid after the expiration time. After the expiration time, a client MAY
continue to attempt to use the GSR, but SHOULD retrieve a new GSR using the GSH of the Grid
service instance. While an expiration time provides no guarantees, it nonetheless is a useful hint
in that it allows clients to refresh GSRs at convenient times (perhaps simultaneously with other
operations), rather than simply waiting until the GSR becomes invalid, at which time it must
perform the (potentially time-consuming) refresh before it can proceed.

Mere possession of a GSR does not entitle a client to invoke operations on the Grid service. In
other words, a GSR is not a capability. Rather, authorization to invoke operations on a Grid
service instance is an orthogonal issue, to be addressed elsewhere.

The XML schema definition for a GSR in general is as follows.

```xml
... targetNamespace =
```
The definitions of the two attributes are given in section 7.3. These give the expected lifetime of the GSR (not of the service to which it refers). A client SHOULD NOT call a service using a given GSR before its goodFrom time or after its goodUntil time.

7.5.1.1 WSDL Encoding of a GSR

It is RECOMMENDED that a WSDL document that encodes a GSR be the minimal information required to describe fully how to reach the particular Grid service instance. The WSDL GSR must contain a wsdl:definitions child element. The wsdl:definitions MUST contain exactly one wsdl:service element.

The XML definition for a WSDL GSR is as follows.

7.5.2 Grid Service Handle (GSH)

Handles have the following properties:

1. A GSH MUST be a valid URI [RFC 2396].
2. A GSH MUST globally and for all time refer to the same Grid service instance. A GSH MUST NOT ever refer to more than one Grid service instance, whether or not they exist simultaneously. See Section 7.5.2.1 for the definition of “same Grid service instance.”
3. A Grid service instance MUST have at least one GSH.
4. A Grid service instance MAY have multiple GSHs that use different URI schemes.
5. A Grid service instance MAY have multiple GSHs that use the same URI scheme, if it is allowed by that URI scheme. However, the specification for a particular URI scheme MAY restrict this property by only allowing a single GSH within that URI scheme to a Grid service instance.
6. The GridServiceHandles service data element (Section 9.1) of a Grid service instance MUST contain only GSHs that refer to that instance. If two or more GSHs are contained
in a Grid service instance’s GridServiceHandles SDE, then they MUST all refer to that same Grid service instance. The GridServiceHandles SDE MAY contain a subset of all GSHs of the GridService.

7. There MAY be multiple GSRs that refer to the same Grid service instance.

8. Multiple resolutions of the same GSH MAY result in different GSRs. A resolver MAY return different GSRs for the same GSH at different times, and it MAY return different GSRs to different clients that are resolving the same GSH.

9. A GSH MAY be unresolvable to a GSR before, during, or after the existence of the Grid service instance to which the GSH refers. However, the specification for a particular URI scheme MAY define a stronger quality of service for resolution. For example, a particular URI scheme MAY guarantee that resolution during the instance’s lifetime, and to a reliable statement of termination after the instance has terminated.

10. A client’s trust of a service MAY be handled in any way the client chooses, and therefore resolution of a GSH to a GSR MAY be either a trusted or an untrusted operation, depending on, for example, the configuration of the client, or the definition of the resolution protocol. This specification permits, but does not require, a client to trust the resolution of a GSH to a GSR.

The XML definition for a GSH is as follows.

```xml
... targetNamespace = "http://www.gridforum.org/namespaces/2003/OGSI"
<xsd:element name="serviceHandle" type="ogsi:serviceHandleType"/>
<xsd:simpleType name="serviceHandleType">
    <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
```

7.5.2.1 Service Instance Sameness

The interpretation of “same Grid service instance” is dependent upon the semantics of a service’s description. A service instance may be implemented in any way, as long as it obeys the semantics associated with its service description (i.e. the portType that the service instance implements).

For example, the implementation of a service MAY be distributed or replicated across multiple resources, as long as it obeys the semantics associated with its service description. A single GSH would be associated with this service, though that GSH may resolve to different GSRs referring to different resources, based on such factors as resource availability and utilization, locality of a client, client privileges, etc. Some service descriptions may require tight state coherency between the replicated implementations -- for example, the semantics of the service description may 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 GSHs are resolved to GSRs. However, other service descriptions may be defined that allow for looser consistency between the various members of the distributed service implementation.

7.5.3 Locators

A service locator is a structure that contains zero or more GSHs and zero or more GSRs, all of which MUST refer to the same Grid service instance. Locators are used by various operations in this specification that may accept either a GSH or a GSR. Its XML definition is as follows.

```xml
...
targetNamespace =
    "http://www.gridforum.org/namespaces/2003/OGSI"

<xsd:element name="serviceLocator" type="ogsi:serviceLocatorType"/>

<xsd:complexType name="serviceLocatorType">
    <xsd:sequence>
        <xsd:element ref="ogsi:serviceHandle"
                      minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ogsi:serviceReference"
                      minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

7.6 Grid Service Lifecycle

The lifecycle of any Grid service is demarked by the creation and destruction of that service. The actual mechanisms by which a Grid service is created or destroyed are fundamentally a property of the hosting environment, and as such are not defined in this document. There is nonetheless a collection of related portTypes defined in this specification that specify how clients may interact with these lifecycle events in a common manner. As we describe in subsequent sections:

- A client may request the **creation** of a Grid service by invoking the `createService` operation on a `Factory` service. (A service instance that implements a portType that extends the `Factory` portType.)

- A client may request the **destruction** of a Grid service via either client invocation of an `explicit` destruction operation request to the Grid service (see the `Destroy` operation, supported by the `GridService` portType: Section 9) or via a `soft-state` approach, in which (as motivated and described in [Grid Physiology]) a client registers interest in the Grid service for a specific period of time, and if that timeout expires without the service having received re-affirmation of interest from any client to extend the timeout, the service may be automatically destroyed. Periodic re-affirmation can serve to extend the lifetime of a Grid service as long as is necessary (see the `RequestTerminationBefore/After` operations in the `GridService` portType: Section 9).

In addition, a Grid service MAY support notification of lifetime-related events, through the standard notification interfaces defined in Section 11.

A Grid service MAY support soft state lifetime management, in which case a client negotiates an initial service instance lifetime when the Grid service is created through a factory (Section 12), and authorized clients MAY subsequently send `RequestTerminationBefore/After` (“keepalive”) messages to request extensions to the service's lifetime. If the Grid service termination time is reached, the server hosting the service MAY destroy the service, reclaim any resources associated with the service, and remove any knowledge of the service maintained in handle resolvers under its control.

Termination time MAY change non-monotonically. That is, a client MAY request a termination time that is earlier than the current termination time. If the requested termination time is before the current time, then this SHOULD be interpreted as a request for immediate termination.

A Grid service MAY decide at any time to extend its lifetime. A service MAY also terminate itself at any time, for example if resource constraints and priorities dictate that it relinquish its resources.
Termination time is typically represented by an element of the form given by the following non-normative grammar:

```xml
<terminationTime
    after="ogsi:extendedDateTime"
    before="ogsi:extendedDateTime" />
```

The extendedDateTime element is defined in section 7.2. This element describes a service’s current plans for termination. The “after” attribute gives the earliest time at which the service plans to terminate. A value in the past indicates that the service may terminate at any time. The special value “infinity” indicates that the service plans to exist indefinitely. The “before” attribute gives the latest time at which the service plans to terminate. A value in the past indicates that the service is trying to terminate. The special value “infinity” indicates that the service has no plans to terminate. The “after” time MUST be less than or equal to the “before” time.

### 7.7 Common Handling of Operation Faults

This section has been added in DRAFT form in an effort to provide a complete picture of the spec as of Feb 10. This section is still subject to review.

Supporting automatic adaptation to faults, as is required in a Grid setting, requires the ability to return faults (also called exceptions in some programming languages) that are rich with information about the cause of the fault, but are consistent in their content, semantics, and means of delivery. Therefore, this specification defines a fault management framework that is used consistently throughout the OGSI portTypes defined here, and SHOULD be used by other portTypes that build upon OGSI to define complete services.

The basis for all OGSI faults is the GridServiceFaultType XSD type:

```xml
... targetNamespace =
    "http://www.gridforum.org/namespaces/2003/OGSI"

<xsd:complexType name='GridServiceFaultType'>
    <xsd:sequence>
        <xsd:element name='code'
            minOccurs='1' maxOccurs='1'
            type='xsd:string'/>
        <xsd:element name='description'
            minOccurs='1' maxOccurs='1'
            type='xsd:string'/>
        <xsd:element name='originator'
            minOccurs='1' maxOccurs='1'
            type='ogsi:serviceLocator'/>
        <xsd:element name='timestamp'
            minOccurs='1' maxOccurs='1'
            type='xsd:dateTime'/>
        <xsd:element name='cause'
            minOccurs='0' maxOccurs='unbounded'
            type='ogsi:GridServiceFaultType'/>
        <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>
```
All faults from a Grid service SHOULD either use this type directly, or should use extensions of this type.

The description element contains a plain language description of the fault. This is expected to be helpful to the client.

The originator is a serviceLocator of the service raising the fault.

The timestamp the approximate time at which the fault occurred.

The cause is any number of GridServiceFaultType elements that describe the underlying cause(s) that resulted in this fault. This allows for “chaining” of fault information up through a service invocation stack, so that a recipient of the fault can drill down through the causes to understand more detail about the reason for the fault.

The extensibility element, if present, contains information specific to the fault from other XML namespaces.

This specification defines two basic subtypes of GridServiceFaultType.

- StructuralFaultType: A structural fault, or subtype thereof, SHOULD be returned whenever the fault can be characterized as a type error, missing (required) operands, unknown SDEs, or other unsupported capabilities, e.g. unsupported queryExpressionType. These faults are essentially those that arise from an incompatibility between the operation received and the WSDL and service data describing the operation.

- SemanticFaultType: A semantic fault, or subtype thereof, SHOULD be returned when an internal failure occurs within the Grid Service. All instances of these faults SHOULD be described in the GridService's documentation.

The subtypes of GridServiceFaultType have the following structure.

```
<xsd:complexType name='StructuralFaultType'>
  <xsd:complexContent>
    <xsd:extension base='ogsi:GridServiceFaultType'/>
  </xsd:complexContent>
</xsd:complexType>
```

A Grid service operation MAY return a more refined fault (i.e. an XSD extension) in place of a particular fault that is required by a portType specification. For example, if an operation is specified to return a GridServiceAuthorizationType fault under some circumstance, then a particular implementation of that operation MAY return an extension of GridServiceAuthorizationType in its place. A GridService operation SHOULD return the most refined fault possible.

All faults raised by a Grid service operation MUST be listed as fault messages in the service description for that operation. In particular, a full type MUST be included and the WSDL fault name MUST match the type name. All Grid service operations MUST return a GridServiceFault, StructuralFault, and SemanticFault in addition operation specific faults. This allows an implementation to raise more general faults where it is not possible (or desirable) to raise a more refined fault. Note that although not listed explicitly, all operations listed in this specification raise GridServiceFault, StructuralFault, and SemanticFault in addition to operation specific faults.
The parent type of all faults listed in this specification are indicated as part of the fault type's definition.

Tooling environment developers are encouraged to introspect the type hierarchy of GridServiceFaults and provide support for catching refined a fault based on the type of its parent or ancestor.

8 Grid Service Interfaces

A Grid service MUST implement the GridService portType, or a portType that extends the GridService portType. (See Section 9).

This specification defines a collection of common distributed computing patterns that are considered to be fundamental to OGSI. The embodiment of these patterns appears as WSDL portTypes. The collection of portTypes specified in this document is listed in Table 2.

The task for the designer of components within OGSI is to design portTypes that extend a combination of the GridService portType, other optional portTypes defined in this specification, and application or domain specific portTypes.

<table>
<thead>
<tr>
<th>PortType Name</th>
<th>See Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridService</td>
<td>9</td>
<td>encapsulates the root behavior of the component model</td>
</tr>
<tr>
<td>HandleResolver</td>
<td>10</td>
<td>mapping from a GSH to a GSR</td>
</tr>
<tr>
<td>NotificationSource</td>
<td>11.1</td>
<td>allows clients to subscribe to notification messages</td>
</tr>
<tr>
<td>NotificationSubscription</td>
<td>11.2</td>
<td>defines the relationship between a single NotificationSource and NotificationSink pair</td>
</tr>
<tr>
<td>NotificationSink</td>
<td>11.3</td>
<td>defines a single operation for delivering a notification message to the service instance that implements the operation</td>
</tr>
<tr>
<td>Factory</td>
<td>12</td>
<td>standard operation for creation of Grid service instances</td>
</tr>
<tr>
<td>Registration</td>
<td>Error! Reference source not found.</td>
<td>allows clients to register and unregister registry contents</td>
</tr>
</tbody>
</table>

All of these portTypes (except of course for GridService) extend the GridService portType.

In the subsequent sections, we describe the various standard portTypes that are defined by OGSI. All portTypes are defined in the ogsi namespace.

9 The GridService PortType

We start with the GridService portType, which MUST be implemented by all Grid services and thus serves as the base interface definition in OGSI. 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 querying and updating against the serviceData set of the Grid service instance, and managing the termination of the instance.

In Web services interface design, there is a choice to be made between document-centric messaging patterns and remote procedure call (RPC). Using a document-centric approach, the interface designer defines a loosely coupled interaction pattern wherein the API to the service is defined in terms of document exchange; both input and output are XML documents. This approach shifts the complexity of the interaction away from the API level and into the data format of the document exchange itself. This style tends to yield simpler, more flexible APIs. The RPC approach defines a specific, strongly-typed operation signature. This approach tends to produce less flexible API, but is often easier to map onto APIs of existing objects and can have better runtime performance.

Designers of Grid service interfaces also face the document-centric vs. RPC choice, and have attempted to take a middle road. The GridService portType provides several operations with typed parameters, but leaves considerable extensibility options within several of those parameters. Service data is then used to express what specific extensibility elements a particular service instance understands. Grid service designers are free to mix and match the document-centric and RPC approaches in the portTypes that they design to compose with those described here.

9.1 GridService PortType: Service Data Declarations and Elements

The GridService portType includes the following serviceData elements:

- PortType
  - The Qnames of the portType(s) within the service instance’s complete interface definition

```xml
<sd:serviceData name="PortType" type="ogsi:portTypeNameType"
  minOccurs="1" maxOccurs="unbounded"
  mutability="constant"/>
```

- ServiceDataNames
  - The Qnames, one for each service data element contained in this service instance.
    Note: the minOccurs corresponds to the number of serviceDataDescription elements in this GridService portType that are required (that have minOccurs > 0).

```xml
<sd:serviceData name="serviceDataName" type="xsd:QName"
  minOccurs="6" maxOccurs="unbounded"/>
```

- FactoryHandle
  - The Grid Service Handle to the factory that created this Grid service instance, if the instance was not created by a factory, this value MUST be xsi:nil.

```xml
<sd:serviceData name="FactoryHandle" type="ogsi:gridServiceHandleType"
  minOccurs="0" mutability="constant" nillable="true"/>
```

- GridServiceHandle
  - Some of the Grid Service Handles of this Grid service instance. It is possible to have multiple handles, for example one for each handle scheme that might be deployed. There MAY be handles to this service that are not included here.

```xml
<sd:serviceData name="GridServiceHandle" type="ogsi:gridServiceHandleType"
  minOccurs="0" maxOccurs="unbounded"/>
GridServiceReferences

- A set of Grid Service References to this Grid service instance. One service data value element MUST be the WSDL representation of the GSR. Other service data value elements may represent other forms of the GSR.

```
<sd:serviceData name="GridServiceReferences"
    type="ogsi:gridServiceReferenceType"
    minOccurs="0" maxOccurs="unbounded"
    mutability="mutable"/>
```

QueryExpressionType

- A set of Qnames of element declarations. Any conforming element MAY be used by the client as a QueryExpression parameter to the instance’s FindServiceData operation, and implies the query semantics and return values that may result from the query.

```
<sd:serviceData name="QueryExpressionType"
    type="xsd:QName"
    minOccurs="1" maxOccurs="unbounded"
    mutability="static"/>
```

TerminationTime

- The earliest and latest termination times for the grid service (see 7.6).

```
<sd:serviceData name="TerminationTime" type="ogsi:terminationTime"
    minOccurs="1" maxOccurs="1"
    mutability="mutable"/>
```

CurrentTime

- The current time as known to this service. This value can be used to gauge the clock skew of this service relative to a client.

```
<sd:serviceData name="CurrentTime" type="ogsi:currentTime"
    minOccurs="1" maxOccurs="1"
    mutability="mutable"/>
```

UpdateExpressionType

- A set of Qnames of element declarations. Any conforming element MAY be used by the client as the UpdateExpression parameter to the instance’s SetServiceData operation, and implies the update semantics and return values that may result from the update.

```
<sd:serviceData name="UpdateExpressionType"
    type="xsd:QName"
    minOccurs="2" maxOccurs="unbounded"
    mutability="static modifiable=false"/>
```

In addition, the GridService portType defines the following initial set of service data value elements:

```
<sd:staticServiceDataValues>
  <ogsi:QueryExpressionType>
    ogsi:queryByMultipleServiceDataNames
  </ogsi:QueryExpressionType>
  <ogsi:UpdateExpressionType>
```
For an explanation of the QueryExpressionTypes, see section 9.2.1. For an explanation of updateExpressions, see section 9.2.2

9.2 GridService PortType: Operations and Messages

9.2.1 GridService :: FindServiceData

Query the service data.

Input

- QueryExpression: The query to be performed. This is an extensible parameter, which MUST conform to an element declaration denoted by one of the QueryExpressionType SDE values. Note: The service infers what to do based on the tag of the root element of this argument.

Output

- Result: The result of the query. The format of this result is dependent upon the QueryExpression.

Fault(s)

- TBD.

Every Grid service instance MUST support QueryExpressions conforming to queryByServiceDataName as defined in 9.2.1.1. A Grid service instance MAY support other QueryExpressions.

The list of query expression types supported by a Grid service instance is expressed in the instance’s QueryExpressionType service data element. Therefore, a client can discover the query expression types supported by a service instance by performing a FindServiceData request on the instance, using the queryByServiceDataName element with name=”ogsi:QueryExpressionTypes”.

The service data that is available for query by a client MAY be subject to policy restrictions. For example, some service data elements MAY not be available to some clients, and some service data value elements within a SDE MAY not be available to some clients.

9.2.1.1 queryByMultipleServiceDataNames

A queryByMultipleServiceDataNames results in a serviceDataValues element containing the service data elements named in the queryByMultipleServiceDataNames parameter. The name listed in the query MUST be among the serviceDataNames (see 9.1) contained in this service instance.

The non-normative grammar of this type is:

<ogsi:queryByMultipleServiceDataNames names="list of Qname"/>
The FindServiceData operation’s Result output parameter for a queryByMultipleServiceDataNames query MUST be a serviceDataValues element containing the service data elements listed in the queryByMultipleServiceDataNames.

Clients MAY use the serviceData and serviceDataValues lifetime attributes (see Section 7.3) to obtain an understanding of the validity of the values returned.

### 9.2.2 GridService :: SetServiceData

This operation allows the modification of a service data element’s values if its service data declaration specifies modifiable="true". Changing a service data element that is modifiable implies changing the corresponding state in the underlying service. If no service data elements have a modifiable="true" attribute then SetServiceData is essentially disabled.

**Input**
- **UpdateExpression**: The update to be performed. This is an extensible parameter, which MUST conform to an element declaration denoted by one of the UpdateExpressionTypes. Note: The service infers what to do based on the tag of the root element of this argument

**Output**
- **Result**: The result of the update. The format of this result is dependent upon the UpdateExpression.

**Fault(s)**
- **OperationFailed**: indicating that the value could not be modified.
- **ElementNotModifiable**: indicating that one or more service data elements were not in the serviceDataValues of modifiable elements.
- **IncompatibleValueType**: indicating that the value is not XML-valid with respect to its XML datatype.
- **IncorrectValue**: indicating that the value is XML-valid but not acceptable for other reasons.
- **ExpressionNotSupported**: indicating that the given UpdateExpression was not supported by the service.
- **CardinalityConstraintsViolation**: indicating that operation requested would violate the minOccurs, maxOccurs cardinality.
- **MutabilityViolation**: indicating that the UpdateExpression was not consistent with the mutability of the SDE.

(Note: Some of these faults are generic in nature and all GS operations may be susceptible to and should probably support)

Every Grid service instance MUST support UpdateExpressions conforming to setByServiceDataName and deleteByServiceDataName as defined in 9.2.2.2. A Grid service instance MAY support other UpdateExpressions.

The list of update expression types supported by a Grid service instance is expressed in the instance’s UpdateExpressionType service data element. Therefore, a client can discover the update expression types supported by a service instance by performing a FindServiceData request on the instance, using the queryByMultipleServiceDataNames element with name="ogsi:UpdateExpressionType".
The service data that is available for setting by a client MAY be subject to policy restrictions. For example, some service data elements MAY not be available to some clients, and some service data value elements within a SDE MAY not be settable to some clients.

9.2.2.1 setByServiceDataName

A setByServiceDataName results in an update of the value in the service data element specified in the setByServiceDataName expression. The name specified in the expression MUST be one of the serviceDataNames (see 9.1) contained in this service instance with an attribute modifiable="true". It is the service’s responsibility to ensure that the update to the SDE is successfully reflected in the underlying service’s state.

A setByServiceDataName MUST adhere to the mutability attribute of the SDE as specified in section 6.2.4. If the mutability value is “constant” or “static”, then setByServiceDataName is not allowed. If the mutability value is “extend”, then setByServiceDataName MUST append the new value(s) to the existing value(s). If the mutability value is “mutable”, then setByServiceDataName MUST replace the existing element value(s) with those passed in as parameters to the SetServiceData operation. The setByServiceDataName MUST result in the serviceData element values adhering to the minOccurs/maxOccurs rules defined in the serviceData declaration.

The non-normative grammar of this type is:

```xml
<ogsi:setByServiceDataName>
  <someServiceDataNameTag>
    new SDE value(s)
  </someServiceDataNameTag>*
</ogsi:setByServiceDataName>
```

The SetServiceData operation’s Result output parameter for a setByServiceDataName invocation is not required. Absence of a fault being thrown indicates success.

Note that the client SHOULD NOT assume that the values in the service’s serviceDataValues are in any way coherent with each other.

9.2.2.2 deleteByServiceDataName

A deleteByServiceDataName results in a deletion of the values in the service data element specified in the deleteByServiceDataName expression. The name specified in the expression MUST be one of the serviceDataNames (see 9.1) contained in this service instance with an attribute modifiable="true". It is the service’s responsibility to ensure that the update to the SDE is successfully reflected in the underlying service’s state.

A deleteByServiceDataName MUST adhere to the mutability attribute of the SDE as specified in section 6.2.4. If the mutability value is “static”, “constant” or “extend”, then deleteByServiceDataName is not allowed. If the mutability value is “mutable”, then deleteByServiceDataName will delete all SDE values with the given SDE name. A deleteByServiceDataName will fail if it would cause the serviceData element values to be in violation of the minOccurs/maxOccurs rules in the serviceData declaration.

The non-normative grammar of this type is:

```xml
<ogsi:deleteByServiceDataName name="QName" />
```

The SetServiceData operation’s Result output parameter for a deleteByServiceDataName invocation is not required. Absence of a fault being thrown indicates success.
Note that the client SHOULD NOT assume that the values in the service’s serviceDataValues are in any way coherent with each other. Clients also SHOULD NOT assume that the update of a single complex service data element was updated with any coherency guarantee.

9.2.2.3 setByMultipleServiceDataNames (optional)
A setByMultipleServiceDataNames results in the update of the SDE values in the serviceDataValues containing the service data elements listed in the setByMultipleServiceDataNames expression. The names listed in the expression MUST be among the serviceDataNames (see 9.1) contained in this service instance with an attribute of modifiable="true". There is no guarantee that the update is executed in any particular sequence. It is the service’s responsibility to ensure that the SDE values are successfully updated in the underlying service’s state. The service SHOULD update as many SDEs successfully as it can and return those that failed. The service MAY return after the first failure and not continue.

A setMultipleByServiceDataNames MUST adhere to the mutability attribute of the SDEs as specified in section 6.2.4. If the mutability value is “static” or “constant”, then setByMultipleServiceDataNames is not allowed. If the mutability value is “extend”, then setByMultipleServiceDataNames MUST append to the new elements to the SDE’s existing values. If the mutability value is “mutable”, then setByMultipleServiceDataNames MUST replace the existing SDE values with the ones that are passed in. The setByMultipleServiceDataNames MUST, for each SDE named in the expression, result in the serviceData element values adhering to the minOccurs/maxOccurs rules defined in the serviceData declaration.

The non-normative grammar of this type is:

```xml
<ogsi:setByMultipleServiceDataNames>
  <someServiceDataNameTag>
    new SDE value(s)
  </someServiceDataNameTag>*
  <someOtherServiceDataNameTag>*
    new SDE value(s)
  </someOtherServiceDataNameTag>*
</ogsi:setByMultipleServiceDataNames>
```

The SetServiceData operation’s Result output parameter for a setByMultipleServiceDataNames invocation MUST be a serviceDataValues element containing the service data elements listed in the setByMultipleServiceDataNames expression that failed to be replaced. An empty value set indicates success.

Note that the client SHOULD NOT assume that the values in the service’s serviceDataValues are in any way coherent with each other. Clients also SHOULD NOT assume that the replacing of any service data elements are carried out with any coherency guarantee.

9.2.2.4 deleteByMultipleServiceDataNames (optional)
A deleteByMultipleServiceDataNames results in the deletion of all the SDE values in the containing the service data elements listed in the deleteByMultipleServiceDataNames expression. The names listed in the expression MUST be among the serviceDataNames (see 9.1) contained in this service instance with an attribute of modifiable="true". There is no guarantee that the update is executed in any particular sequence. It is the service’s responsibility to ensure that the SDEs are successfully deleted in the underlying service’s state. The service SHOULD delete as many
SDEs successfully as it can and return those that failed. The service MAY return after the first failure and not continue.

A deleteByMultipleServiceDataNames MUST adhere to the mutability attribute of the SDE as specified in section 6.2.4. If the mutability value is “static”, “constant” or “extend”, then deleteByMultipleServiceDataNames is not allowed. If the mutability value is “mutable”, then deleteByMultipleServiceDataNames will delete all elements with the SDE names. The deleteByMultipleServiceDataNames MUST, for each SDE named in the expression, result in the serviceData element values adhering to the minOccurs/maxOccurs rules defined in the serviceData declaration.

The non-normative grammar of this type is:

\[
<\text{ogsi:deleteByMultipleServiceDataNames} \text{ names="list of QName"/}> 
\]

The SetServiceData operation’s Result output parameter for a deleteByMultipleServiceDataNames invocation MUST be a serviceDataValues element containing the service data elements listed in the deleteByMultipleServiceDataNames expression that failed to be deleted. An empty value set indicates success.

Note that the client SHOULD NOT assume that the values in the service’s serviceDataValues are in any way coherent with each other or with the actual state of the service instance before they are updated. Clients also SHOULD NOT assume that the deletions of any service data elements are carried out with any coherency guarantee.

9.2.3 GridService :: RequestTerminationAfter

Request that the termination time of this service be changed. The request specifies the earliest desired termination time. Upon receipt of the request, the service MAY adjust its termination time, if necessary, based on its own polices and the requested time. Upon receipt of the response, the client SHOULD discard any responses that have arrived out of order, based on the CurrentTimestamp in the response.

Input:

- \textit{TerminationTime}: The earliest termination time of the Grid service that is acceptable to the client. Its format is an extendedDateTime element as defined in section 7.2. A value in the past indicates that the client no longer cares about the earliest termination time. The special value “infinity” means that the client requests that the service continue to exist indefinitely.

Output:

- \textit{CurrentTimestamp}: The time at which the Grid service handled the request.
- \textit{CurrentTerminationTime}: A terminationTime element (as defined in section 7.6) that gives the service’s currently planned termination time.

Fault(s):

9.2.4 GridService :: RequestTerminationBefore

Request that the termination time of this service be changed. The request specifies the latest desired termination time. Upon receipt of the request, MAY adjust its termination time, if necessary, based on its own polices and the requested time. Upon receipt of the response, the client SHOULD discard any responses that have arrived out of order, based on the timestamp in the response.
Input:
- **TerminationTime**: The latest termination time of the Grid service that is acceptable to the client. Its format is an extendedDateTime element as defined in section 7.2. A time in the past indicates a desire that the service terminate as soon as possible. The special value "infinity" indicates that the client no longer cares about a maximum termination time.

Output:
- **CurrentTimestamp**: The time at which the Grid service handled the request.
- **CurrentTerminationTime**: A terminationTime element (as defined in section 7.6) that gives the service's currently planned termination time.

Fault(s):
- None

9.2.5 GridService :: Destroy
Explicitly request destruction of this service. Upon receipt of an explicit destruction request, a Grid service MUST either initiate its own destruction and return a response acknowledging the receipt of the destroy message; or ignore the request and return a fault message indicating failure. Once destruction of the Grid service is initiated, the service SHOULD NOT respond to further requests. Following a successful Destroy() call, the client SHOULD NOT rely on the existence of the service.

Input:
- None

Output:
- None, indicting that the destroy has been initiated. Some services may strengthen this to mean that the service destruction is effectively complete.

Fault(s):
- **ServiceNotDestroyed**, indicating that the service chose not to initiate self-destruction.

10 The HandleResolver PortType
A handle resolver is a Grid service instance that implements the HandleResolver portType.

Issue 17: The authoring team is divided on the recommendations regarding the role of the resolver protocol vs the use of HandleResolver. Should there be language such as: “Clients SHOULD use one or more HandleResolver services to resolve GSHs to GSRs. The handle resolver protocols discussed in this section SHOULD NOT normally be used by clients, but SHOULD be used by HandleResolver services only.”

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 which HandleResolver service instance a client should use to resolve 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 resolution requests to a HandleResolver service.
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 ServiceGroup 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.

10.1 HandleResolver PortType: Service Data Descriptions

The HandleResolver portType includes the following serviceData elements:

- HandleResolverScheme
  - A set of URIs that correspond to the handleResolver schemes that the HandleResolver implements.

```
<sd:serviceData name="HandleResolverScheme" type="xsd:anyURI"
  minOccurs="1" maxOccurs="unbounded"
  mutability="mutable"/>
```

10.2 HandleResolver PortType: Operations and Messages

10.2.1 HandleResolver :: FindByHandle

Returns a serviceLocator, which contains one or more Grid Service References for a Grid Service Handle.

Input

- **Handle**: A Grid Service Handle.
- **GSRSet**: (optional) a set of one or more GSRs that the client already possesses that are not satisfactory for some reason. This is a hint from the client that these existing references should not be returned in response to this message.

Output

- **Locator**: A service locator containing one or more Grid Service References, and zero or more alternate Grid Service Handles to the same Grid service instance.

Fault(s)

- **InvalidHandle**, indicating that the handle violates the syntax of its URI scheme.
- **NoAdditionalReferencesAvailable**, indicating that the resolver service cannot return a GSR that is not already contained in the GSRSet input parameter.
- **NoReferencesAvailable**, indicating that the resolver service is unable to return a GSR for the input handle, irregardless of the GSRSet input parameter.
- **Redirection**, indicating an alternate handle resolver to which the client MAY direct the request. The GSH of the alternate handle resolver is returned with the fault.
- **NoSuchService**, indicating that either there was never a service with this handle, or the service with this handle has terminated. This fault MAY only be applicable to some URI schemes.
- **NoSuchServiceStarted**, indicating that there was never a service with this handle.
- **ServiceHasTerminated**, indicating that the service with this handle has terminated. This fault MAY only be applicable to some URI schemes.
- **TemporarilyUnavailable**, indicating that the handle refers to a valid service, but that it cannot be resolved to a valid reference at this time, though it may be resolvable later. This fault optionally returns a time at which the service MAY be available. This fault MAY only be applicable to some URI schemes.

### 11 Notification

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 expression 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 The NotificationSource PortType

The NotificationSource portType allows clients to subscribe to notification messages from the Grid service instance that implements this portType.

11.1.1 NotificationSource PortType: Service Data Descriptions and Elements

The NotificationSource portType includes the following serviceData elements:

- NotifiableServiceDataName
  - A set of QNames of service data elements to which a requestor MAY subscribe for notification of changes.

```xml
<sd:serviceData name="NotifiableServiceDataName"
    type="xsd:QName"
    minOccurs="0" maxOccurs="unbounded"
    mutability="mutable"/>
```

- SubscriptionExpressionType
  - A set of Qnames of element declarations. Any conforming element MAY be used by the requestor as a SubscriptionExpression parameter to a Subscribe operation.

```xml
<sd:serviceData name="SubscriptionExpressionType"
    type="xsd:QName"
    minOccurs="1" maxOccurs="unbounded"
    mutability="static"/>
```

The NotificationSource portType also includes the following initial service data value elements:

```xml
<sd:staticServiceDataValues>
  <ogsi:SubscriptionExpressionType>
    <ogsi:subscribeByServiceDataName/>
  </ogsi:SubscriptionExpressionType>
</sd:staticServiceDataValues>
```

11.1.2 NotificationSource PortType: Operations and Messages

11.1.2.1 NotificationSource :: Subscribe

Subscribe to be notified of subsequent changes to the target instance’s service data. This operation creates a Grid service subscription instance, which MAY subsequently be used to manage the lifetime and discovery properties of the subscription.

Input:

- **SubscriptionExpression**: The subscription to be performed. This is an extensible parameter, which MUST conform to an element declaration referred to by one of the QName SDE values in the SubscriptionExpressionType SDE.

- **Sink**: The serviceLocator of the notification sink to which messages will be delivered. This locator MAY be to some other service than the one that is issuing this subscription request, thus allowing for third-party subscriptions.

- **SpecializedNotificationDeliveryOperation (optional)**: The name of the operation, and the QName of the portType in which that operation is defined, to be used by the notification.
source when delivering messages to the notification sink. The operation signature MUST be the same as, or a specialization of, the NotificationSink::DeliverNotification operation. If this parameter is not specified, then it defaults to the “DeliverNotification” operation name that is defined in the ogsi:NotificationSink portType.

- **ExpirationTime**: The initial time at which this subscription instance should terminate, and thus notification delivery to this sink be halted. Normal GridService lifetime management operations MAY be used on the subscription instance to change its lifetime.

**Output:**

- **SubscriptionInstanceLocator**: A serviceLocator to the subscription instance that was created to manage this subscription. This subscription instance MUST implement the NotificationSubscription portType.
- **CurrentTimestamp**: The time at which the NotificationSubscription was created, represented as an ogsi:extendedDateTime element.
- **CurrentTerminationTime**: The NotificationSubscription’s currently planned termination time, represented as a terminationTime element as defined in section 7.6.

**Fault(s):**

Every Grid service instance that implements the NotificationSource portType MUST support a SubscriptionExpressionType of subscribeByServiceDataName as defined in 11.1.2.1.1. A Grid service instance MAY support other SubscriptionExpressionTypes.

The list of subscription expression types supported by a Grid service instance is expressed in the instance’s SubscriptionExpressionType service data element. Therefore, a client can discover the subscription expression types supported by a service instance by performing a FindServiceData request on the instance, using a queryByMultipleServiceDataNames element, which contains the name “ogsi:SubscriptionExpressionType”.

### 11.1.2.1.1 subscribeByServiceDataName

A subscribeByServiceDataName results in notification messages being sent whenever the named service data element changes.

The non-normative grammar of this type is:

```xml
<ogsi:subscribeByServiceDataName
  name="xsd:qname"
  minInterval="xsd:duration"?
  maxInterval=("xsd:duration"|"unbounded")?
/>
```

The minInterval property specifies the minimum interval between notification messages, expressed in xsd:duration. If this property is not specified, then the notification source MAY choose this value. A notification source MAY also reject a subscription request if it cannot satisfy the minimum interval requested.

The maxInterval property specifies the maximum interval between notification messages, expressed in xsd:duration. If this interval elapses without a change to the named service data element’s value, then the source MUST resend the same value. When the value is “unbounded” the source need never resend a service data value if it does not change. If this property is not specified, then the notification source MAY choose this value.

ogsi-wg@gridforum.org
For a subscribeByServiceDataName subscription, the type of the notification message sent from the notification source to the notification sink MUST be a serviceDataValues element containing the SDE values for the serviceData element corresponding to the requested serviceDataName.

11.2 The NotificationSubscription PortType

A subscription for notification causes the creation of a Grid service subscription instance, which MUST implement the NotificationSubscription portType. This instance MAY be used by clients to manage the lifetime of the subscription, and discover properties of the subscription.

11.2.1 NotificationSubscription PortType: Service Data Descriptions

The NotificationSubscription portType includes the following serviceData elements:

- SubscriptionExpression
  - The current subscription expression managed by this subscription instance.

  ```xml
  <sd:serviceData name="SubscriptionExpression"
  type="xsd:Any"
  minOccurs="1" maxOccurs="1"
  mutability="constant"/>
  ```

- SinkLocator
  - The Grid Service Locator of the Notification sink to which this subscription is delivering messages.

  ```xml
  <sd:serviceData name="SinkLocator"
  type="ogsi:serviceLocator"
  minOccurs="1" maxOccurs="1"
  mutability="constant"/>
  ```

11.2.2 NotificationSubscription PortType: Operations and Messages

None.

11.3 The NotificationSink PortType

A notification sink portType defines a single operation for delivering a notification message to the service instance that implements the operation. Note: unlike the other portTypes described in this specification, the Web service implementing the NotificationSink portType is not required to be a Grid service (i.e. the Web service is not required to also implement the GridService portType).

11.3.1 NotificationSink PortType: Service Data Descriptions

None.

11.3.2 NotificationSink PortType: Operations and Messages

11.3.2.1 NotificationSink :: DeliverNotification

Deliver message to this service.

Input:

- Message: An XML element containing the notification message. The content of the message is dependent upon the notification subscription.
Output:
- The service does not reply to this request.

Fault(s):

12 The Factory PortType

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. OGSI defines 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 OGSI, a factory is a Grid service that MUST implement the Factory portType, which provides a standard operation for creation of Grid service instances. A factory MAY also implement other portTypes, such as:

- Registration (Section 12.1 Factory PortType: Service Data Descriptions), 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 10). The method by which this registration is accomplished is specific to the hosting environment, and is therefore outside the scope of this specification.

12.1 Factory PortType: Service Data Descriptions

The Factory portType includes the following serviceData elements:

- CreatesPortType
  - A set of QNames to most derived portTypes of service instances that can be created by this Factory. Issue 93: Make this SDE a parallel construct to PortType SDE in GridService.

```xml
<sd:serviceData name="CreatesPortType" 
    type="xsd:QName"
    minOccurs="1" maxOccurs="unbounded"
    mutability="append"/>
```

- CreatesInputType
  - A set of QNames of XML elements supported by this Factory for the ServiceParameters argument of the CreateService operation.

```xml
<sd:serviceData name="CreatesInputType" 
    type="xsd:QName"
    minOccurs="1" maxOccurs="unbounded"
    mutability="append"/>
```

12.2 Factory PortType: Operations and Messages

12.2.1 Factory :: CreateService

Create a new Grid service instance. Note that to support soft state lifetime management (Section 7.6), a client may specify an initial termination time, within a window of earliest and latest
acceptable initial termination times. The factory selects an initial termination time within this window, and returns this to the client as part of its response to the creation request. Additionally, the factory returns the maximum lifetime extension that clients can subsequently request of this new Grid service instance. Alternatively, the Grid service creation request may fail if the requested termination time is not acceptable to the factory.

Input

- **TerminationTime** (optional): The earliest and latest initial termination times of the Grid service instance that are acceptable to the client. This is a terminationTime element as defined in section 7.6.
- **ServiceParameters** (optional): An XML document that is specific to the factory and the services that it creates.

Output

- **ServiceLocator**: A serviceLocator (see section 7.5.3) to the newly created Grid service instance.
- **CurrentTimestamp**: The time at which the Grid service was created, represented as an ogsi:extendedDateTime element.
- **CurrentTerminationTime**: The Grid service’s currently planned termination time, represented as a terminationTime element as defined in section 7.6.
- **ExtensibilityOutput** (optional): An XML extensibility element that is specific to the factory and the services that it creates.

Fault(s):

**Issue 19**: Can we reduce the number of output parameters in Factory::CreateService by moving many of them into service data of the created instance?

## 13 ServiceGroup

At the teleconference on Feb 05, a small subgroup agreed to put the current draft of the ServiceGroup section (and the faults section) into the main document in an effort to provide a fairly complete picture of where we have arrived at as of Feb 10, 2003.

A ServiceGroup is a Grid service that maintains information about a group of other Grid services. These services may be members of a group for a specific reason, such as being part of a federated service, or they may have no specific relationship, such as the services contained in an index or registry operated for discovery purposes. A classical Grid services registry could be defined by a portType that extends the base behavior described by ServiceGroup.

Two portTypes provide the interface to ServiceGroups: ServiceGroupEntry and ServiceGroup.

### 13.1 The ServiceGroupEntry PortType

This portType defines the interface of the individual entries in a ServiceGroup. Each entry refers (points) to a Grid Service that is a member of the ServiceGroup. The GSH of a ServiceGroupEntry serves as the unique key for that entry. This key allows multiple references to the same service to be included in a ServiceGroup as separate entries, for example to record multiple properties of the service. The service data of a ServiceGroupEntry includes a serviceLocator referring to the member Grid service (MemberServiceLocator) and information about that service (Content).
13.1.1 ServiceGroupEntry: Service Data Declarations

- MemberServiceLocator
  - Contains a serviceLocator for the member Grid service to which this entry pertains.

```
<sd:serviceData
  name="MemberServiceLocator"
  type="gsdl:serviceLocatorType"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant"
  modifiable="false">
</sd:serviceData>
```

- Content
  - An XML element advertising some information about the member service. The type of the Content conforms to one of the elements in the ContentModelType SDE of the ServiceGroup portType containing the ServiceGroupEntry.

```
<sd:serviceData
  name="Content"
  type="##any" <!-- See ContentModelType SDE of ServiceGroup. -->
  minOccurs="1"
  maxOccurs="1"
  mutability="mutable"
  modifiable="true">
</sd:serviceData>
```

13.1.2 ServiceGroupEntry: Operations

The ServiceGroupEntry portType defines no operations. The operations inherited from the GridService portType SHOULD be used to destroy the entry (i.e. remove the service from the ServiceGroup) and to query or update the SDEs of the entry.

13.2 The ServiceGroup PortType

The ServiceGroup contains a set of entries, where each entry is a Grid service implementing the ServiceGroupEntry portType (see 13.1). There is a ServiceGroupEntry for each service in the group (i.e. group members). The ServiceGroupEntry contains a serviceLocator for the member service and information (Content) about that service.

The following properties hold for ServiceGroups, ServiceGroupEntries, and the member GridServices of the ServiceGroup.

- Clients MAY add and remove services to/from a ServiceGroup through operations defined on the ServiceGroup portType.
- Clients MAY, using the FindServiceData operation, query the Service Data (MemberServiceLocator and Content) of the ServiceGroup.
- A GridService MAY be a member of several ServiceGroups.
- A GridService MAY be referenced by more than one entry in a service group (depending on the UniquenessRule SDE (see Section 13.2.1).
- Member Grid services of a ServiceGroup MAY implement different portTypes.
- The Content MAY be homogeneous or heterogeneous, i.e. the Content MAY be the same type for all entries in the ServiceGroup or a given ServiceGroup MAY include several different Content types.
- A ServiceGroupEntry MAY be removed from a ServiceGroup by invoking the destroy operation on the ServiceGroupEntry.
- A ServiceGroupEntry MAY be destroyed by removing it from the ServiceGroup.
- Once a ServiceGroup is destroyed the client has no responsibility for the destruction of the ServiceGroupEntries.
- Once a ServiceGroup is destroyed the client can make no assumptions about the existence of the ServiceGroupEntries or the validity of their contents (e.g. lifetime properties).
- A ServiceGroupEntry can only belong to one ServiceGroup.
- If a GridService, referenced by a ServiceGroupEntry, terminates the ServiceGroupEntry need not reflect this.

13.2.1 ServiceGroup: Service Data Descriptions

- ContentArgumentType
  - The types accepted by the add operation for the Content argument. These types MAY be different than the types listed in the ContentModelType. The add operation converts the value in the Content argument into one of the types contained in the ContentModelType.

```
<sd:serviceData
    name="ContentArgumentType"
    type="xsd:QName"
    minOccurs="1"
    maxOccurs="unbounded"
    mutability="constant"
    modifiable="false"/>
```

- ContentModelType
  - Contains a list of types supported by the ServiceGroup for the Content SDE of the ServiceGroupEntries. As the add operation creates (or updates) a ServiceGroupEntry, the Content argument is converted into a type consistent with one of types contained in the ContentModelType SDE.

```
<sd:serviceData
    name="ContentModelType"
    type="xsd:QName"
    minOccurs="1"
    maxOccurs="unbounded"
    mutability="constant"
    modifiable="false"/>
```

- ServiceGroupEntryType
The QName of the Grid service (typically ServiceGroupEntry or an extension) created by the add operation.

```xml
<sd:serviceData
    name="ServiceGroupEntryType"
    type="xsd:QName"
    minOccurs="1"
    maxOccurs="1"
    mutability="constant"
    modifiable="false">
</sd:serviceData>
```

- **UniquenessRule**
  - The rule followed by the add operation when adding a Grid service to a ServiceGroup. The UniquenessRule is used to compare the serviceLocator argument to the serviceLocators of all entries in the ServiceGroup to decide whether the add operation will create a new ServiceGroupEntry or replace the contents of an existing entry. The absence of a UniquenessRule is interpreted as none, i.e. a new ServiceGroupEntry is always created. There are no specific UniquenessRules defined in this version of the specification.

```xml
<sd:serviceData
    name="UniquenessRule"
    type="xsd:QName"
    minOccurs="0"
    maxOccurs="1"
    mutability="constant"
    modifiable="false">
</sd:serviceData>
```

- **Entry**
  - The structure of the ServiceGroup as Service Data. There is one element for each ServiceGroupEntry in the ServiceGroup. Each SDE value is a triple consisting of the serviceLocator of the ServiceGroupEntry itself, the serviceLocator of the member GridService referenced by this entry, and the Content SDE of the ServiceGroupEntry. The Content in the Entry SDE and that in the Content SDE of the corresponding ServiceGroupEntry SHOULD/MUST be coherent.

```xml
<sd:serviceData
    name="Entry"
    type="gsdl:EntryType"
    minOccurs="0"
    maxOccurs="unbounded"
    mutability="mutable"
    modifiable="false">
</sd:serviceData>
```

```
… targetNamespace =
    "http://www.gridforum.org/namespaces/2003/OGSI"

<xsd:complexType name="EntryType">
    <xsd:sequence>
        <ServiceGroupEntryLocator type="ogsi:serviceLocator"
```
• **MatchExpressionType**
  
  o Those expression types that are supported by the remove operation. All match expressions MUST be boolean valued functions which take a value of type "ogsi:EntryType" as an argument.

```xml
<sd:serviceData
  name="MatchExpressionType"
  type="xsd:QName"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="static"
  modifiable="false">
</sd:serviceData>
```

The only MatchExpressionType defined, in this version of the specification, tests the textual equality of the ServiceGroupEntryLocator ?in each entry contained by the serviceGroup? to that included in the match expression argument. Note that a successful remove, with this match expression, is equivalent to performing destroy on the referenced ServiceGroupEntry.

```xml
<sd:staticServiceDataValues>
  <ogsi:matchByLocatorEquivalence >
    <ogsi:serviceLocator>*</ogsi:serviceLocator>
  </ogsi:matchByLocatorEquivalence>
</sd:staticServiceDataValues>
```

### 13.2.2 ServiceGroup: Operations and Messages

#### 13.2.2.1 ServiceGroup :: Add

The add operation creates a ServiceGroupEntry and adds it to the ServiceGroup based on the add operation's arguments and the UniquenessRule specified in the ServiceGroup's service data. The QName of the sub-type of ServiceGroupEntry created by the add operation is the value of the ServiceGroupEntryType SDE.

**Input:**

- **serviceLocator**: This is a serviceLocator for the member Grid service to be included in the ServiceGroup. It is compared, according to the UniquenessRule SDE, to the serviceLocators of other members of the ServiceGroup. If the argument serviceLocator matches that of an existing member, the Contents SDE and lifetime attributes of the existing ServiceGroupEntry are updated according to the Content and Lifetime arguments (see the UniquenessRule SDE). Otherwise a new ServiceGroupEntry is created, based on the add operation's arguments, and added to the ServiceGroup.

- **Content**: This argument's type MUST conform to one of the values in the ContentArgumentType SDE. The Content is processed, according to service specific
semantics, into an element consistent with one of the entries in the ContentModelType SDE. The resulting Content becomes the ServiceGroupEntry's Content SDE.

- **Lifetime**: The Lifetime argument is used to establish the initial (or updated) lifetime attributes of the ServiceGroupEntry created (or updated) by the add operation. This value is used for both the earliest and latest termination time acceptable to the client, similar to the CreateService operation in the Factory portType.

**Output**:

- **ServiceLocator**: A serviceLocator to the newly created ServiceGroupEntry or to the existing entry that matched the input serviceLocator according to the UniquenessRule.

- **CurrentTimestamp**: The time at which the ServiceGroupEntry was created, represented as an xsd:DateTime element. Note that in the case of an update, this time will be that at which the existing entry was created, rather than the time at which the update was performed.

- **CurrentTerminationTime**: The ServiceGroupEntry's currently planned termination time, represented as a terminationTime element.

**Faults**:

- **LocatorMalformed, ContentArgMalformed, LifetimeMalformed**: Structural faults in the arguments.

- **ContentCreationFailed**: Semantic fault. The ServiceGroup was unable to create a valid Content element (as defined by the ContentModelType SDE) from the provided ContentArg.

13.2.2.2 **ServiceGroup :: Remove**

**Input**:

- **MatchExpression**: The MatchExpression MUST conform to one of the entries in the MatchExpressionTypes SDE. The MatchExpression is evaluated against all entries in the ServiceGroup and for all that match, the entry is removed from the ServiceGroup. Note that this is equivalent to performing a destroy operation on the matching ServiceGroupEntries. Also, that this operation has no effect on the member Grid services referred to by the entries.

**Output**:

- None, except acknowledgment of the operation completion.

**Faults**:

- **MatchExpressionMalformed**: Structural fault in the argument.

- **MatchFailed**: Semantic fault. No entry matched the MatchExpression.

- **RemoveFailed**: Semantic fault. A match was found, but the remove failed for other reasons.

14 **Security Considerations**

This specification defines the abstract interaction between a Grid service and clients of that service. While it is assumed that such interactions must be secured, the details of security are out of the scope of this specification.
of scope of this specification. Instead, security should be addressed in related specifications that
defined how the abstract interactions are bound to specific communication protocols, and to
specific programming environments.

15 Change Log


- Improved introduction to Section 4, “The Grid Service, and reordered the subsections to
  make it flow better.
- Added Section 7.1, “Service Description and Service Instance”, containing an
  explanation of service description vs service instance.
- Added/rewrote “Service Data” section (7.2) including: cleaned up serviceData container;
moved lifetime declarations out to an extensibility element that can be included on any
XML element; introduced schema to be able include service data declarations into the
WSDL service description.
- Changed tables containing service data declarations to use correct XML elements that
  conform to the new serviceDataDescription element
- Moved description of instanceOf to be part of the WSDL GSR description, since it is a
  sub-element of the WSDL service element, which is part of the WSDL GSR.
- Removed old Section 5, “How portType Definitions are Presented”. This was subsumed
  by the rewrite of Section 4, including the new service data specification.
- Removed all primary key material, including old Section 10, and references to it from the
  Factory discussion.
- Simplified the schema for serviceType.
- Added Section 11, Change Log.

15.2 Draft 2 (6/13/2002) \rightarrow Draft 3 (07/17/2002)

- Changed draft to assume new features in WSDL v1.2 draft 1, including serviceType and
  an open extensibility model.
- Added serviceType reuse/extension.
- Modified notion of Handle to be a URI, reflected changes in GSR and HandleResolver
  (previously called HandleMap) discussion. Introduced resolver protocols for the http and
  https GSH schemes.
- Substantially changed “Service Data” section (4.3), primarily to cleanup and plug holes
  in service data descriptions and elements, particularly around naming and typing.
  Changed various portType descriptions to reflect this change to service data.
- Added section “Modeling Time in OGSA”
- Overhauled the notification section, to completely integrate with service data, and to
  provide a “push model” that parallels the FindServiceData “pull model”.
- Renamed Registry portType to Registration, and did some cleanup on the section.
- Introduced ogsi:serviceLocator, which is an XML schema type that can be either a GSH
  or GSR. Changed various GSH and GSR argument to use this type.
• Renamed “Terminology and Abbreviations” section to “Notational Conventions”. In this section, added a table of namespace prefixes used in throughout the document, and cleaned up the rest of the section.

• Added inline “Issue” that need to be resolved, with numbers that refer to the GGF OGSI working group bugzilla database.


• Reformatted for GGF document compliance.

• Updated and reformatted references.

• Added “Security Considerations” section.

• TBD: “Action items” from minutes.

• Changed (in Error! Reference source not found.) the “type” attribute of SDD to be “element”, and reference an XSD element declaration, not a type definition, resolving issue 32. The defaults for minOccurs and maxOccurs were changed to 1 to be consistent with XML schema.

• Added text to Error! Reference source not found. about whether minOccurs>0 makes sense in light of authorization (issue 40, M2002-08-28).

• Added non-normative text to Service Data 0 to help better explain/motivate the concept.

• Added a definition of ogsi:wsdlLocation (See minutes 2002-09-06E).

• In 4.5.2, changed “ogsi:serviceType” to “ogsi:extends”.

• Added ogsi:serviceHandle and ogsi:serviceReference types (Error! Reference source not found.). Changed serviceLocator to contain sets of these.

• Added a CurrentTime SDD to GridService (M2002-09-06F).

• Removed the URI argument to GridService::FindServiceData (9.2.1). The tag of the QueryExpression argument (a QName) is used to determine the query to be performed. Similarly, removed the URI argument to NotificationSource::Subscribe (11.1.2.1). (Issue 33, M2002-09-06A)

• Changed SetTerminationTime to be two operations (see, 9.2.2 and 9.2.41 – see minutes 2002-08-21) and modified them according to M2002-09-06F. Also modified the TerminationTime SDD to reflect both times.

• Modified Destroy according to 2002-09-06H minutes.

• Added to the Registration portType (Error! Reference source not found.) an SDD to indicate the allowed extensibility arguments and an SDD to indicate registration content SDEs (2002-08-28 minutes). Also, the WSIL SDE was made optional.

• Modified RegisterService (Error! Reference source not found.) according to 2002-09-06B minutes.

• In addition to the above, the descriptions of the following issues were removed:

<table>
<thead>
<tr>
<th>Issue(s)</th>
<th>Location</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>9.2.2 ?</td>
<td>See minutes 2002-09-06F</td>
</tr>
</tbody>
</table>

- Added reference for Grid Service Primer.
- Inconsistent reference to former http(s) resolver scheme removed.
- Text corrected to reflect notion of sameness and usage of instance where it implied sameness.
- Some clean up on the definition of GSHs and the GridServiceHandles SDE.
- Removed some left over SetTerminationTime references.
- Text added to 4.5.2 to reflect immutability statement, including guidance on usage of portType extension to manage changes to interface and implementation.
- Added extendedDateTime to section 4.4.4.
- mutable='append' changed to 'extend', text written to reflect bag semantics on value elements, and text written for 'soft-append' semantics for 'mutable' on structural SDEs. Minutes 02_Oct_02.
- Text reflecting the absence of lifetime attributes in SDEs meaning 'don't know' added.
- Changed text to reflect queryServiceDataByNames (plural) and added explanatory text covering 'soft' coherency model reflected in lifetime attributes. Section 6.2.1.1.
- In addition to the above, the descriptions of the following issues were removed:

<table>
<thead>
<tr>
<th>Issue(s)</th>
<th>Location</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>6.2</td>
<td>See minutes 2002-09-07</td>
</tr>
<tr>
<td>24</td>
<td>4.5.2</td>
<td>Text changed, ready for validation.</td>
</tr>
<tr>
<td>52</td>
<td>10</td>
<td>Resolved at GGF6 not to factor.</td>
</tr>
<tr>
<td></td>
<td>10.1</td>
<td>Resolved on 2002-08-28 not to require WSIL.</td>
</tr>
</tbody>
</table>
- Moved the XML definition of extendedDateTime from 4.4.4 to 4.3. Also moved terminationTime definition from 6.2.2 to 4.7.
- Added TTL attributes to 4.4.4.
- Added XML definitions for GSR, GSH and Locator, as according to bug 37.
- Deleted text for issue 37 (which is now resolved).
- Deleted text for issue 5, which was resolved to be out of scope.
- Updated Factory::CreateService according to the resolution of bug 19.
- Added notes to guide in understanding the current draft in relation to WS-SD.
- Inserted the Rofrano-Stokes SetServiceData proposal, version 0.06, 1/6/03, except (a) the suggestion that GridServiceHandle and TerminationTime be modifiable (as per email discussion), (b) “Notes on Coherency Considerations” and (c) the revision history. Annotations were carried forward into (what is now) 6.2.2. Deleted text for issue 51.

- Accept changes introduced in Draft 6

- Begin to reflect changes agreed to at the LA f2f of the OGSi wg
- Refactored the structure of the document
  - Non-normative intro, gwsdl discussion, serviceData, other Core concepts, the list of portTypes
- Introduced the gwsdl namespace to act as a temporary holding place for WSDL 1.2 concepts needed now in OGSi. The intent is to factor out gwsdl from ogsi, allowing gwsdl to disappear when WSDL 1.2 is finished in W3C and allow ogsi to continue to evolve in future GS Spec revisions.
- Various small clean up in intro, notation and setting context sections to reflect various details of OGSi-wg discussions and WSDL 1.2 wg current thinking
- Refactored material that was in section 4 into
  - New section 4, really just introductory material now
  - Section 5 a gwsdl discussion
  - Section 6 serviceData material from the Ws-ServiceData paper submitted to the OGSi-wg
  - Section 7 concepts/properties that were in the old section 4
- Replaced the old serviceData material with new material from Ws-Servicedata
  - Added an sd namespace to separate sd from gwsdl and ogsi
  - Various clarifying edits
Added annotation capability to SDD schema definition (per Pete Vanderbilt email)
- Removed the material decorating wsdl operations with serviceData references
- New section 7 Core concepts
  - Purged notion of “most derived portType”
  - Proposed deletion of the wsdl:location concept
- New Section 8 through 13
  - Reformatted SD section, text reformatting and SDE syntax update
- Section 9: GridService
  - Some modifications to setServiceData discussion
- Section 10: NotificationSource
  - Clarified NotificationSink can be just a Web service
  - Removed “originator” SDE reference from the text, since the originator SDE does not exist
- Section 13: Registration
  - Removed references to WS-Inspection
- Added Section 17: Contributors per LA F2F

- Steve T accepted changes made by Steve G up to section 10.
- Also, made changes reflecting the move to ogsi namespace and the distinction between OGSA and OGSI
- Other textual changes as required.

- Dave S accepted changed made by Steve G after section 10.
- Added the DRAFT version of the Faults summary section (ref 7.7).
- Added the DRAFT version of the ServiceGroup section (ref 13).
- Other textual changes as required.

- Steve G accepted changes made by Steve T. up to section 10.
- Accepted changes made by Dave S after section 10
- Accepted and edited Draft section 7.7, including Steve T’s comments
- Accepted and edited Draft section 13.
16 Editor Information

Steven Tuecke
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-8711
Email: tuecke@mcs.anl.gov

Karl Czajkowski
University of Southern California, Information Sciences Institute
Email: karlcz@isi.edu

Ian Foster
Argonne National Laboratory & University of Chicago
Email: foster@mcs.anl.gov

Jeffrey Frey
IBM
Poughkeepsie, NY 12601
Email: jafrey@us.ibm.com

Steve Graham
IBM
4400 Silicon Drive
Research Triangle Park, NC, 27713
Email: sggraham@us.ibm.com

Carl Kesselman
University of Southern California, Information Sciences Institute
Email: carl@isi.edu

Peter Vanderbilt
NASA Ames Research Center
Email: pv@nas.nasa.gov

Dr. David Snelling
Fujitsu Laboratories of Europe
Hayes Park Central
Hayes End Road
Hayes, Middlesex UB4 8FE
UK
Email: d.snelling@fle.fujitsu.com

17 Contributors

We gratefully acknowledge the contributions made to this specification by the following people:
Tom Maguire, Frank Leymann, Martin Nally, John Rofrano, Ellen Stokes, Tony Storey, Sanjiva Weerawarana
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) Malcolm Atkinson, Tim Banks, Ed Boden, Brian Carpenter, Francisco Curbera, Dennis Gannon, Andrew Grimshaw, Marty Humphrey, Keith Jackson, Bill Johnston, Kate Keahey, Gregor von Laszewsiki, Lee Liming, Miron Livny, Norman Paton, Jean-Pierre Prost, Thomas Sandholm, Scott Sylvester, Von Welch and Mike Williams.

This work was supported in part by IBM and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and DE-AC03-76SF0098; by the National Science Foundation; and by the NASA Information Power Grid project.

19 Glossary

20 References

20.1 Normative References

[RFC 1738]

[RFC 2119]

[RFC 2246]

[RFC 2396]

[RFC 2616]

[RFC 2818]

[RFC 3023]

[WSDL 1.2]


20.2 Informative References

[Globus Overview]

Globus: A Toolkit-Based Grid Architecture, I. Foster, C. Kesselman, Authors. In [Grid Book], 259-278.

[Grid Anatomy]


[Grid Book]


[Grid Physiology]


[Grid Service Primer]

A Primer for Grid Services, Tim Banks, 2002. ???

[JAX-RPC]

Java™ API for XML-Based RPC (JAX-RPC), http://java.sun.com/xml/jaxrpc/docs.html

[Web Services Book]


21 XML and WSDL Specifications

This Section will contain the full WSDL types, message, and portType for each of the operations described in this document. Watch this space.

Pending agreement from the OGSI-WG community on the directions and changes in this draft of the specification, the authors will produce formal WSDL and related XML definitions shortly after GGF5.