GWD-R (draft-ggf-ogsi-gridservice-33)
Open Grid Services Infrastructure (OGSI)
http://www.ggf.org/ogsi-wg

Editors:
S. Tuecke, ANL
K. Czajkowski, USC/ISI
I. Foster, ANL
J. Frey, IBM
S. Graham, IBM
C. Kesselman, USC/ISI
T. Maquire, IBM
T. Sandholm, ANL
D. Snelling, Fujitsu Labs
P. Vanderbilt, NASA

June 27, 2003

Open Grid Services Infrastructure (OGSI)

Version 1.0

Status of This Memo

This document provides information to the community regarding the specification of the Open Grid Services Infrastructure (OGSI). Distribution of this document is unlimited.

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.

GLOBAL GRID FORUM

office@gridforum.org
www.ggf.org

Full Copyright Notice

Intellectual Property Statement

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

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

Contents

1 Introduction

The *Open Grid Services Architecture* (OGSA) [Grid Physiology] integrates key Grid technologies [Grid Book, Grid Anatomy] (including the Globus Toolkit® [Globus Overview]) with Web services mechanisms [Web Services Book] to create a distributed system framework based on the *Open Grid Services Infrastructure (OGSI)*. A *Grid service instance* is a (potentially transient) service that conforms to a set of conventions, expressed as Web Service Definition Language (WSDL) interfaces, extensions, and behaviors, for such purposes as lifetime management, discovery of characteristics, and notification. 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.

OGSI version 1.0 defines a component model that extends WSDL and XML Schema definition to incorporate the concepts of

- stateful Web services,
- extension of Web services interfaces,
- asynchronous notification of state change,
- references to instances of services,
- collections of service instances, and
- service state data that augments the constraint capabilities of XML Schema definition.

In this specification we define the minimal, integrated set of extensions and interfaces necessary to support definition of the services that will compose OGSA.

No specification is written in isolation, and Web services and XML are particularly dynamic and evolving environments. We intend to ensure that the evolution of OGSI conforms with broader standards that evolve. Many of the concepts we define – for example, serviceData (§6) – are special cases of more general concepts that may appear in XML documents, messages, and Web services. In addition, we anticipate that work to implement the OGSI Web services component model in various hosting environments, such as J2EE, will lead to the need for modifications to subsequent revisions of this OGSI V1.0 specification.

In this document, we propose detailed specifications for the conventions that govern how clients create, discover, and interact with a Grid service instance.[1] That is, we specify (1) how Grid service instances are named and referenced; (2) the base, common interfaces (and associated behaviors) that all Grid services implement; and (3) 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 any client program that follows the conventions can invoke any Grid service instance conforming to this specification (of course, subject to policy and compatible protocol bindings).

Our presentation is deliberately terse; 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.

[1] We use the term client loosely to mean an entity that calls a Grid service. In many scenarios, a Grid service is a client of other Grid services.

This document has four major parts:

1. Sections 1 through 3 are introductory, non-normative, and add detail to the context set by [Grid Physiology].
2. Sections 4 through 7 introduce how the Grid services specification builds on the Web Services Description Language. We define a gwsdl extension to WSDL 1.1, intended as a temporary measure until WSDL 1.2 is complete. We also define the notion of serviceData to expose state data of a service instance, and we define other core Grid services concepts.
3. Sections 8 through 13 define various required and optional portTypes, including GridService, HandleResolver, Notification, Factory, and serviceGroup.
4. Sections 14 through 19 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 1: Prefixes and namespaces used in this specification.

Prefix	Namespace
ogsi	"http://www.gridforum.org/namespaces/2003/03/OGSI"
gwsdl	"http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
sd	"http://www.gridforum.org/namespaces/2003/03/serviceData"
wsdl	“http://schemas.xmlsoap.org/wsdl/”
http	"http://www.w3.org/2002/06/wsdl/http"
xsd	"http://www.w3.org/2001/XMLSchema"
xsi	"http://www.w3.org/2001/XMLSchema-instance"

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [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*, 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 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, we examine in this section several details 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 between OGSI and the existing (and evolving) Web services framework. We examine both the client-side programming patterns for Grid services and a conceptual hosting environment for Grid services.

The patterns described in this section are enabled but not *required* by OGSI. We discuss these patterns here to help put into context certain details described later in 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 (§12) can be used to create instances implementing a given set of portType(s). Each Grid service instance has a notion of identity with respect to the other instances in the distributed Grid (§7.5.2.1). Each instance can be characterized as state coupled with behavior published through type-specific operations. The architecture also supports introspection in that a client application can ask a Grid service instance to return information describing itself, such as the collection of portTypes that it implements.

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

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

Each of the characteristics introduced above (stateful instances, typed interfaces, global names, etc.) is frequently also cited as a fundamental characteristic of *distributed object-based systems*. There are, however, 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 we 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 instance 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 at 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 Grid service instances. Grid service providers are free to implement the semantic contract of the service description in any technology and hosting

architecture of their choosing. We envision implementations in J2EE, .NET, traditional commercial transaction management servers, traditional procedural Unix servers, and so forth. We also envision service implementations in a wide variety of both object-oriented and non-object-oriented programming languages.

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. The *Web Services Invocation Framework* [WSIF] and *Java API for XML RPC* [JAX-RPC] are among the many examples of infrastructure software that provide this capability.

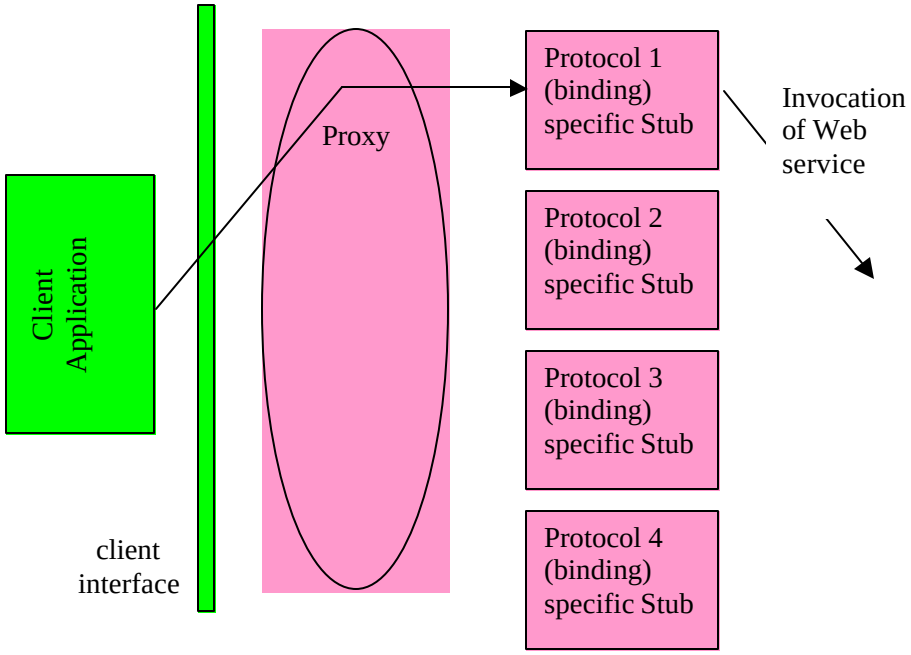

Figure 1: Possible client-side runtime architecture

Figure 1 depicts a possible (but not required) client-side architecture for OGSI. In this approach, a clear separation exists between the client application and the client-side representation of the Web service (proxy), including components for marshaling 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 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, C#). This interface is a front-end to specific parameter marshaling 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.

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 instance, 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 instance. Although certain circumstances demand potential efficiencies gained this style of customization, this approach introduces significant inflexibility into a system and therefore should only be used under extraordinary circumstances.

We expect the stub and client s ide infrastructure model that we describe to be a common approach to enabling client access to Grid services. This includes both application specific services and common infrastructure services that are defined by OGSA. Thus, for most developers using Grid services, the infrastructure and application-level services appear in the form of a class library or programming language interface that is natural to the caller.

WSDL and the GWSDL extensions provide support for enabling heterogeneous tools and enabling infrastructure software.

3.3 Client Use of Grid Service Handles and References

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

OGSI provides a mechanism, the HandleResolver (see §10) to support client resolution of a Grid Service Handle into a Grid Service Reference. Figure 2 shows a client application that needs to resolve a GSH into a GSR.

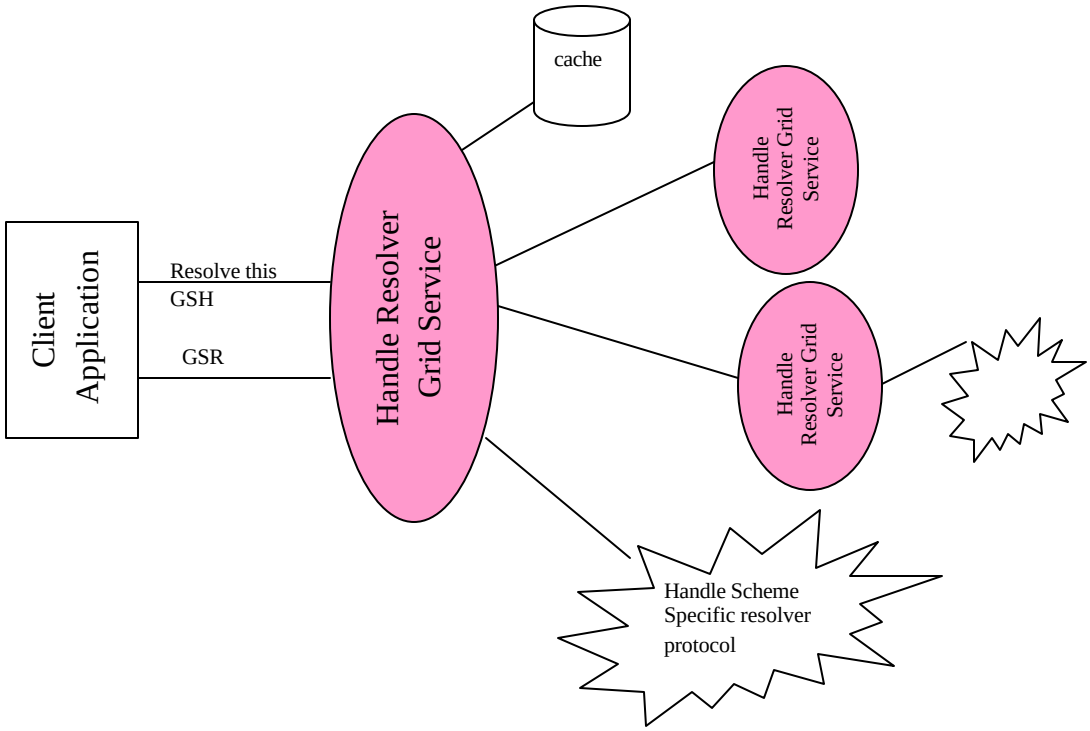

Figure 2: Resolving a GSH

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

3.4 Relationship to Hosting Environment

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

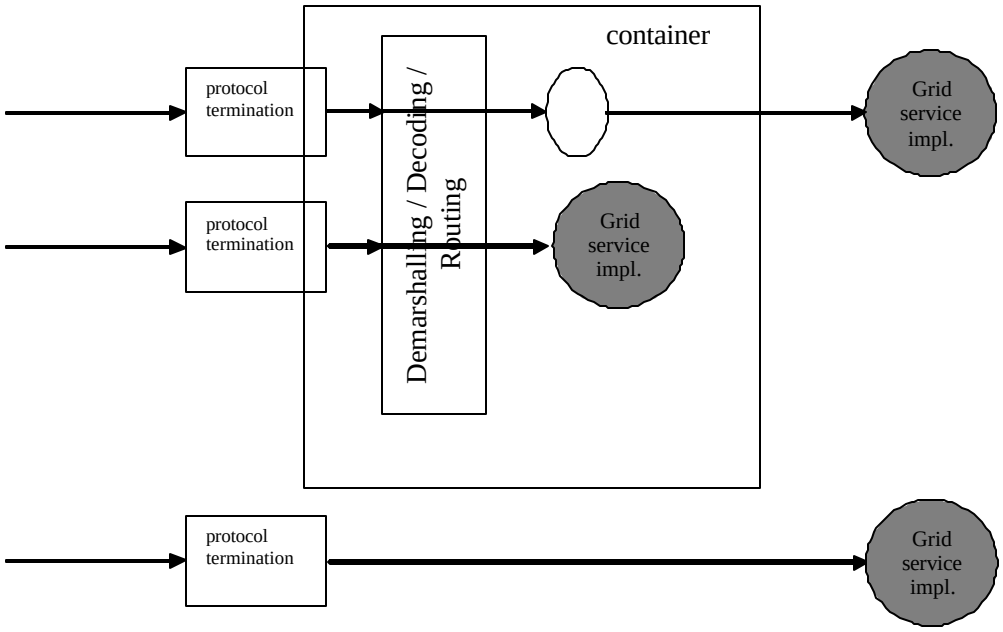

Figure 3: Two approaches to the implementation of argument demarshaling functions in a Grid service hosting environment

Figure 3 illustrates these differences by showing two different approaches to the implementation of argument demarshaling 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 3, we illustrate two Grid service instances (the ovals) associated with container-managed components (e.g., EJBs within a J2EE container). Here, the message is dispatched to these components, with the container frequently providing facilities for demarshaling 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 instance 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 service behavior. At the bottom of Figure 3, we depict another scenario wherein the entire behavior of the Grid service instance, including the demarshaling/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 demarshaling. For example, the container implementation may provide lifetime management functions, automatic support for authorization and authentication, request logging, intercepting lifetime management functions, and terminating service instances when a service lifetime expires or an explicit destruction request is received. Thus, we avoid the need to reimplement 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 use in our Grid service specification; these conventions have been incorporated in WSDL 1.2 [WSDL 1.2 DRAFT].
- Defining *service data*, which provides a standard way for representing and querying metadata and state data from a service instance.
- Introducing a series of core properties of *Grid service*, including:
    - Defining *Grid service description* and *Grid service instance*, as organizing principles for their extension and their use.
    - Defining how OGSI models time.
    - Defining the Grid Service Handle and Grid Service Reference constructs, which we use to refer to Grid service instances.
    - Defining a common approach for conveying fault information from operations. This approach defines a base XML Schema definition and associated semantics for WSDL fault messages to support a common interpretation. The approach simply defines the base format for fault messages, without modifying the WSDL fault message model.
    - Defining the lifecycle of a Grid service instance.

5 WSDL Extensions and Conventions

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

Instead, OGSI defines an extension to WSDL 1.1, isolated to the wsdl:portType element, that provides the minimal required extensions to WSDL 1.1. These extensions to WSDL 1.1 match equivalent functionality agreed to by the W3C Web Services Description Working Group. Once WSDL 1.2 [WSDL 1.2] is published as a recommendation by the W3C, the Global Grid Forum commits to defining a follow-on version of OGSI that exploits WSDL 1.2, and to defining a translation from this OGSI v1.0 extension to WSDl 1.2.

We define a separate (and temporary) namespace, with the prefix *gwsdl*, to isolate the modifications to WSDL 1.1. In particular, gwsdl adds the following new constructs to the wsdl:portType element in the fashion proposed in [WSDL 1.2 DRAFT]to support open content model and portType extension.

The following is the XSD definition of gwsdl:portType:

```
…
targetNamespace=
   http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions
xmlns:gwsdl=
   http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions
…

<element name="portType" type="gwsdl:portTypeType"/>
```

```
<complexType name="portTypeType">
   <complexContent>
      <extension base="wsdl:portTypeType">
         <sequence>
            <any namespace="##other"
                 minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
         <attribute name=”extends” use=”optional”>
            <simpleType>
               <list itemType=”QName”/>
            </simpleType>
         </attribute>
         <anyAttribute namespace="##other"/>
      </extension>
   </complexContent>
</complexType>
```

The *extends* attribute on the gwsdl:portType element is a list of QNames, where each QName MUST refer to either a wsdl:portType or a gwsdl:portType.

In light of portType extension, there is a required clarification with respect to the namespace of operation elements. Although WSDL 1.1 allows operation name overloading (a WSDL 1.1 portType can have multiple operation child elements with the same name), the W3C Web Services Description Working Group decided to restrict this feature. The following is from [WSDL 1.2 DRAFT]:

> A port type operation component describes an operation that a given port type supports. An operation is a set of message references. Message references may be to messages this operation accepts, that is input messages, or messages this operation sends, that is output or fault messages.
>
> Port type operation components are local to port type components, they cannot be refered to by QName, despite having both {name} and {target namespace} properties
>
> The properties of the Port Type Operation Component are as follows:
>
> - {name} An NCName as defined by [XML Namespaces].
> - {target namespace} A namespace name, as defined in [XML Namespaces].
> - {variety} One of Input-Only, Output-Only, Input-Output or Output-Input
> - {messages} A set of message reference components
>
> For each port type operation component in the {operations} property of a port type component the combination of {name} and {target namespace} properties must be unique.
>
> In cases where, due to a port type extending one or more other port types, two or more port type operation components have the same value for their {name} and {target namespace} properties, then the component models of those port type operation components MUST be equivalent (see 2.12 Equivalence of components). If the port type operation components are equivalent then they are considered to collapse into a single component. It is an error if two port type operation components have the same value for their {name} and {target namespace} properties but are not equivalent.
>
> Note:

> Due to the above rules, if two port types that have the same value for their {target namespace} property also have one or more operations that have the same value for their {name} property then those two port types cannot both form part of the derivation chain of a derived port type unless those operations are the same operation. Therefore it is considered good practice to ensure , where necessary, that operation names within a namespace are unique, thus allowing such derivation to occur without error.

And about Equivalence of components:

> Two components of the same type are considered equivalent if the values of the properties of one component are the same as the values of the properties in the second component.
>
> With respect to top-level components (messages, port types, bindings and services) this effectively translates to name-based equivalence given the constraints on names. That is, given two top-level components of the same type, if the {name} properties have the same value and the {target namespace} properties have the same values then the two components are in fact, the same component.

Operation naming and extension in gwsdl portTypes is to be interpreted in the same fashion as defined by the WSDL 1.2 DRAFT text above.

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, update and change notification. Since this concept is applicable to any Web service including those used outside the context of Grid applications, we propose a common approach to exposing Web service state data called "serviceData." We are endeavoring to introduce this concept to the broader Web services community.

In order to provide a complete description of the interface of a stateful Web service (i.e., a *Grid service*), it is necessary to describe the elements of its state that are externally observable. By externally observable, we mean that the state of the service instance 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 instance itself. The need to declare service data as part of the service's external interface is roughly equivalent to the idea of declaring attributes as part of an object-oriented interface described in an object-oriented interface definition language (IDL). Service data can be exposed for read, update, or subscription purposes.

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

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

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

6.1 Motivation and Comparison to JavaBean Properties

The OGSI specification introduces the serviceData concept to provide a flexible, properties-style approach to accessing state data of a Web service. The serviceData concept is similar to the notion of a public instance variable or field in object-oriented programming languages such as Java™, Smalltalk and C++.

ServiceData is similar to JavaBean™ properties. The JavaBean model defines conventions for method signatures (getXXX/setXXX) to access properties, and helper classes (BeanInfo) to document properties. The OGSI model uses the serviceData elements and XML Schema types to achieve a similar result.

The OGSI specification has chosen not to require getXXX and setXXX WSDL operations for each serviceData element, although service implementers MAY choose to define such type safe get and set operations themselves. Instead, OGSI defines extensible operations for querying (get), updating (set), and subscribing to notification of changes in serviceData elements. Simple expressions are required by OGSI to be supported by these operations, which allows for access to serviceData elements by their names, relative to a service instance. This by-name approach gives functionality roughly equivalent to the getXXX and setXXX approach familiar to JavaBean and Enterprise JavaBean programmers. However, these OGSI operations MAY be extended by other service interfaces to support richer query, update, and subscription semantics, such as complex queries that span multiple serviceData elements in a service instance.

The serviceDataName element in a GridService portType definition corresponds to the BeanInfo class in JavaBeans. However, OGSI has chosen an XML (WSDL) document that provides information about the serviceData, instead of using a serializable implementation class as in the BeanInfo model.

6.2 Extending portType with serviceData

ServiceData defines a new portType child element named serviceData, used to define serviceData elements, or SDEs, associated with that portType. These serviceData element definitions are referred to as serviceData declarations, or SDDs. Initial values for those serviceData elements (marked as "static" serviceData elements) MAY be specified using the staticServiceDataValues element within portType. The values 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.

Note in the following the use of the gwsdl:portType, which allows the appearance of elements from other namespaces.

```
   <gwsdl:portType name="NCName"> *
      <wsdl:documentation … /> ?
      <wsdl:operation name="NCName"> … </wsdl:operation> ?
…
      <sd:serviceData name="NCName" … /> *
      <sd:staticServiceDataValues>?
         <some element>*
```

```
      </sd:staticServiceDataValues>
…
   </gwsdl:portType>
```

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

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

6.2.1 Structure of the serviceData Declaration

Since sd:serviceData declarations are defining XML elements that may appear in sd:serviceDataValue and sd:staticServiceDataValue elements, the XML Schema definition of sd:serviceData is modeled after the XML Schema definition of xsd:element. The sd:serviceData element uses five attributes with the same interpretation as xsd:element: name, type, minOccurs, maxOccurs,and nillable. The other xsd:element attributes do not make sense within the context of sd:serviceData. The sd:serviceData element also allows two additional elements: mutability, and modifiable.

The XML Schema definition for sd:serviceData is as follows.

```
…
   targetNamespace =
     ″http://www.gridforum.org/namespaces/2003/serviceData″
   xmlns:sd =
     ″http://www.gridforum.org/namespaces/2003/03/serviceData″
…

<attributeGroup name="occurs">
   <attribute name="minOccurs"
              type="nonNegativeInteger"
              use="optional"
              default="1"/>
   <attribute name="maxOccurs">
      <simpleType>
         <union memberTypes="nonNegativeInteger">
            <simpleType>
               <restriction base="NMTOKEN">
                  <enumeration value="unbounded"/>
               </restriction>
            </simpleType>
         </union>
```

```
      </simpleType>
   </attribute>
</attributeGroup>

<complexType name="ServiceDataType">
   <sequence>
      <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="name" type="NCName"/>
   <attribute name="type" type="QName"/>
   <attribute name="nillable"
              type="boolean"
              use="optional"
              default="false"/>
   <attributeGroup ref="sd:occurs"/>
   <attribute name="mutability" use="optional" default="extendable">
      <simpleType>
         <restriction base="string">
            <enumeration value="static"/>
            <enumeration value="constant"/>
            <enumeration value="extendable"/>
            <enumeration value="mutable"/>
         </restriction>
      </simpleType>
   </attribute>
   <attribute name="modifiable" type="boolean" default="false"/>
   <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<element name="serviceData" type="sd:ServiceDataType"/>

<xsd:complexType name=”ServiceDataValuesType”>
   <xsd:sequence>
      <xsd:any namespace="##any" minOccurs="0"
                                 maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>

<element name=”serviceDataValues”
         type=”sd:ServiceDataValuesType”/>

<element name=”staticServiceDataValues”
         type=”sd:ServiceDataValuesType”/>
```

The serviceData attributes are as follows.

- 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 among 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
  - This value 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
  - This value defines the XML Schema type of the serviceData element value
- modifiable = "boolean" : default to false
  - If true, it is legal for requestors to directly update the serviceData value through the setServiceData operation (see §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" | "extendable" | "mutable" : default to extendable
  - This value indicates whether and how the values of a serviceData element can change (see §6.2.3).
- {any attributes with non-schema namespace}
  - Open content on the attributes of serviceData declaration is allowed.
- Content
  - This is an open content element model, meaning elements from any other namespace (besides serviceData) may appear as child elements of the serviceData element.

6.2.2 Using serviceData, an Example from GridService portType

To show how serviceData can be used, we present an example, namely, the GridService portType, described in Section 9. The (non-normative) serviceData elements declared for the Grid Service portType are as follows.

```
<wsdl:definitions …

<gwsdl:portType name="GridService" …>
   <wsdl:operation name=…> … </wsdl:operation>
   …

   <sd:serviceData name="interface" type="xsd:QName"
        minOccurs="1" maxOccurs="unbounded"
        mutability="constant"/>
   <sd:serviceData name="serviceDataName" type="xsd:QName"
        minOccurs="0" maxOccurs="unbounded"
        mutability="mutable" nillable="false"/>
   <sd:serviceData name="factoryHandle"
        type="ogsi:HandleType"
        minOccurs="1" maxOccurs="1"
        mutability="constant" nillable="true"/>
   <sd:serviceData name="gridServiceHandle"
```

```
        type="ogsi:HandleType"
        minOccurs="0" maxOccurs="unbounded"
        mutability="extendable"/>
   <sd:serviceData name="gridServiceReference"
        type="ogsi:ReferenceType"
        minOccurs="0" maxOccurs="unbounded"
        mutability="mutable"/>
   <sd:serviceData name="findServiceDataExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="1" maxOccurs="unbounded"
        mutability="static"/>
   <sd:serviceData name="terminationTime" type="ogsi:terminationTime"
        minOccurs="1" maxOccurs="1"
        mutability="mutable"/>
   <sd:serviceData name="setServiceDataExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="2" maxOccurs="unbounded"
        mutability="static" />
…
```

The normative description of the individual serviceData elements is given in Section 9.1.

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

```
…
   xmlns:crm="http://gridforum.org/namespaces/2002/11/crm"
   xmlns:tns="http://example.com/exampleNS"
   xnlns="http://example.com/exampleNS">

<sd:serviceDataValues>
   <ogsi:interface>crm:GenericOSPT</ogsi:interface>
   <ogsi:interface>ogsi:GridService</ogsi:interface>

   <ogsi:serviceDataName>ogsi:interface
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:serviceDataName
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:factoryHandle
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:gridServiceHandle
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:gridServiceReference
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:findServiceDataExtensibility
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:terminationTime
   </ogsi:serviceDataName>
   <ogsi:serviceDataName>ogsi:setServiceDataExtensibility
   </ogsi:serviceDataName>

   <ogsi:factoryHandle>someURI</ogsi:factoryHandle>

   <ogsi:gridServiceHandle>someURI</ogsi:gridServiceHandle>
   <ogsi:gridServiceHandle>someOtherURI</ogsi:gridServiceHandle>

   <ogsi:gridServiceReference>…</ogsi:gridServiceReference>
   <ogsi:gridServiceReference>…</ogsi:gridServiceReference>
```

```
   <ogsi:findServiceDataExtensibility
         inputElement="ogsi:queryByServiceDataNames" />

   <ogsi:terminationTime after="2002-11-01T11:22:33"
                         before="2002-12-09T11:22:33"/>

   <ogsi:setServiceDataExtensibility
         inputElement="ogsi:setByServiceDataNames" />
   <ogsi:setServiceDataExtensibility
         inputElement="ogsi:deleteByServiceDataNames" />

</sd:serviceDataValues>
```

6.2.3 Mutability

A mutability attribute on the serviceData element declaration indicates how a serviceData element's values may change over its lifetime.

- mutability="static": this attribute value implies that the SDE value is assigned in the WSDL declaration (staticServiceDataValues) and MUST remain that value for any instance of that portType. A "static" SDE is analogous to a class member variable in programming languages.
- mutability="constant": this attribute value 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="extendable": this attribute value implies that once elements are in the SDE value, they MUST be part of the SDE value for the lifetime of the Grid service instance. New elements MAY be added to the SDE value, but once these elements are added, they MUST NOT be removed.
- mutability="mutable": this attribute value implies that any of the elements in the SDE value MAY be removed at any time and others MAY be added.

Note: The functionality described here is different from the "fixed" and "default" attributes in theXML Schema's element definition. While the XML Schema's "fixed" attribute could be used to suggest a "static" value, the "extendable" and "mutable" attributes would have to be modeled by a mutability attribute. The case where mutability="constant" would be used to specify a property that does not change after a value is assigned. However, the value is not assigned by the service description but, rather, must be initialized at run time.

6.3 serviceDataValues

Each service instance is associated with a collection of serviceData elements: those serviceData elements defined within the various portTypes that form the service's interface, and also, potentially, additional serviceData elements added at runtime (see §6.5). We call the set of serviceData elements associated with a service instance its "serviceData set". A serviceData set may also refer to the set of serviceData elements aggregated from all serviceData elements declared in a portType interface hierarchy (see §6.4).

Each service instance MUST have a "logical" XML document, with a root element of serviceDataValues that contains the serviceData element values. We showed an example of a serviceDataValues element above. A service implementation is free to choose how the SDE

values are stored; for example, it may store the SDE values not as XML but as instance variables that are converted into XML or other encodings as necessary.

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

Through the use of the staticServiceDataValues element, 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 interface 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 declarations are legal and declare two initial values for tns:otherSD: "initial value 1" and "initial value 2".

```
<wsdl:definitions xmlns:tns=”xxx” targetNamespace=”xxx”>
   <gwsdl:portType name="otherPT">
      <wsdl:operation name=…> … </wsdl:operation>
…
      <sd:serviceData name="otherSD" type=”xsd:String”
                     mutability=”static” maxOccurs=”unbounded”/>

      <sd:staticServiceDataValues>
         <tns:otherSD>initial value 1</tns:otherSD>
      </sd:staticServiceDataValues>
…
   </gwsdl:portType>

   <gwsdl:portType name="exampleSDUse" extends=“tns:otherPT”>
      <wsdl:operation name=…> … </wsdl:operation>
…
      <sd:serviceData name="sd1" type=”xsd:String”
                     mutability=”static” />
      <sd:serviceData name="sd2" type=”tns:SomeComplexType”/>

      <sd:staticServiceDataValues>
         <tns:sd1>an initial value</tns:sd1>
         <tns:otherSD>initial value 2</tns:otherSD>
      </sd:staticServiceDataValues>
…
   </gwsdl:portType>
…
</wsdl:definitions>
```

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

6.4 SDE Aggregation within a portType Interface Hierarchy

WSDL 1.2 has introduced the notion of multiple portType extension, and 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 service instance. 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.

```
<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 the four portTypes defined here are as follows.

if a service implements…	its serviceData set contains…
Pt1	sd1
Pt2	sd1, sd2
Pt3	sd1, sd3
Pt4	sd1, sd2, sd3, sd4

6.4.1 Initial Values of Static SDEs within a portType Interface Hierarchy

Initial values of static SDEs can be aggregated down a portType interface hierarchy. However, the cardinality requirements (minOccurs and maxOccurs) MUST be preserved. For example, in the following, a service instance that implements pt1 would have the value <sd1>1</sd1> for SDE named sd1.

```
<gwsdl:portType name="pt1">
   <sd:serviceData name="sd1" minOccurs="1" maxOccurs="1"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd1>1</sd1>
   </sd:staticServiceDataValues>

</gwsdl:portType>
```

In the following, however, a service instance that implements pt2 would inherit the value <sd1>1</sd1> for SDE named sd1 and would have the value <sd2>2</sd2> for the SDE named sd2.

```
<gwsdl:portType name="pt2" extends="pt1">
   <sd:serviceData name="sd2" minOccurs="1" maxOccurs="1"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd2>2</sd2>
   </sd:staticServiceDataValues>
</gwsdl:portType>
```

A service instance that implements pt3 would have two values <sd3>3a</sd3> and <sd3>3b</sd3> for the SDE named sd3. It would, of course, inherit the value for the SDE named sd1.

```
<gwsdl:portType name="pt3" extends="pt1">
   <sd:serviceData name="sd3" minOccurs="1" maxOccurs="unbounded"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd3>3a</sd3>
      <sd3>3b</sd3>
   </sd:staticServiceDataValues>
</gwsdl:portType>
```

A 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 most likely one would be defined in a portType that extends pt4).

```
<gwsdl:portType name="pt4" extends="pt1">
   <sd:serviceData name="sd4" minOccurs="0" maxOccurs="unbounded"
                           mutability="static"/>
</gwsdl:portType>
```

A service instance that implements pt5 could not be created. Since there is no initial value for sd5 and since 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.

```
<gwsdl:portType name="pt5" extends="pt1">
   <sd:serviceData name="sd5" minOccurs="1" maxOccurs="unbounded"
                           mutability="static"/>
</gwsdl:portType>
```

A 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) that exceeds the maxOccurs value for the SDE named sd1, an error is generated when the instance is created.

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

A service instance that implements pt7 has an 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, inherited from pt2. The SDE named pt3 has three values: <sd3>3a</sd3>, <sd3>3b</sd3> (inherited from pt3), and <sd3>7</sd3> locally defined. Moreover, there is a locally defined value for the SDE named sd7 (<sd7>7</sd7>).

```
<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>
```

In summary, values for static SDEs are aggregated down a portType interface 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 most naturally defined in a service's interface definition, situations can arise where it is useful to add or remove serviceData elements dynamically to or from an instance. The means by which such updates are achieved is implementation-specific; for example, a service instance MAY implement operations for adding a new service data element.

The GridService portType (§9) illustrates the use of dynamic SDEs. This contains a serviceData element named "serviceDataName" that lists the serviceData elements currently defined. This property of a service instance may return a superset of the service data elements declared in the GWSDL defining the service interface, allowing the requestor to use the `subscribe` operation (§11.1.2.1) if this serviceDataSet changes, and `findServiceData` (§9.2.1) operation to determine the current serviceDataSet value.

7 Core Grid Service Properties

We discuss here a number of properties and concepts common to all Grid services.

7.1 Service Description and Service Instance

We distinguish in OGSI 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 used primarily 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, and so forth. 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 and (in a very rudimentary, non-normative fashion) semantics. *Interface syntax* is 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 more formal descriptions. These names can then be used by clients to discover services with desired semantics, by searching for service instances and factories with the appropriate names. The use of namespaces to define these names also provides a vehicle for assuring globally unique names.

7.2 Modeling Time in OGSI

The need arises at various points throughout this specification 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 are out of range because of inadequate synchronization, where "appropriately" MAY include refusing to 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 represent both zero time and infinite time (for examples, see §7.3 and §9.2). Zero time SHOULD be represented by a time in the past. However, infinite time requires an extended notion of time. We therefore introduce the following type in the ogsi namespace, which MAY be used in place of xsd:dateTime when a special value of "infinity" is appropriate.

```
…  targetNamespace =
```

```
     "http://www.gridforum.org/namespaces/2003/03/OGSI"

<simpleType name="ExtendedDateTimeType">
   <union memberTypes="ogsi:InfinityType xsd:dateTime"/>
</simpleType>

<simpleType name="InfinityType">
   <restriction base="string">
      <enumeration value="infinity"/>
   </restriction>
</simpleType>
```

7.3 XML Element Lifetime Declaration Properties

Since serviceData elements may represent instantaneous 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.

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

The three lifetime declaration properties are as follows:

- 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. (The definition of ExtendedDateTimeType is given in §7.2.)

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:attribute name="goodFrom" type="ogsi:ExtendedDateTimeType"/>
<xsd:attribute name="goodUntil" type="ogsi:ExtendedDateTimeType"/>
<xsd:attribute name="availableUntil" type="ogsi:ExtendedDateTimeType"/>

<xsd:attributeGroup name="LifeTimePropertiesGroup">
   <xsd:attribute ref="ogsi:goodFrom" use="optional"/>
   <xsd:attribute ref="ogsi:goodUntil" use="optional"/>
   <xsd:attribute ref="ogsi:availableUntil" use="optional"/>
</xsd:attributeGroup>
```

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

```
<wsdl:definitions
```

```
   targetNamespace=”http://example.com/ns”
   xmlns:n1=”http://example.com/ns”
   … >

   <wsdl:types>
      <xsd:schema …
         “targetNamespace=http://example.com/ns”
         …>
         <xsd:complexType name="MyType">
            <xsd:sequence>
               <xsd:element name="e1" type="xsd:string" minOccurs="1"/>
               <xsd:element name="e2" type="xsd:string" minOccurs="1"/>
               <xsd:element name="e3" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
         <anyAttribute namespace="##any"/>
         </xsd:complexType>
      </xsd:schema>
   </wsdl:types>
   …
   <gwsdl:portType name=”MyPortType”>
      …
      <sd:serviceData name=”mySDE” type= “n1:MyType”
                      minOccurs=”1” maxOccurs=”1”
                      mutability=”mutable”/>
      …
   </gwsdl:portType>
   …
</wsdl:definitions>
```

And within the service instance’s serviceDataValues:

```
<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”>
         ghi
      </n1:e3>
   </n1:mySDE>
</sd:serviceDataValues>
```

The goodFrom and goodUntil attributes of the n1:mySDE element refer to all of its child elements. These attributes declare to a consumer of this SDE the expected lifetime for the element’s value, which in this example is from 10:20 am until 11:20 am EST on 27 April 2002. A consumer receiving this SDE at 10:20 am is thus advised that after one hour the SDE value is likely no longer valid. The client might therefore query the service instance again at 11:20 (giving the same name for the SDE: n1:mySDE) to obtain a newer value.

The availableUntil refers *not* 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 instance until 28 April 2002 10:20 am EDT for the serviceData element named n1:mySD, and receive a response with an updated copy of its value. After that time, however, the client SHOULD NOT assume such a value will be available: Such a query might 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 one day, but after that time the service instance may no longer have an SDE with the name n1:mySDE. Despite this suggestion of availability, an SDE MAY cease to be available prior to the availableUntil time exposed to clients.

It can be desirable for the SDE's value to contain child elements with lifetimes different from those declared in the parent element. To meet this requirement, an 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 expected to be valid only 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.

7.4 Interface Naming and Change Management

A critical issue in distributed systems is enabling the upgrade of services over time. This requirement 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 OGSI mechanisms, requirements, and recommendations that address it.

7.4.1 The Change Management Problem

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

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 is typically defined in specification documents such as this one, although 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 instance having behaviors that are ill-defined or at odds with the interface specification. Nonetheless, clients of that service interface may come to rely upon such implementation semantics, whether by accident or by design.

In order for a client to be able reliably to discover and use a Grid service instance, the client must be able to determine whether it is compatible with both of these two service definitions. In other words, does the Grid service description include 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 instance 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 backward-compatible.

However, a client MUST be able to detect if a Grid service description is changed in a way that is *not* backward-compatible. Again, the lack of backward compatibility could be the result of incompatible changes to the Grid service interface or implementation. For example, 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 names for 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 description may change frequently at 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 instance without modification. Such interface extensions MAY also be modeled with totally new portTypes, but this approach is not recommended.

7.5 Naming Grid Service Instances

Each Grid service instance is named, globally and for all time, by one or more *Grid Service Handles (GSH)*. 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 client wishing to communicate with a service instance must resolve the GSH to a *Grid Service Reference (GSR)*. A GSR contains all information that a client requires to communicate with the service instance via one or more network protocol bindings.

Like any URI, a GSH consists of a *scheme*, followed by a string containing scheme-specific data. 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 instance that implements the HandleResolver portType (see § 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 instance) 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. A GSR may be localized to a given client context or hosting environment, however; the scope of portability for a GSR is determined by the binding mechanism(s) it supports. A GSR localization format SHOULD include sufficient scoping information such that reference-passing between services can be detected in the binding layer. For example, the use of a GSR passed out of its localized scope SHOULD NOT lead to erroneous binding to the wrong service instance.

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.

Nevertheless, 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. An invalid GSR MUST be detectable by a client attempting to use the GSR, and MAY be detectable by other means for some GSR schemes. A GSR MAY become invalid during the lifetime of the Grid service instance. Typically this situation 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 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 by 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 instance. 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 is as follows.

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:element name="reference" type="ogsi:ReferenceType"/>

<xsd:complexType name="ReferenceType" abstract="true">
   <xsd:attribute ref="ogsi:goodFrom" use="optional"/>
   <xsd:attribute ref="ogsi:goodUntil" use="optional"/>
</xsd:complexType>
```

The two attributes, defined previously in §7.3,give the expected lifetime of the GSR, not of the service instance to which it refers. A client SHOULD NOT call a service instance 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 contain just 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 element MUST contain exactly one wsdl:service element and MAY contain other elements.

The XML definition for a WSDL GSR is as follows.

```
targetNamespace = http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:complexType name="WSDLReferenceType">
   <xsd:complexContent>
      <xsd:extension base="ogsi:ReferenceType">
         <xsd:sequence>
            <xsd:any namespace=”http://schemas.xmlsoap.org/wsdl/”
                 minOccurs="1" maxOccurs="1" processContents="lax"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

7.5.2 Grid Service Handle (GSH)

Grid Service 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 refer to more than one Grid service instance, whether or not they exist simultaneously. See §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 gridServiceHandle service data element values (§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 gridServiceHandle SDE values, then they MUST all refer to that same Grid service instance. The gridServiceHandle SDE values 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 instance 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.

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="handle" type="ogsi:HandleType"/>

<xsd:simpleType name="HandleType">
   <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
```

7.5.2.1 Service Instance Sameness

The preceding discussion specified that "a GSH MUST globally and for all time refer to the *same Grid service instance.*" This requirement is an important one because it allows clients to reason about, for example, the meaning of multiple calls to the same service instance. One should note, however, that the interpretation of the phrase "same Grid service instance" depends on 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, in other words, the portType(s) 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 instance, 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, and so forth. Some service descriptions may require tight state coherency between any such replicated implementations—for example, the semantics of the service description may require that the service instance 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. Other service descriptions may be defined, however, that allow for looser consistency between the various members of the distributed service implementation.

7.5.3 Service Locator

A service locator is a structure that contains zero or more GSHs, zero or more GSRs, and zero or more interface (portType) QNames. All of the GSHs and GSRs MUST refer to the same Grid service instance, and all of the interfaces MUST be implemented by that Grid service instance. Locators are used by various operations in this specification that may accept either a GSH or a GSR. The XML definition of a locator is as follows.

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="locator" type="ogsi:LocatorType"/>
```

```
<xsd:complexType name="LocatorType">
   <xsd:sequence>
      <xsd:element ref="ogsi:handle"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ogsi:reference"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name=”interface” type=”QName”
                   minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

7.6 Grid Service Lifecycle

The lifecycle of any Grid service instance is demarcated by the creation and destruction of that service instance. The actual mechanisms by which a Grid service instance is created or destroyed are fundamentally a property of the hosting environment, and as such are not defined in this document. Nevertheless, there is 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 instance by invoking the createService operation on a service instance that implements a portType that extends the Factory portType or that contains specialized methods defined to instantiate new services, such as the NotificationSource portType (a “factory” service).
- A client may request the *destruction* of a Grid service instance via either client invocation of an *explicit* destruction operation request to the service instance (see the destroy operation, supported by the GridService portType: §9) or via a *soft-state* approach, in which (as motivated and described in [Grid Physiology]) a client registers interest in the Grid service instance for a specific period of time, and if that timeout expires without the service instance having received reaffirmation of interest from any client to extend the timeout, the service instance may be automatically destroyed. Periodic reaffirmation can serve to extend the lifetime of a Grid service instance as long as is necessary (see the requestTerminationBefore/After operations in the GridService portType: §9).

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

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

Termination time MAY change nonmonotonically. 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 MUST be interpreted as a request for immediate or prior termination.

A Grid service instance MAY decide at any time to extend its lifetime. A service instance MAY also terminate itself at any time, for example if resource constraints and priorities dictate that it relinquish its resources.

Termination time is represented throughout OGSI by using the following type:

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”
```

```
<xsd:complexType name="TerminationTimeType">
   <xsd:attribute name="after" type="ogsi:ExtendedDateTimeType"
                  use="optional"/>
   <xsd:attribute name="before" type="ogsi:ExtendedDateTimeType"
                  use="optional"/>
   <xsd:attribute name="timestamp" type="xsd:dateTime”
                  use="optional"/>
</xsd:complexType>

<xsd:simpleType name="ExtendedDateTimeType">
   <xsd:union memberTypes="ogsi:InfinityType xsd:dateTime"/>
</xsd:simpleType>

<xsd:simpleType name="InfinityType">
   <xsd:restriction base="string">
      <xsd:enumeration value="infinity"/>
   </xsd:restriction>
</xsd:simpleType>
```

The *after* attribute of TerminationTimeType contains the earliest time at which the service instance plans to terminate. A value in the past indicates that the service instance may terminate at any time. The special value “infinity” indicates that the service instance plans to exist indefinitely. (ogsi:ExtendedDateTimeType is defined in §7.2, which allows for the expression of an xsd:dateTime or the special value “infinity”.)

The *before* attribute of TerminationTimeType contains the latest time at which the service instance plans to terminate. A value in the past indicates that the service instance is trying to terminate. The special value “infinity” indicates that the service instance has no plans to terminate. The *after* time MUST be less than or equal to the *before* time.

The *timestamp* attribute of TerminationTimeType contains the time, as known to the service instance to which this TerminationTimeType pertains, at which the after and before attributes are known to be valid for the service instance. This timestamp MAY be used to order TerminationTimeType values relating to a particular service instance and, in some circumstances, MAY be used to gauge the clock skew of this service instance relative to a client or some other time source.

7.7 Common Handling of Operation Faults

OGSI defines a base XSD type for all fault messages that Grid services MUST return. This simplifies problem determination by having a common base set of information that all fault messages contain.

The basis for all OGSI faults is the ogsi:FaultType XSD type:

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:element name=”fault” type=”FaultType”>

<xsd:complexType name=“FaultType”>
   <xsd:sequence>
      <xsd:element name=“description”
                   type=“xsd:string”
                   minOccurs=“0” maxOccurs=“unbounded”/>
      <xsd:element name=“originator”
```

```
                    type="ogsi:LocatorType"
                    minOccurs="1" maxOccurs="1"/>
      <xsd:element name="timestamp"
                    type="xsd:dateTime"
                    minOccurs="1" maxOccurs="1"/>
      <xsd:element name="faultcause"
                    type="ogsi:FaultType"
                    minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="faultcode"
                    type="ogsi:FaultCodeType"
                    minOccurs="0" maxOccurs="1"/>
      <xsd:element name="extension"
                    type="ogsi:ExtensibilityType"
                    minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FaultCodeType">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string">
         <xsd:attribute name="faultscheme" type="anyURI"
                        use="required"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="ExtensibilityType">
   <xsd:sequence>
      <xsd:any namespace="##any"/>
   </xsd:sequence>
</xsd:complexType>
```

It is RECOMMENDED that all faults from a Grid service instance use either the FaultType directly or an extension of FaultType. However, there MAY be circumstances in which faults from a Grid service instance are not derived from FaultType, such as when the Grid service uses legacy portTypes in addition to OGSI portTypes.

The OPTIONAL *description* element contains a plain language description of the fault. This description is expected to be helpful in explaining the fault to users. There MAY be any number of description elements.

The REQUIRED *originator* element is a locator of the service instance raising the fault.

The REQUIRED *timestamp* element is the time at which the fault occurred.

The OPTIONAL *faultcause* element is an ogsi:FaultType element that describes an underlying cause that resulted in this fault. This element is conventionally used with xsi:type to describe a more specialized fault that extends FaultType. There MAY be any number of faultcause elements. The ability to include faultcause elements in a fault allows for "chaining" of fault information up through a service invocation stack, so that a recipient of a fault can drill down through the causes to understand more detail about the reason for the fault.

The OPTIONAL *faultcode* element provides convenient support for legacy fault reporting systems (e.g., POSIX errno). The *faultscheme* attribute on faultcode MUST be a URI (not a URL) that defines the context in which the faultcode SHOULD be interpreted. For example, a URI might be defined that describes how a POSIX errno is mapped to a faultcode, and that URI would appear on any faultcode element carrying a POSIX errno.

The OPTIONAL *extension* element, if present, contains additional information specific to the fault from other XML namespaces. There MAY be any number of extensibility elements within the *extension* element.

Each fault returned by a Grid service operation MUST be listed as a separate fault response in the WSDL operation definition. Each operation fault response MUST have the same name as the fault element and MUST refer to a WSDL message definition that has a single part element named "fault", with an "element=" attribute that refers to the fault element. All Grid service operations MUST return the ogsi:fault in addition to any operation-specific faults, for example:

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

   <message name=myFaultMessage″>
      <part name=″fault″ element=″tns:myFault″/>
   </message>

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

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 an operation definition. For example, if an operation is specified to return a fault with the element myFault under some circumstance, then a particular implementation of that operation MAY return an extension of myFault in its place. This SHOULD be done by returning the myFault message with an xsi:type of the more refined fault.

7.8 Extensible Operations

Several OGSI operations accept an input argument that is an untyped extensibility element. This element allows for common patterns of behavior to be expressed in an extensible manner. To enable a client to discover the valid extensions supported by such an operation, we define a common approach for expressing extensible operation capabilities via static service data values.

For example, the NotificationSource::subscribe operation allows a client to ask a service instance for notification messages whenever portions of that service's serviceDataValues change. The specific portions of service data upon which to send notifications are defined by a subscription expression. This argument is not fully typed but is instead an extensible argument. One simple subscription expression is defined by the NotificationSource portType that can be passed in this argument to any service that implements NotificationSource. However, services that implement a

portType that inherits from NotificationSource can extend the capabilities of subscription by defining new query expressions that are more powerful or more customized to a specific problem domain. This section defines the means by which a client can determine what subscription expressions are supported by services that implement extensions to the NotificationSource portType.

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

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:complexType name="OperationExtensibilityType">
  <xsd:attribute name="inputElement" type="QName" use=”optional”/>
</xsd:complexType>
```

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

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

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

The inputElement attribute of the SDE MUST be a QName that uniquely implies the types and behavior of a particular extension to the operation. The inputElement, if present, SHOULD be the QName of an XSD element declaration that is a valid element that can be passed to an operation as an extensible input argument. All other properties of the extensible operation are implied by the inputElement unless they are explicitly defined in an extension of OperationExtensibilityType. For example, the inputElement MAY imply the type of output parameters from the operation and MAY imply the semantics of how the operation when it receives this inputElement.

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

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

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

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

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

8 Grid Service Interfaces

This specification defines, and embodies as WSDL portTypes and associated behaviors, a collection of common distributed computing patterns that are considered to be fundamental to OGSI. These OGSI portTypes are listed in Table 2 and described in subsequent sections. All are defined in the ogsi namespace.Thus, the task for the designer of OGSI-compliant components is to design portTypes that extend a combination of the GridService portType, the other optional portTypes listed in Table 2, and application or domain specific portTypes.

Table 2 Summary of the portTypes defined in this document

portType Name	Section	Description
GridService	§9	encapsulates the root behavior of the service model
HandleResolver	§10	mapping from a GSH to a GSR
NotificationSource	§11.1	allows clients to subscribe to notification messages
NotificationSubscription	§11.2	defines the relationship between a single NotificationSource and NotificationSink pair
NotificationSink	§11.3	defines a single operation for delivering a notification message to the service instance that implements the operation
Factory	§12	Is standard operation for creation of Grid service instances
ServiceGroup	§13	allows clients to maintain groups of services
ServiceGroupRegistration	§13.3	allows Grid services to be added and removed from a ServiceGroup
ServiceGroupEntry	§13.2	defines the relationship between a Grid service instance and its membership within a ServiceGroup

All of these OGSI portTypes are defined in the ogsi namespace.

9 GridService PortType

The GridService portType MUST be implemented (i.e., must be one of the portTypes associated with a Grid service description) 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, one must choose between document-centric messaging patterns and remote procedure call (RPC). Designers of Grid service interfaces also face the document-centric vs. RPC choice. 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: Service Data Declarations

The GridService portType includes the following serviceData elements:

- interface

  The QNames of all the portType(s) within the service instance's complete interface definition. This set MUST contain the transitive closure of the portType QNames implemented by the service instance.

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

- serviceDataName

  The QNames for each service data element supported by this service instance. This set MUST contain the QNames of all service data elements declared in the WSDL definition or the portType. This set MAY also contain the QNames of service data elements added dynamically by the instance.

```
<sd:serviceData name="serviceDataName"
     type="xsd:QName"
     minOccurs="0" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

- factoryLocator

  A service locator to the factory that created this Grid service instance. If the instance was not created by a factory, this value MUST be xsi:nil. This locator MUST refer to the

same factory service throughout the lifetime of this Grid service instance, although the handles and references contained in this locator MAY change during the service instance's lifetime.

```
<sd:serviceData name="factoryLocator"
     type="ogsi:LocatorType"
     minOccurs="1" maxOccurs="1"
     mutability="mutable"
     modifiable="false"
     nillable="true"/>
```

- gridServiceHandle

  Zero or more Grid Service Handles of this Grid service instance. There MAY be handles to this service instance that are not included in this set.

```
<sd:serviceData name="gridServiceHandle"
     type="ogsi:HandleType"
     minOccurs="0" maxOccurs="unbounded"
     mutability="extendable"
     modifiable="false"
     nillable="false"/>
```

- gridServiceReference

  One or more 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. There MAY be references to this service instance that are not included in this set.

```
<sd:serviceData name="gridServiceReference"
     type="ogsi:ReferenceType"
     minOccurs="1" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

- findServiceDataExtensibility

  A set of operation extensibility declarations (§7.8) for the findServiceData operation. Any conforming inputElement declared by values of this SDE 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="findServiceDataExtensibility"
     type="ogsi:OperationExtensibilityType"
     minOccurs="1" maxOccurs="unbounded"
     mutability="static"
     modifiable="false"
     nillable="false"/>
```

- setServiceDataExtensibility

  A set of operation extensibility declarations (7.8) for the setServiceData operation. Any conforming inputElement declared by values of this SDE 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="setServiceDataExtensibility"
     type="ogsi:OperationExtensibilityType"
     minOccurs="2" maxOccurs="unbounded"
     mutability="static"
     modifiable="false"
     nillable="false"/>
```

- terminationTime

  The termination time for this service instance (see §7.6).

```
<sd:serviceData name="terminationTime"
     type="ogsi:TerminationTimeType"
     minOccurs="1" maxOccurs="1"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

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

```
<sd:staticServiceDataValues>

   <ogsi:findServiceDataExtensibility
         inputElement="ogsi:queryByServiceDataNames" />

   <ogsi:setServiceDataExtensibility
         inputElement="ogsi:setByServiceDataNames" />

   <ogsi:setServiceDataExtensibility>
         inputElement="ogsi:deleteByServiceDataNames" />

</sd:staticServiceDataValues>
```

9.2 GridService: Operations

9.2.1 GridService :: findServiceData

Query the service data.

Input

- *QueryExpression*: The query to be performed. This extensible parameter MUST conform to an inputElement declaration denoted by one of the findServiceDataExtensibility SDE values. The service instance 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)

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the QueryExpression because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the QueryExpression violates that value's type.
- *TargetInvalidFault*: One or more of the SDEs that the QueryExpression requires do not exist in this service instance.
- *Fault*: Any other fault.

Every Grid service instance MUST support *QueryExpressions* conforming to queryByServiceDataNames 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 findServiceDataExtensibility SDE values. Therefore, a client can discover the query expression types supported by a service instance by performing a findServiceData request on the instance, using the queryByServiceDataNames element with a name of "ogsi:findServiceDataExtensibility".

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 queryByServiceDataNames

A queryByServiceDataNames results in a serviceDataValues element containing the service data elements named in the queryByServiceDataNames parameter. The names listed in the queryByServiceDataNames MUST be contained in the serviceDataName service data element (see §9.1) contained in this service instance.

The queryByServiceDataNames element is defined as follows:

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="queryByServiceDataNames" type="ogsi:QNamesType"/>

<xsd:complexType name="QNamesType">
   <xsd:sequence>
      <xsd:element name="name" type="QName"
                   minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

The findServiceData operation's *Result* output parameter for a queryByServiceDataNames query MUST be a sd:serviceDataValues element containing the service data elements listed in the queryByServiceDataNames.

For example, a findServiceData invocation with this QueryExpression:

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

might return this Result:

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

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

9.2.2 GridService :: setServiceData

The setServiceData operation allows for the modification of a service data element's values, if its service data declaration specifies modifiable="true". Changing a modifiable service data element implies changing the corresponding state in the underlying service instance. If no service data elements have a modifiable="true" attribute, then setServiceData is essentially disabled.

Input

- *UpdateExpression*: The update to be performed. This extensible parameter MUST conform to an inputElement declaration denoted by one of the setServiceDataExtensibility SDE values. The service instance 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 depends on the UpdateExpression.

Fault(s)

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the UpdateExpression because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the UpdateExpression violates that value's type.
- *CardinalityViolationFault:* The operation requested would violate the "minOccurs" and/or "maxOccurs" attributes of the service's SDE(s).
- *MutabilityViolationFault:* The UpdateExpression was not consistent with the "mutability" attribute of the service's SDE(s).
- *ModifiabilityViolationFault:* The UpdateExpression was not consistent with the "modifiable" attribute of the services's SDE(s).
- *TypeViolationFault:* The UpdateExpression contains values that do not conform to the XSD type of the service's SDE(s).
- *IncorrectValueFault:* The UpdateExpression contains values that are XSD type conformant, but are not acceptable to the service instance for other reasons.
- *PartialFailureFault:* The service instance was unable to satisfy all portions of the UpdateExpression. This fault extends FaultType with an element that contains a list of qnames of SDEs from the UpdateExpression that could not be updated. This fault MAY

have 1 or more "faultcause" elements that describe in more detail the portions that failed, using any of the fault types allowed by setServiceData.

- *Fault*: Any other fault.

Every Grid service instance MUST support *UpdateExpressions* conforming to setByServiceDataNames and deleteByServiceDataNames as defined in §9.2.2.1 and §9.2.2.2, respectively. 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 setServiceDataExtensibility SDE values. Thus, a client can discover the update expression types supported by a service instance by performing a findServiceData request on the instance, using the queryByServiceDataNames element with a name of "ogsi:setServiceDataExtensibility".

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 setByServiceDataNames

A setByServiceDataNames results in the update of the SDE values in the serviceDataValues containing the service data elements listed in the setByServiceDataNames 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. The service instance is responsible for ensuring that the SDE values are successfully updated in the underlying service's state. The service instance SHOULD update as many SDEs successfully as it can and return those that failed. The service instance MAY return after the first failure and not continue.

Partial failures (i.e., some SDEs are updated but some are not) MUST be indicated by returning a PartialFailureFault, with a list of the QNames of the SDEs that could not be set. This fault MAY also have one "faultcause" element for each SDE in the setByServiceDataNames expression that could not be set, where the type of the "faultcause" is any of the valid setServiceData faults, and the extensibility element in the "faultcause" is the SDE value from the setByServiceDataNames expression that could not be set.

A setByServiceDataNames MUST adhere to the mutability attribute of the SDEs as specified in §6.2.3. If the mutability value is "static" or "constant", then setByServiceDataNames is not allowed. If the mutability value is "extendable", then setByServiceDataNames MUST append to the new elements to the SDE's existing values. If the mutability value is "mutable", then setByServiceDataNames MUST replace the existing SDE values with the ones that are passed in. The setByServiceDataNames 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 as follows.

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

The setServiceData operation's *Result* output parameter for a setByServiceDataNames invocation MUST be a serviceDataValues element containing the service data elements listed in the setByServiceDataNames 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 is carried out with any coherency guarantee.

9.2.2.2 deleteByServiceDataNames

A deleteByServiceDataNames results in the deletion of all SDE values for the service data elements listed in the deleteByServiceDataNames 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 instance SHOULD delete as many SDEs successfully as it can and return those that failed. The service instance MAY return after the first failure and not continue.

Partial failures (i.e. some SDEs are deleted, but some are not) MUST be indicated by returning a PartialFailureFault, with a list of the QNames of the SDEs that could not be deleted. This fault MAY also have one "faultcause" element for each SDE in the deleteByServiceDataNames expression that could not be set, where the type of the "faultcause" is any of the valid deleteServiceData faults, and the extensibility element in the "faultcause" contains the QName that could not be deleted.

A deleteByServiceDataNames MUST adhere to the mutability attribute of the SDE as specified in §6.2.3. If the mutability value is "static", "constant", or "extendable", then deleteByServiceDataNames is not allowed. If the mutability value is "mutable", then deleteByServiceDataNames will delete all elements with the SDE names. The deleteByServiceDataNames 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 deleteByServiceDataNames element is defined as follows:

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="deleteByServiceDataNames" type="ogsi:QNamesType"/>
```

ogsi:QNamesType is defined in Section 9.2.1.1.

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

The client SHOULD NOT assume that the values in the service's serviceDataValues are in any way coherent with each other 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

The requestTerminationAfter operation requests that the termination time of the service instance be changed. The request specifies the earliest desired termination time. Upon receipt of the

request, the service instance 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:

- *TerminationTime*: The earliest termination time of the Grid service instance that is acceptable to the client. Its type is an ExtendedDateTimeType as defined in §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 instance continue to exist indefinitely.

Output:

- *CurrentTerminationTime*: An element of type TerminationTimeType (as defined in §7.6) that gives the service's currently planned termination time. The timestamp attribute of this CurrentTerminationTime MUST be the time at which the service instance handled this request.

Fault(s):

- *TerminationTimeUnchangedFault*: The service instance ignored the requested termination time.
- *Fault*: Any other fault.

9.2.4 GridService :: requestTerminationBefore

The requestTerminationBefore operation requests that the termination time of the service instance be changed. The request specifies the latest desired termination time. Upon receipt of the request, the service instance 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 instance that is acceptable to the client. Its type is an ExtendedDateTimeType as defined in §7.2. A time in the past indicates a desire that the service instance terminate as soon as possible. The special value "infinity" indicates that the client no longer cares about a maximum termination time.

Output:

- *CurrentTerminationTime*: An element of type TerminationTimeType (as defined in §7.6) that gives the service's currently planned termination time. The timestamp attribute of this CurrentTerminationTime MUST be the time at which the service instance handled this request.

Fault(s):

- *TerminationTimeUnchangedFault*: The service instance chose to ignore the requested termination time.
- *Fault*: Any other fault.

9.2.5 GridService :: destroy

The destroy operation explicitly requests destruction of the service instance. Upon receipt of an explicit destruction request, a Grid service instance MUST either (1) initiate its own destruction and return a response acknowledging the receipt of the destroy message or (2) ignore the request and return a fault message indicating failure. Once destruction of the Grid service instance is initiated, the service instance SHOULD NOT respond successfully to further requests. Following a successful destroy operation, the client MUST NOT rely on the existence of the service instance.

Input:

- None (an empty input message).

Output:

- None (an empty output message, acknowledging that the destroy has been initiated).

Fault(s):

- *ServiceNotDestroyedFault:* The service instance did not initiate self-destruction.
- *Fault*: Any other fault.

10 HandleResolver PortType

The HandleResolver portType defines a standard means for resolving a GSH to a GSR, independent of any particular URI scheme of the GSH. A service instance that implements the HandleResolver portType is called a *handle resolver.*

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 automatically add and remove mappings by 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 gridServiceHandle and gridServiceReference 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.

This portType extends the GridService portType.

10.1 HandleResolver: Service Data Declarations

The HandleResolver portType includes the following serviceData element.

- **handleResolverScheme**

  A set of URIs that specifies the GSH schemes that this handle resolver service instance MAY be able to resolve to a GSR. The only relevant portion of a handleResolverScheme URI is the scheme portion of the URI (i.e., the part preceeding the ":"). For example, a handleResolverScheme URI of "abc:def" means that this handle resolver MAY be able to resolve a GSH with a scheme of "abc". An xsi:nil handleResolverScheme value means that this resolver MAY be able to resolve any GSH with any scheme.

```
<sd:serviceData name="handleResolverScheme"
     type="xsd:anyURI"
     minOccurs="1" maxOccurs="unbounded"
```

```
        mutability="mutable"
        modifiable="false"
        nillable="true"/>
```

10.2 *HandleResolver: Operations*

10.2.1 HandleResolver :: findByHandle

The findByHandle operation returns a locator containing one or more Grid Service References for the Grid Service Handles in the HandleSet locator.

Input

- *HandleSet*: A locator containing one or more Grid Service Handles, which necessarily (by the definition of locator) refer to the same Grid service instance. The resolver MAY base its resolution on any GSH in this HandleSet, and MAY base its resolution on multiple GSHs in this HandleSet. The HandleSet MAY include any number of GSRs, which the handle resolver MAY ignore, or which the handle resolver MAY consider previously valid resolutions of the GSHs.
- *GSRExclusionSet*: (optional) A locator containing one or more GSRs that the client already possesses that are not satisfactory for some reason. These GSRs SHOULD NOT be returned in the output Locator. The GSRExclusionSet MAY contain any number of GSHs, but they MUST be ignored by the resolver when making its resolution decisions.

Output

- *Locator*: A service locator containing one or more GSRs for the input HandleSet. The locator MUST also contain all GSHs in the input HandleSet and MAY include additional GSHs to the same Grid service instance.

Fault(s)

- *InvalidHandleFault*: The handle violates the syntax of its URI scheme.
- *NoAdditionalReferencesAvailableFault:* The resolver cannot return a GSR that is not already contained in the GSRExclusionSet input parameter.
- *NoReferencesAvailableFault:* The resolver is unable to return a GSR for the input handle(s), regardless of the GSRExclusionSet input parameter. The following faults extend the NoReferencesAvailable fault:
    - *NoSuchServiceFault*: Either there was never a service instance with this handle, or the service instance wit h this handle has terminated. This fault MAY be applicable to only some URI schemes.
    - *NoSuchServiceStartedFault*: There was never a service instance with this handle. This fault MAY be applicable to only some URI schemes.
    - *ServiceHasTerminatedFault*: The service instance with this handle has terminated. This fault MAY be applicable to only some URI schemes.
    - *TemporarilyUnavailableFault*: The handle refers to a valid service instance, but 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 instance MAY be available. This fault MAY only be applicable to some URI schemes.

- *RedirectionFault*: An alternate handle resolver exists to which the client MAY direct the request. This fault extends FaultType with an additional element containing the locator of the alternate handle resolver.
- *Fault*: Any other fault.

11 Notification

The purpose of notification is to deliver interesting messages from a notification source to a notification sink, as described in the following.

- 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 MUST implement the NotificationSink portType**,** which allows it to receive notification messages
- 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 serviceDataValues.
- 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 locator of the notification sink to which notification messages are to be sent, and an initial lifetime for the subscription.
- A subscription request causes the creation of a Grid service instance, called a *subscription*, that 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 both direct service-to-service notification message delivery and the integration of various intermediary delivery services. Intermediary delivery services might include: messaging services, message filtering services, and message archival and replay services.

The treatment of subscriptions as Grid service instances allows them to be managed using the same interfaces as other Grid services. An OGSI implementation might well be expected to use specialized, more lightweight implementation techniques for subscriptions than for other Grid service instances.

11.1 NotificationSource PortType

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

The NotificationSource portType extends the GridService portType.

11.1.1 NotificationSource: Service Data Declarations

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.

```
<sd:serviceData name="notifiableServiceDataName"
     type="xsd:QName"
     minOccurs="0" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

- subscribeExtensibility

  A set of operation extensibility declarations (§7.8) for the subscribe operation. Any conforming inputElement declared by values of this SDE MAY be used by the client as a SubscriptionExpression parameter to the instance's subscribe operation, and implies the subscription semantics and notification message that result from the subscription.

```
<sd:serviceData name="subscribeExtensibility"
     type="ogsi:OperationExtensibilityType"
     minOccurs="1" maxOccurs="unbounded"
     mutability="static"
     modifiable="false"
     nillable="false"/>
```

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

```
<sd:staticServiceDataValues>
   <ogsi:subscribeExtensibility
         inputElement="ogsi:subscribeByServiceDataNames" />
</sd:staticServiceDataValues>
```

11.1.2 NotificationSource: Operations

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 extensible parameter MUST conform to an inputElement declaration denoted by one of the subscribeExtensibility SDE values. The service instance infers what to do based on the tag of the root element of this argument.
- *Sink*: The locator of the notification sink to which messages will be delivered. This locator MAY be to some other service instance than the one issuing this subscription request, thus allowing for third-party subscriptions. This locator MAY contain only references, thus allowing it to refer, for example, to a Web service that implements the NotificationSink portType but that does not have a handle.

- *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 locator to the subscription instance that was created to manage this subscription. This subscription instance MUST implement the NotificationSubscription portType.
- *CurrentTerminationTime*: An element of type TerminationTimeType (as defined in §7.6) that gives the NotificationSubscription service's currently planned termination time. The timestamp attribute of this CurrentTerminationTime MUST be the time at which the NotificationSubscription service instance was created.

Fault(s):

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the SubscriptionExpression because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the SubscriptionExpression violates that value's type.
- *TargetInvalidFault*: One or more of the SDEs that the SubscriptionExpression requires does not exist in this service instance.
- *Fault*: Any other fault.

Every Grid service instance that implements the NotificationSource portType MUST support a *subscribeExtensibility* SDE value of subscribeByServiceDataNames as defined in §11.1.2.1.1. A Grid service instance MAY support other *subscribeExtensibility* SDE values.

The list of subscription expression types supported by a Grid service instance is expressed in the instance's subscribeExtensibility SDE values. Therefore, a client can discover the subscription expression types supported by a service instance by performing a findServiceData request on the instance, using a queryByServiceDataNames element, which contains the name "ogsi:subscribeExtensibility".

11.1.2.1.1 subscribeByServiceDataNames

A subscribeByServiceDataNames results in notification messages being sent whenever any of the named service data elements change.

The subscribeByServiceDataNames element is defined as follows:

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="subscribeByServiceDataNames"
         type="ogsi:SubscribeByNameType"/>

<xsd:complexType name="SubscribeByNameType">
   <xsd:complexContent>
      <xsd:extension base="ogsi:QNamesType">
         <xsd:attribute name="minInterval"
                        type="duration"
                        use="optional"/>
         <xsd:attribute name="maxInterval"
                        type="ogsi:MaxIntervalType"/>
      </xsd :extension>
```

```
   </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="MaxIntervalType">
   <xsd:union memberTypes="ogsi:InfinityType xsd:duration"/>
</xsd :simpleType>
```

ogsi:QNamesType is defined in §9.2.1.1. ogsi:InfinityType is defined in §7.6.

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 elements' values, then the source MUST resend the same values. When the value is "infinity", the source need never resend a service data values if they do not change. If this property is not specified, then the notification source MAY choose this value.

For a subscribeByServiceDataNames 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 all SDE values corresponding to each requested serviceDataName, even if only some of the elements' values changed since the last message.

11.2 NotificationSubscription PortType

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

The NotificationSubscriptio n portType extends the GridService portType.

11.2.1 NotificationSubscription: Service Data Declarations

The NotificationSubscription portType includes the following serviceData elements.

- subscriptionExpression

  The current subscription expression managed by this subscription instance.

```
   <sd:serviceData name="subscriptionExpression"
        type="xsd:anyType"
        minOccurs="1" maxOccurs="1"
        mutability="mutable"
        modifiable="false"
        nillable="false"/>
```

- sinkLocator

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

```
   <sd:serviceData name="sinkLocator"
        type="ogsi:LocatorType"
        minOccurs="1" maxOccurs="1"
```

```
        mutability="mutable"
        modifiable="false"
        nillable="false"/>
```

11.2.2 NotificationSubscription: Operations

None.

11.3 NotificationSink PortType

The NotificationSink portType defines a single operation for delivering a notification message to the service instance that implements the operation.

Note: Unlike all the other portTypes described in this specification, a Web service implementing the NotificationSink portType is not required to be a Grid service instance. That is, the Web service is not required to also implement the GridService portType.

11.3.1 NotificationSink: Service Data Declarations

None.

11.3.2 NotificationSink: Operations

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.

This operation is input-only, so it does not return an output or faults.

12 Factory PortType

A *factory* is an abstract concept or pattern, corresponding to a Grid service instance that is used by a client to create another Grid service instance. A client invokes a create operation on a factory and receives as response a locator for the newly created service instance. A factory MAY be a Grid service instance that implements the document-centric Factory portType, or a Grid service instance that implements a more specialized factory operation (e.g., the NotificationSource::subscribe operation described in §11.1.2.1).

Upon creation by a factory, the newly created Grid service instance SHOULD be registered with, and receive a GSH from, a handle resolution service (see §10). The method by which this registration is accomplished is specific to the hosting environment, and is therefore outside the scope of this specification.

The Factory portType MUST extend the GridService portType.

12.1 Factory: Service Data Declarations

The Factory portType includes the following serviceData elements.

- createServiceExtensibility

  A set of operation extensibility declarations (§7.8) for the createService operation. Any conforming inputElement declared by values of this SDE MAY be used by the client as a

CreationParameters parameter to the instance's createService operation, and implies the creation semantics and return values that may result from the creation.

```
<sd:serviceData name="createServiceExtensibility"
     type="ogsi:CreateServiceExtensibilityType"
     minOccurs="1" maxOccurs="unbounded"
     mutability="static"
     modifiable="false"
     nillable="false"/>
```

ogsi:CreateServiceExtensibilityType is an extension of ogsi:OperationExtensibilityType that define additional elements containing the set of portTypes that are implemented by the service instance that is created as a result of this inputElement. This type is defined as:

```
<xsd:complexType name="CreateServiceExtensibilityType">
   <xsd:complexContent>
      <xsd:extension base="ogsi:OperationExtensibilityType">
         <xsd:sequence>
            <xsd:element name="createsInterface" type="xsd:QName"
                     minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

12.2 Factory: Operations

12.2.1 Factory :: createService

The factory createService creates a new Grid service instance. In order to support soft-state lifetime management (§7.6), a client may specify a window of earliest and latest acceptable initial termination times. The factory selects an initial termination time within this window and returns this time to the client as part of its response to the creation request. The factory also 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. These values are represented as a terminationTime element as defined in §7.6.
- *CreationParameters* (optional): A factory-specific element containing information necessary for the creation of the service instance. This extensible parameter MUST conform to an inputElement declaration denoted by one of the createServivceExtensibility SDE values. Note: The service instance infers what to do based on the tag of the root element of this argument.

Output

- *Locator*: A locator (see §7.5.3) to the newly created Grid service instance.
- *CurrentTerminationTime*: An element of type TerminationTimeType (as defined in §7.6) that gives the newly created service's currently planned termination time. The timestamp

attribute of this CurrentTerminationTime MUST be the time at which the new service instance was created.

- *ExtensibilityOutput* (optional): An XML extensibility element that is specific to the factory and the services that it creates.

Fault(s):

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the CreationParameters because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the CreationParameters violates that value's type.
- *ServiceAlreadyExistsFault*: The requested service instance already exists. This fault extends FaultType with an element that is a locator to the existing service instance.
- *Fault*: Any other fault

13 ServiceGroup

A ServiceGroup is a Grid service instance 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.

Three portTypes provide the interface to service groups: ServiceGroup, ServiceGroupEntry, and ServiceGroupRegistration.

13.1 ServiceGroup portType

The ServiceGroup portType provides an interface for representing a *service group* comprising zero or more *member* services. The Entry SDE of a ServiceGroup contains an element for each member Grid service instance in the group. The ServiceGroupEntryLocator element of the Entry SDE SHOULD refer to a service instance that implements the ServiceGroupEntry portType (see §13.2), which provides management functions for that entry. In particular, it provides independent lifetime management functions for individual entries, and a unique key (GSH) for each entry, and it can be extended to provide more advanced entry management functions.

The ServiceGroup portType extends the GridService portType.

The following properties hold for ServiceGroup, ServiceGroupEntry, ServiceGroupRegistration, and the member Grid service instances of the ServiceGroup.

- A Grid service instance MAY be a member of several ServiceGroups.
- Member Grid service instances of a ServiceGroup MAY implement different portTypes.
- A ServiceGroupEntry MAY be removed from a ServiceGroup by managing the lifetime of the ServiceGroupEntry.
- Once a ServiceGroup is destroyed, the client has no responsibility for the destruction of the ServiceGroupEntry services.
- Once a ServiceGroup is destroyed, the client can make no assumptions about the existence of the ServiceGroupEntry services or the validity of their contents (e.g., lifetime properties).

- A ServiceGroupEntry MUST belong to at most one ServiceGroup.
- If a GridService included in a ServiceGroup terminates, the ServiceGroup need not reflect this.
- All of the member Grid service instances of a ServiceGroup MUST conform to (i.e. be a, or subtype of,) at least one of the portTypes listed in the membershipContentRule SDE of the ServiceGroup.
- A member Grid service instance MAY be included in a ServiceGroup multiple times. The ServiceGroup is a "bag" of entries. Designers of portTypes that extend ServiceGroup MAY extend the semantics to a "set" or other forms of collection.

13.1.1 ServiceGroup: Service Data Declarations

- membershipContentRule

  This SDE contains a structure that associates a portType (memberInterface) with a set of XSD element QNames (content). Each service instance that is a member of a ServiceGroup MUST implement one or more of the memberInterfaces listed in the ServiceGroup's membershipContentRule SDE values. A ServiceGroup entry MUST include all content elements associated with any memberInterfaces that the entry's member service implements.

  This SDE serves as a "data type invariant" for the entry SDE values of the ServiceGroup and for the content SDE value of the ServiceGroupEntry for services in the ServiceGroup. That is, membership is restricted to Grid services that satisfy or conform to the restrictions specified in the membershipContentRule SDE values. If more than one rule applies to a Grid service instance (i.e., it implements several memberInterfaces included in the membershipContentRule SDE values), all rules MUST be satisfied. We note that implementations will need to be aware (either by introspection or other means) which interfaces are implemented by member Grid service instances in order to ensure that the membershipContentRule is satisfied.

```
<sd:serviceData
        name="membershipContentRule"
        type="MembershipContentRuleType"
        minOccurs="1"
        maxOccurs="unbounded"
        mutability="constant"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>

<xsd:complexType name="MembershipContentRuleType">
   <xsd:sequence>
      <xsd:element
           name="memberInterface"
           type="xsd:QName"
           minOccurs="1"
           maxOccurs="1"/>
      <xsd:element
           name="content"
           type="xsd:QName"
           minOccurs="0"
           maxOccurs="unbounded"/>
   </xsd:sequence>
```

```
</xsd:complexType>
```

- entry

  This element provides the structure of the constituent members of the ServiceGroup as service data. There is one entry SDE value for each entry in the ServiceGroup. Each SDE value is a triple consisting of: a locator that refers to the ServiceGroupEntry service instance that manages this entry; a locator that refers to the member Grid service instance referenced by this entry; and the content of the entry. The values in the entry SDE and those in the SDEs of the corresponding ServiceGroupEntry MUST conform to the membershipContentRule, MAY be incoherent (with respect to each other), but SHOULD converge in the absence of changes.

```
<sd:serviceData
        name="entry"
        type="ogsi:EntryType"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="mutable"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>
```

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:complexType name="EntryContentType">
   <xsd:sequence>
      <xsd:any namespace=”##any”
               minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EntryType">
   <xsd:sequence>
      <xsd:element name=”serviceGroupEntryLocator”
              type="ogsi:LocatorType"
              minOccurs="1" maxOccurs="1"
              nillable="true"/>
      <xsd:element name=”memberServiceLocator”
              type="ogsi:LocatorType"
              minOccurs="1" maxOccurs="1"/>
      <xsd:element name=”content”
              type="ogsi:EntryContentType"
              minOccurs="1" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
```

13.1.2 ServiceGroup: Operations

The ServiceGroup portType defines no operations. The operations inherited from the GridService portType SHOULD be used to manage the lifetime and to query the SDEs of the ServiceGroup.

13.2 ServiceGroupEntry portType

This portType defines the interface through which the individual entries in a ServiceGroup may be managed. Each ServiceGroupEntry service instance refers to a Grid service instance that is a member of the ServiceGroup. The GSH of a ServiceGroupEntry service instance MAY be used as the unique key for that entry. This key allows multiple references to the same service instance to be included in a ServiceGroup as separate entries, for example to record multiple properties of the service instance. Clients MAY use the findServiceData operation to query this information.

The ServiceGroupEntry portType extends the GridService portType.

13.2.1 ServiceGroupEntry: Service Data Declarations

- memberServiceLocator

  Contains a service locator to the member Grid service instance to which this entry pertains. This locator MUST refer to the same Grid service instance throughout the lifetime of this Grid service instance, although the handles and references contained in this locator MAY change during the service instance's lifetime.

```
<sd:serviceData
        name="memberServiceLocator"
        type="ogsi:LocatorType"
        minOccurs="1"
        maxOccurs="1"
        mutability="mutable"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>
```

- content

  This is an XML element advertising some information about the member service instance. The Content elements conform to the XSD element declarations listed (by QName) in the membershipContentRule SDE of the ServiceGroup portType containing this ServiceGroupEntry.

```
<sd:serviceData
        name="content"
        type="ogsi:EntryContentType"
        minOccurs="1"
        maxOccurs="1"
        mutability="mutable"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>
```

13.2.2 ServiceGroupEntry: Operations

The ServiceGroupEntry portType defines no operations. The lifetime management operations inherited from the GridService portType SHOULD be used to manage the lifetime of the entry service instance, such that destruction of the ServiceGroupEntry service instance SHOULD remove the member service instance from the ServiceGroup. The service data operations inherited from the GridService portType SHOULD be used to query the SDEs of the entry service instance, such as the content SDE.

13.3 ServiceGroupRegistration portType

The ServiceGroupRegistration portType provides a management interface (add and remove operations) for a ServiceGroup.

This portType extends the ServiceGroup portType.

13.3.1 ServiceGroupRegistration: Service Data Declarations

- addExtensibility

  A set of operation extensibility declarations (§7.8) for the add operation. Any conforming inputElement declared by values of this SDE MAY be used by the client as a Content parameter to the instance's add operation, and implies the add semantics and return values that may result from the add operation. These addExtensibility elements MAY be different than the elements listed in the membershipContentRule SDE of ServiceGroup. The add operation converts the value in the Content argument into the elements contained in the membershipContentRule.

```
<sd:serviceData
        name="addExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="static"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>
```

  OGSI defines one possible addExtensibility SDE value, ogsi:EntryContentType, that MAY be used to add content that is already structured in the same manner as the entry elements and ServiceGroupsEntries of the ServiceGroup.

- removeExtensibility

  A set of operation extensibility declarations (§7.8) for the remove operation. Any conforming inputElement declared by values of this SDE MAY be used by the client as a MatchExpression parameter to the instance's remove operation and implies the removal semantics and return values that may result from the remove operation. All match expressions MUST be Boolean valued functions that take a value of type "ogsi:EntryType" as an argument.

```
<sd:serviceData
        name="removeExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="1"
        maxOccurs="unbounded"
        mutability="static"
        modifiable="false"
        nillable=”false” >
</sd:serviceData>
```

This portType MUST define one standard removeExtensibility SDE value that MUST be supported by all ServiceGroupRegistration services. This matchByLocatorEquivalence match

expression compares one or more locators for textual equality with the canonical XML form of the memberServiceLocator of each entry contained by the service group. The XSD definition of this argument is as follows:

```
targetNamespace = “http://www.gridforum.org/namespaces/2003/03/OGSI”

<xsd:element name="matchByLocatorEquivalence"
             type="ogsi:MatchByLocatorEquivalenceType"/>

<xsd:complexType name="MatchByLocatorEquivalenceType">
   <xsd:sequence>
      <xsd:element name="locator" type="ogsi:LocatorType"
                   minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

This portType MUST define the static SDE value.

```
<sd:staticServiceDataValues>
   <ogsi:removeExtensibility
         inputElement=”ogsi:matchByLocatorEquivalence” />
</sd:staticServiceDataValues>
```

13.3.2 ServiceGroupRegistration:Operations

13.3.2.1 ServiceGroupRegistration :: add

The add operation creates a ServiceGroupEntry and adds it to the ServiceGroup. When an add operation fails, a new entry is not created. Other semantics of a failed add operation MAY be defined by a portType that extends ServiceGroupRegistration.

Input:

- *ServiceLocator*: A serviceLocator for the member Grid service instance to be included in the ServiceGroup. It MUST refer to a member service instance whose type conforms to one of the elements in the membershipContentRule.
- *Content*: Content to associate with the the ServiceLocator in the service group. This extensible parameter MUST conform to an inputElement declaration denoted by one of the addExtensibility SDE values. The service instance infers what to do based on the tag of the root element of this argument. The Content is processed according to service group specific semantics into an element conformant with one or more of the entries in the membershipContentRule SDE for a Grid service instance of the type given by the serviceLocator argument. The resulting transformed Content becomes the ServiceGroupEntry's Content SDE.
- *TerminationTime* (optional): The earliest and latest initial termination times of the created ServiceGroupEntry service instance that are acceptable to the client. These values are represented as a terminationTime element as defined in §7.6.

Output:

- *ServiceLocator*: A serviceLocator to the newly created ServiceGroupEntry.
- *CurrentTerminationTime*: An element of type TerminationTimeType (as defined in §7.6) that gives the ServiceGroupEntry service's currently planned termination time. The

timestamp attribute of this CurrentTerminationTime MUST be the time at which the ServiceGroupEntry service instance was created.

Faults:

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the Content because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the Content violates that value's type.
- *ContentCreationFailedFault*: The operation was unable to create a valid Content element (as defined by the membershipContentRule SDE) from the provided Content and serviceLocator arguments.
- *UnsupportedMemberInterfaceFault*: The type of the member service instance referred to by the ServiceLocator argument does not conform to the membershipContentRule.
- *AddRefusedFault*: The ServiceGroupRegistration refused to create a new entry for the member service instance based the semantics of the ServiceGroupRegistration (or subtype).
- *Fault*: All other faults.

13.3.2.2 ServiceGroupRegistration :: remove

The remove operation removes each ServiceGroupEntry from the ServiceGroup that matches an input expression.

Input:

- *MatchExpression*: This extensible parameter MUST conform to an inputElement declaration denoted by one of the removeExtensibility SDE values. The service instance infers what to do based on the tag of the root element of this argument. The MatchExpression is evaluated against all entries in the ServiceGroup. Each entry that matches is removed from the ServiceGroup. If a removed entry has a ServiceGroupEntry service instance, then a GridService::destroy operation is sent to that instance, though this remove operation MAY complete before all such destroy operations have completed. This operation has no effect on the member Grid services referred to by the entries.

Output:

- None, except acknowledgment of the operation completion.

Faults:

- *ExtensibilityNotSupportedFault:* The service instance cannot evaluate the MatchExpression because its type is not supported by this service instance.
- *ExtensibilityTypeFault*: The value passed as the MatchExpression violates that value's type.
- *MatchFailedFault*: No entry matched the MatchExpression.
- *RemoveFailedFault*: A match was found, but the remove failed for other reasons.
- *Fault*: All other faults.

14 Security Considerations

This specification defines the abstract interaction between a Grid service instance and clients of that service instance. While it is assumed that such interactions must be secured, the details of

security are out of scope of this specification. Instead, security should be addressed in related specifications that define how the abstract interactions are bound to specific communication protocols, how service behaviors are specialized via policy-management interfaces, and how security features are delivered in specific programming environments.

15 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
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
Email: karlcz@isi.edu

Ian Foster
Argonne National Laboratory & University of Chicago
Argonne, IL 60439
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
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
Email: carl@isi.edu

Tom Maquire
IBM
Poughkeepsie, NY 12601
Email: tmaguire@us.ibm.com

Thomas Sandholm
Argonne National Laboratory
Argonne, IL 60439
Email: sandholm@mcs.anl.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

Peter Vanderbilt
NASA Ames Research Center
Moffett Field, CA 94035-1000
Email: pv@nas.nasa.gov

16 Contributors

We gratefully acknowledge the contributions made to this specification by the following people:

Nick Butler, Donald Ferguson, Andrew Grimshaw, Shel Finkelstein, Frank Leymann, Martin Nally, Jeff Nick, John Rofrano, Ellen Stokes, Tony Storey, Jay Unger, Sanjiva Weerawarana

17 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, Marty Humphrey, Keith Jackson, Bill Johnston, Kate Keahey, Lee Liming, Miron Livny, Sastry Malladi, Savas Parastatidis, Norman Paton, Jean-Pierre Prost, Frank Siebenlist, Scott Sylvester, Gregor von Laszewski, Von Welch, and Mike Williams.

This work was supported in part by IBM; 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.

18 References

18.1 Normative References

[RFC 2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, RFC 2119, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

[WSDL 1.2]

Web Services Description Language (WSDL) Version 1.2, Published W3C Working Draft, World Wide Web Consortium. Available at http://www.w3.org/TR/wsdl12/

[WSDL 1.2 DRAFT]

Web Services Description Language (WSDL) Version 1.2, W3C Working Draft 3 March 2003, World Wide Web Consortium. Available at http://www.w3.org/TR/2003/WD-wsdl12-20030303

18.2 Informative References

[Globus Overview]

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

[Grid Anatomy]

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

[Grid Book]

The Grid: Blueprint for a New Computing Infrastructure, I. Foster, C. Kesselman, eds. Morgan Kaufmann, 1999.

[Grid Physiology]

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

[JAX-RPC]

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

[Web Services Book]

Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, s. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, R. Neyama. Sams, 2001.

[WSIF]

Welcome to WSIF: Web Services Invocation Framework, http://www.apache.org/wsif/

19 Normative XSD and WSDL Specifications

This section contains the full normative XSD and WSDL definitions for everything described in this document. The definitions in this section MUST be considered normative, if there are any discrepancies between the definitions in this section and those portions described in other sections above.

19.1 http://www.gridforum.org/namespaces/2003/03/OGSI

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="OGSI"

targetNamespace="http://www.gridforum.org/namespaces/2003/03/OGSI"

xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"

xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtens
ions"

xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
```

```
  <schema
targetNamespace="http://www.gridforum.org/namespaces/2003/03/OGSI"
          xmlns="http://www.w3.org/2001/XMLSchema"
          attributeFormDefault="qualified"
          elementFormDefault="qualified">
    <!-- Common Types -->
    <simpleType name="ExtendedDateTimeType">
      <union memberTypes="ogsi:InfinityType xsd:dateTime"/>
    </simpleType>

    <simpleType name="InfinityType">
      <restriction base="string">
        <enumeration value="infinity"/>
      </restriction>
    </simpleType>

    <attribute name="goodFrom" type="ogsi:ExtendedDateTimeType"/>
    <attribute name="goodUntil" type="ogsi:ExtendedDateTimeType"/>
    <attribute name="availableUntil"
        type="ogsi:ExtendedDateTimeType"/>

    <attributeGroup name="LifeTimePropertiesGroup">
      <attribute ref="ogsi:goodFrom" use="optional"/>
      <attribute ref="ogsi:goodUntil" use="optional"/>
      <attribute ref="ogsi:availableUntil" use="optional"/>
    </attributeGroup>

    <element name="reference" type="ogsi:ReferenceType"/>
    <complexType name="ReferenceType" abstract="true">
      <attribute ref="ogsi:goodFrom" use="optional"/>
      <attribute ref="ogsi:goodUntil" use="optional"/>
    </complexType>

    <!-- The content of this type MUST be a wsdl:definitions element
    with a single wsdl:service child element -->
    <complexType name="WSDLReferenceType">
      <complexContent>
        <extension base="ogsi:ReferenceType">
          <sequence>
            <any namespace="http://schemas.xmlsoap.org/wsdl/"
                  minOccurs="1" maxOccurs="1" processContents="lax"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="handle" type="ogsi:HandleType"/>
    <simpleType name="HandleType">
      <restriction base="anyURI"/>
    </simpleType>

    <element name="locator" type="ogsi:LocatorType"/>
    <complexType name="LocatorType">
      <sequence>
          <element ref="ogsi:handle" minOccurs="0"
              maxOccurs="unbounded"/>
          <element ref="ogsi:reference" minOccurs="0"
```

```
            maxOccurs="unbounded"/>
        <element name="interface" type="QName" minOccurs="0"
            maxOccurs="unbounded"/>
    </sequence>
  </complexType>


  <!-- Grid Service Types -->
  <complexType name="ExtensibilityType">
    <sequence>
      <any namespace="##any"/>
    </sequence>
  </complexType>

  <!-- Grid Service Service Data Types -->
  <complexType name="OperationExtensibilityType">
    <attribute name="inputElement" type="QName" use="optional"/>
  </complexType>

  <complexType name="TerminationTimeType">
    <attribute name="after" type="ogsi:ExtendedDateTimeType"
          use="optional"/>
    <attribute name="before" type="ogsi:ExtendedDateTimeType"
          use="optional"/>
    <attribute name="timestamp" type="dateTime" use="optional"/>
  </complexType>

  <element name="queryByServiceDataNames" type="ogsi:QNamesType"/>

  <element name="deleteByServiceDataNames" type="ogsi:QNamesType"/>

  <element name="setByServiceDataNames"
      type="ogsi:ExtensibilityType"/>

  <complexType name="QNamesType">
    <sequence>
      <element name="name" type="QName" minOccurs="0"
            maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <!-- Grid Service Message Types -->
  <element name="findServiceData">
    <complexType>
      <sequence>
        <element name="queryExpression"
              type="ogsi:ExtensibilityType"/>
      </sequence>
    </complexType>
  </element>
  <element name="findServiceDataResponse">
    <complexType>
      <sequence>
        <element name="result" type="ogsi:ExtensibilityType"/>
      </sequence>
    </complexType>
  </element>
```

```
    <element name="setServiceData">
      <complexType>
        <sequence>
          <element name="updateExpression"
                type="ogsi:ExtensibilityType"/>
        </sequence>
      </complexType>
    </element>
    <element name="setServiceDataResponse">
      <complexType>
        <sequence>
          <element name="result" type="ogsi:ExtensibilityType"/>
        </sequence>
      </complexType>
    </element>
    <element name="requestTerminationBefore">
      <complexType>
        <sequence>
          <element name="terminationTime"
                type="ogsi:ExtendedDateTimeType"/>
        </sequence>
      </complexType>
    </element>
    <element name="requestTerminationBeforeResponse">
      <complexType>
        <sequence>
          <element name="currentTerminationTime"
                type="ogsi:TerminationTimeType"/>
        </sequence>
      </complexType>
    </element>
    <element name="requestTerminationAfter">
      <complexType>
        <sequence>
          <element name="terminationTime"
                type="ogsi:ExtendedDateTimeType"/>
        </sequence>
      </complexType>
    </element>
    <element name="requestTerminationAfterResponse">
      <complexType>
        <sequence>
          <element name="currentTerminationTime"
                type="ogsi:TerminationTimeType"/>
        </sequence>
      </complexType>
    </element>
    <element name="destroy">
      <complexType/>
    </element>
    <element name="destroyResponse">
      <complexType/>
    </element>

    <!-- Grid Service Fault Types -->
    <element name="fault" type="ogsi:FaultType"/>
    <complexType name="FaultType">
```

```
      <sequence>
        <element name="description"
                 type="string"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
        <element name="originator"
                 type="ogsi:LocatorType"
                 minOccurs="1"
                 maxOccurs="1"/>
        <element name="timestamp"
                 type="dateTime"
                 minOccurs="1"
                 maxOccurs="1"/>
        <element name="faultcause"
                 type="ogsi:FaultType"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
        <element name="faultcode"
                 type="ogsi:FaultCodeType"
                 minOccurs="0"
                 maxOccurs="1"/>
        <element name="extension"
                 type="ogsi:ExtensibilityType"
                 minOccurs="0"
                 maxOccurs="1"/>
      </sequence>
    </complexType>
    <complexType name="FaultCodeType">
      <simpleContent>
        <extension base="string">
          <attribute name="faultscheme" type="anyURI" use="required"/>
        </extension>
      </simpleContent>
    </complexType>
    <element name="serviceNotDestroyedFault"
        type="ogsi:ServiceNotDestroyedFaultType"/>
    <complexType name="ServiceNotDestroyedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="extensibilityTypeFault"
        type="ogsi:ExtensibilityTypeFaultType"/>
    <complexType name="ExtensibilityTypeFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="extensibilityNotSupportedFault"
        type="ogsi:ExtensibilityNotSupportedFaultType"/>
    <complexType name="ExtensibilityNotSupportedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="targetInvalidFault"
        type="ogsi:TargetInvalidFaultType"/>
```

```
    <complexType name="TargetInvalidFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="cardinalityViolationFault"
        type="ogsi:CardinalityViolationFaultType"/>
    <complexType name="CardinalityViolationFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="mutabilityViolationFault"
        type="ogsi:MutabilityViolationFaultType"/>
    <complexType name="MutabilityViolationFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="modifiabilityViolationFault"
        type="ogsi:ModifiabilityViolationFaultType"/>
    <complexType name="ModifiabilityViolationFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="typeViolationFault"
        type="ogsi:TypeViolationFaultType"/>
    <complexType name="TypeViolationFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="incorrectValueFault"
        type="ogsi:IncorrectValueFaultType"/>
    <complexType name="IncorrectValueFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="partialFailureFault"
        type="ogsi:PartialFailureFaultType"/>
    <complexType name="PartialFailureFaultType">
      <complexContent>
        <extension base="ogsi:FaultType">
          <sequence>
            <element name="failedServiceData" type="ogsi:QNamesType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
    <element name="terminationTimeUnchangedFault"
        type="ogsi:TerminationTimeUnchangedFaultType"/>
    <complexType name="TerminationTimeUnchangedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
```

```
    </complexType>

    <!-- Handle Resolver Message Types -->
    <element name="findByHandle">
      <complexType>
        <sequence>
          <element name="handleSet" type="ogsi:LocatorType"/>
          <element name="gsrExclusionSet" type="ogsi:LocatorType"
              minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
    <element name="findByHandleResponse">
      <complexType>
        <sequence>
          <element ref="ogsi:locator"/>
        </sequence>
      </complexType>
    </element>

    <!-- Handle Resolver Fault Types -->
    <element name="invalidHandleFault"
        type="ogsi:InvalidHandleFaultType"/>
    <complexType name="InvalidHandleFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="noAdditionalReferencesAvailableFault"
        type="ogsi:NoAdditionalReferencesAvailableFaultType"/>
    <complexType name="NoAdditionalReferencesAvailableFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="noReferencesAvailableFault"
        type="ogsi:NoReferencesAvailableFaultType"/>
    <complexType name="NoReferencesAvailableFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="redirectionFault"
        type="ogsi:RedirectionFaultType"/>
    <complexType name="RedirectionFaultType">
      <complexContent>
        <extension base="ogsi:FaultType">
          <sequence>
            <element ref="ogsi:locator"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
    <element name="noSuchServiceFault"
        type="ogsi:NoSuchServiceFaultType"/>
    <complexType name="NoSuchServiceFaultType">
      <complexContent>
```

```
      <extension base="ogsi:NoReferencesAvailableFaultType"/>
    </complexContent>
  </complexType>
  <element name="noSuchServiceStartedFault"
      type="ogsi:NoSuchServiceStartedFaultType"/>
  <complexType name="NoSuchServiceStartedFaultType">
    <complexContent>
      <extension base="ogsi:NoReferencesAvailableFaultType"/>
    </complexContent>
  </complexType>
  <element name="serviceHasTerminatedFault"
      type="ogsi:ServiceHasTerminatedFaultType"/>
  <complexType name="ServiceHasTerminatedFaultType">
    <complexContent>
      <extension base="ogsi:NoReferencesAvailableFaultType"/>
    </complexContent>
  </complexType>
  <element name="temporarilyUnavailableFault"
      type="ogsi:TemporarilyUnavailableFaultType"/>
  <complexType name="TemporarilyUnavailableFaultType">
    <complexContent>
      <extension base="ogsi:NoReferencesAvailableFaultType">
        <sequence>
          <element name="available" type="dateTime"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Notification Source Service Data Types -->
  <element name="subscribeByServiceDataNames"
      type="ogsi:SubscribeByNameType"/>
  <complexType name="SubscribeByNameType">
    <complexContent>
      <extension base="ogsi:QNamesType">
        <attribute name="minInterval" type="duration"
            use="optional"/>
        <attribute name="maxInterval" type="ogsi:MaxIntervalType"/>
      </extension>
    </complexContent>
  </complexType>

  <simpleType name="MaxIntervalType">
    <union memberTypes="ogsi:InfinityType xsd:duration"/>
  </simpleType>

  <!-- Notification Source Message Types -->
  <element name="subscribe">
    <complexType>
      <sequence>
        <element name="subscriptionExpression"
            type="ogsi:ExtensibilityType"/>
        <element name="sink" type="ogsi:LocatorType"/>
        <element name="expirationTime"
            type="ogsi:ExtendedDateTimeType"/>
      </sequence>
    </complexType>
```

```
    </element>
    <element name="subscribeResponse">
      <complexType>
        <sequence>
          <element name="subscriptionInstanceLocator"
              type="ogsi:LocatorType"/>
          <element name="currentTerminationTime"
              type="ogsi:TerminationTimeType"/>
        </sequence>
      </complexType>
    </element>

    <!-- Notification Sink Message Types -->
    <element name="deliverNotification">
      <complexType>
        <sequence>
          <element name="message" type="ogsi:ExtensibilityType"/>
        </sequence>
      </complexType>
    </element>

    <!-- Factory Service Data Types -->
    <complexType name="CreateServiceExtensibilityType">
      <complexContent>
        <extension base="ogsi:OperationExtensibilityType">
          <sequence>
            <element name="createsInterface" type="QName"
                minOccurs="1" maxOccurs="unbounded"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <!-- Factory Message Types -->
    <element name="createService">
      <complexType>
        <sequence>
          <element name="terminationTime"
              type="ogsi:TerminationTimeType" minOccurs="0"
              maxOccurs="1"/>
          <element name="creationParameters"
              type="ogsi:ExtensibilityType" minOccurs="0"
              maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
    <element name="createServiceResponse">
      <complexType>
        <sequence>
          <element name="locator" type="ogsi:LocatorType"/>
          <element name="currentTerminationTime"
              type="ogsi:TerminationTimeType"/>
          <element name="extensibilityOutput"
              type="ogsi:ExtensibilityType" minOccurs="0"
              maxOccurs="1"/>
        </sequence>
      </complexType>
```

```
    </element>

    <!-- Factory Fault Types -->
    <element name="serviceAlreadyExistsFault"
        type="ogsi:ServiceAlreadyExistsFaultType"/>
    <complexType name="ServiceAlreadyExistsFaultType">
      <complexContent>
        <extension base="ogsi:FaultType">
          <sequence>
            <element name="existingService" type="ogsi:LocatorType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <!-- Service Group Service Data Types -->
    <complexType name="MembershipContentRuleType">
      <sequence>
        <element name="memberInterface"
                 type="QName"
                 minOccurs="1"
                 maxOccurs="1"/>
        <element name="content"
                 type="QName"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <complexType name="EntryContentType">
      <sequence>
        <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>

    <complexType name="EntryType">
      <sequence>
        <element name="serviceGroupEntryLocator"
                 type="ogsi:LocatorType"
                 minOccurs="1"
                 maxOccurs="1"
                 nillable="true"/>
        <element name="memberServiceLocator"
                 type="ogsi:LocatorType"
                 minOccurs="1"
                 maxOccurs="1"/>
        <element name="content"
            type="ogsi:EntryContentType"
                 minOccurs="1"
                 maxOccurs="1"/>
      </sequence>
    </complexType>


    <!-- Service Group Registration Service Data Types -->
    <element name="matchByLocatorEquivalence"
        type="ogsi:MatchByLocatorEquivalenceType"/>
    <complexType name="MatchByLocatorEquivalenceType">
```

```
      <sequence>
        <element name="locator" type="ogsi:LocatorType" minOccurs="0"
            maxOccurs="unbounded"/>
      </sequence>
    </complexType>

    <!-- Service Group Registration Message Types -->
    <element name="add">
      <complexType>
        <sequence>
          <element name="serviceLocator" type="ogsi:LocatorType"/>
          <element name="content" type="ogsi:ExtensibilityType"/>
          <element name="terminationTime"
              type="ogsi:TerminationTimeType" minOccurs="0"
              maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
    <element name="addResponse">
      <complexType>
        <sequence>
          <element name="serviceLocator" type="ogsi:LocatorType"/>
          <element name="currentTerminationTime"
              type="ogsi:TerminationTimeType"/>
        </sequence>
      </complexType>
    </element>
    <element name="remove">
      <complexType>
        <sequence>
          <element name="matchExpression"
              type="ogsi:ExtensibilityType"/>
        </sequence>
      </complexType>
    </element>
    <element name="removeResponse">
      <complexType/>
    </element>

    <!-- Service Group Registration Fault Types -->
    <element name="contentCreationFailedFault"
        type="ogsi:ContentCreationFailedFaultType"/>
    <complexType name="ContentCreationFailedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="unsupportedMemberInterfaceFault"
        type="ogsi:UnsupportedMemberInterfaceFaultType"/>
    <complexType name="UnsupportedMemberInterfaceFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="addRefusedFault" type="ogsi:AddRefusedFaultType"/>
    <complexType name="AddRefusedFaultType">
      <complexContent>
```

```
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="matchFailedFault"
        type="ogsi:MatchFailedFaultType"/>
    <complexType name="MatchFailedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
    <element name="removeFailedFault"
        type="ogsi:RemoveFailedFaultType"/>
    <complexType name="RemoveFailedFaultType">
      <complexContent>
        <extension base="ogsi:FaultType"/>
      </complexContent>
    </complexType>
  </schema>
</types>

<!-- Grid Service Messages -->
<message name="FindServiceDataInputMessage">
  <part name="parameters" element="ogsi:findServiceData"/>
</message>
<message name="FindServiceDataOutputMessage">
  <part name="parameters" element="ogsi:findServiceDataResponse"/>
</message>
<message name="SetServiceDataInputMessage">
  <part name="parameters" element="ogsi:setServiceData"/>
</message>
<message name="SetServiceDataOutputMessage">
  <part name="parameters" element="ogsi:setServiceDataResponse"/>
</message>
<message name="RequestTerminationBeforeInputMessage">
  <part name="parameters" element="ogsi:requestTerminationBefore"/>
</message>
<message name="RequestTerminationBeforeOutputMessage">
  <part name="parameters"
      element="ogsi:requestTerminationBeforeResponse"/>
</message>
<message name="RequestTerminationAfterInputMessage">
  <part name="parameters" element="ogsi:requestTerminationAfter"/>
</message>
<message name="RequestTerminationAfterOutputMessage">
  <part name="parameters"
      element="ogsi:requestTerminationAfterResponse"/>
</message>
<message name="DestroyInputMessage">
  <part name="parameters" element="ogsi:destroy"/>
</message>
<message name="DestroyOutputMessage">
  <part name="parameters" element="ogsi:destroyResponse"/>
</message>

<!-- Grid Service Fault Messages -->
<message name="FaultMessage">
  <part name="fault" element="ogsi:fault"/>
```

```
</message>
<message name="ServiceNotDestroyedFaultMessage">
  <part name="fault" element="ogsi:serviceNotDestroyedFault"/>
</message>
<message name="ExtensibilityTypeFaultMessage">
  <part name="fault" element="ogsi:extensibilityTypeFault"/>
</message>
<message name="ExtensibilityNotSupportedFaultMessage">
  <part name="fault" element="ogsi:extensibilityNotSupportedFault"/>
</message>
<message name="TargetInvalidFaultMessage">
  <part name="fault" element="ogsi:targetInvalidFault"/>
</message>
<message name="CardinalityViolationFaultMessage">
  <part name="fault" element="ogsi:cardinalityViolationFault"/>
</message>
<message name="MutabilityViolationFaultMessage">
  <part name="fault" element="ogsi:mutabilityViolationFault"/>
</message>
<message name="ModifiabilityViolationFaultMessage">
  <part name="fault" element="ogsi:modifiabilityViolationFault"/>
</message>
<message name="TypeViolationFaultMessage">
  <part name="fault" element="ogsi:typeViolationFault"/>
</message>
<message name="IncorrectValueFaultMessage">
  <part name="fault" element="ogsi:incorrectValueFault"/>
</message>
<message name="PartialFailureFaultMessage">
  <part name="fault" element="ogsi:partialFailureFault"/>
</message>
<message name="TerminationTimeUnchangedFaultMessage">
  <part name="fault" element="ogsi:terminationTimeUnchangedFault"/>
</message>

<!-- HandleResolver Messages -->
<message name="FindByHandleInputMessage">
  <part name="parameters" element="ogsi:findByHandle"/>
</message>
<message name="FindByHandleOutputMessage">
  <part name="parameters" element="ogsi:findByHandleResponse"/>
</message>

<!-- HandleResolver Fault Messages -->
<message name="InvalidHandleFaultMessage">
  <part name="fault" element="ogsi:invalidHandleFault"/>
</message>
<message name="NoAdditionalReferencesAvailableFaultMessage">
  <part name="fault"
      element="ogsi:noAdditionalReferencesAvailableFault"/>
</message>
<message name="NoReferencesAvailableFaultMessage">
  <part name="fault" element="ogsi:noReferencesAvailableFault"/>
</message>
<message name="RedirectionFaultMessage">
  <part name="fault" element="ogsi:redirectionFault"/>
</message>
```

```
<message name="NoSuchServiceFaultMessage">
  <part name="fault" element="ogsi:noSuchServiceFault"/>
</message>
<message name="NoSuchServiceStartedFaultMessage">
  <part name="fault" element="ogsi:noSuchServiceStartedFault"/>
</message>
<message name="ServiceHasTerminatedFaultMessage">
  <part name="fault" element="ogsi:serviceHasTerminatedFault"/>
</message>
<message name="TemporarilyUnavailableFaultMessage">
  <part name="fault" element="ogsi:temporarilyUnavailableFault"/>
</message>

<!-- Factory Messages -->
<message name="CreateServiceInputMessage">
  <part name="parameters" element="ogsi:createService"/>
</message>
<message name="CreateServiceOutputMessage">
  <part name="parameters" element="ogsi:createServiceResponse"/>
</message>

<!-- Factory Fault Messages -->
<message name="ServiceAlreadyExistsFaultMessage">
  <part name="fault" element="ogsi:serviceAlreadyExistsFault"/>
</message>

<!-- NotificaitonSource Messages -->
<message name="SubscribeInputMessage">
  <part name="parameters" element="ogsi:subscribe"/>
</message>
<message name="SubscribeOutputMessage">
  <part name="parameters" element="ogsi:subscribeResponse"/>
</message>

<!-- NotificationSink Messages -->
<message name="DeliverNotificationInputMessage">
  <part name="parameters" element="ogsi:deliverNotification"/>
</message>

<!-- ServiceGroupRegistration Messages -->
<message name="AddInputMessage">
  <part name="parameters" element="ogsi:add"/>
</message>
<message name="AddOutputMessage">
  <part name="parameters" element="ogsi:addResponse"/>
</message>
<message name="removeInputMessage">
  <part name="parameters" element="ogsi:remove"/>
</message>
<message name="removeOutputMessage">
  <part name="parameters" element="ogsi:removeResponse"/>
</message>

<!-- ServiceGroupRegistration Fault Messages -->
<message name="ContentCreationFailedFaultMessage">
  <part name="faults" element="ogsi:contentCreationFailedFault"/>
</message>
```

```
<message name="UnsupportedMemberInterfaceFaultMessage">
  <part name="faults" element="ogsi:unsupportedMemberInterfaceFault"/>
</message>
<message name="AddRefusedFaultMessage">
  <part name="faults" element="ogsi:addRefusedFault"/>
</message>
<message name="MatchFailedFaultMessage">
  <part name="faults" element="ogsi:matchFailedFault"/>
</message>
<message name="RemoveFailedFaultMessage">
  <part name="faults" element="ogsi:removeFailedFault"/>
</message>

<!-- Grid Service Port Type -->
<gwsdl:portType name="GridService">
  <operation name="findServiceData">
    <input message="ogsi:FindServiceDataInputMessage"/>
    <output message="ogsi:FindServiceDataOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="TargetInvalidFault"
        message="ogsi:TargetInvalidFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="setServiceData">
    <input message="ogsi:SetServiceDataInputMessage"/>
    <output message="ogsi:SetServiceDataOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="CadinalityViolationFault"
        message="ogsi:CardinalityViolationFaultMessage"/>
    <fault name="MutabilityViolationFault"
        message="ogsi:MutabilityViolationFaultMessage"/>
    <fault name="ModifabilityViolationFault"
        message="ogsi:ModifiabilityViolationFaultMessage"/>
    <fault name="TypeViolationFault"
        message="ogsi:TypeViolationFaultMessage"/>
    <fault name="IncorrectValueFault"
        message="ogsi:IncorrectValueFaultMessage"/>
    <fault name="PartialFailureFault"
        message="ogsi:PartialFailureFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="requestTerminationAfter">
    <input message="ogsi:RequestTerminationAfterInputMessage"/>
    <output message="ogsi:RequestTerminationAfterOutputMessage"/>
    <fault name="TerminationTimeUnchangedFault"
        message="ogsi:TerminationTimeUnchangedFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="requestTerminationBefore">
    <input message="ogsi:RequestTerminationBeforeInputMessage"/>
    <output message="ogsi:RequestTerminationBeforeOutputMessage"/>
```

```
    <fault name="TerminationTimeUnchangedFault"
        message="ogsi:TerminationTimeUnchangedFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="destroy">
    <input message="ogsi:DestroyInputMessage"/>
    <output message="ogsi:DestroyOutputMessage"/>
    <fault name="ServiceNotDestroyedFault"
        message="ogsi:ServiceNotDestroyedFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <sd:serviceData name="interface"
                  type="xsd:QName"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="constant"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="serviceDataName"
                  type="xsd:QName"
                  minOccurs="0"
                  maxOccurs="unbounded"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="factoryLocator"
                  type="ogsi:LocatorType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="true"/>
  <sd:serviceData name="gridServiceHandle"
                  type="ogsi:HandleType"
                  minOccurs="0"
                  maxOccurs="unbounded"
                  mutability="extendable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="gridServiceReference"
                  type="ogsi:ReferenceType"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="findServiceDataExtensibility"
                  type="ogsi:OperationExtensibilityType"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="static"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="setServiceDataExtensibility"
                  type="ogsi:OperationExtensibilityType"
                  minOccurs="1"
                  maxOccurs="unbounded"
```

```
                  mutability="static"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="terminationTime"
                  type="ogsi:TerminationTimeType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:staticServiceDataValues>
    <ogsi:findServiceDataExtensibility
        inputElement="ogsi:queryByServiceDataNames"/>
    <ogsi:setServiceDataExtensibility
        inputElement="ogsi:setByServiceDataNames"/>
    <ogsi:setServiceDataExtensibility
        inputElement="ogsi:deleteByServiceDataNames"/>
  </sd:staticServiceDataValues>
</gwsdl:portType>


<!-- HandleResolver Port Type -->
<gwsdl:portType name="HandleResolver" extends="ogsi:GridService">
  <operation name="findByHandle">
    <input message="ogsi:FindByHandleInputMessage"/>
    <output message="ogsi:FindByHandleOutputMessage"/>
    <fault name="InvalidHandleFault"
        message="ogsi:InvalidHandleFaultMessage"/>
    <fault name="NoAdditionalReferencesAvailableFault"
        message="ogsi:NoAdditionalReferencesAvailableFaultMessage"/>
    <fault name="NoReferencesAvailableFault"
        message="ogsi:NoReferencesAvailableFaultMessage"/>
    <fault name="NoSuchServiceFault"
        message="ogsi:NoSuchServiceFaultMessage"/>
    <fault name="NoSuchServiceStartedFault"
        message="ogsi:NoSuchServiceStartedFaultMessage"/>
    <fault name="ServiceHasTerminatedFault"
        message="ogsi:ServiceHasTerminatedFaultMessage"/>
    <fault name="TemporarilyUnavailableFault"
        message="ogsi:TemporarilyUnavailableFaultMessage"/>
    <fault name="RedirectionFault"
        message="ogsi:RedirectionFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <sd:serviceData name="handleResolverScheme"
                  type="xsd:anyURI"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="mutable"
                  modifiable="false"
                  nillable="true"/>
</gwsdl:portType>

<!-- Factory PortType -->
<gwsdl:portType name="Factory" extends="ogsi:GridService">
  <operation name="createService">
    <input message="ogsi:CreateServiceInputMessage"/>
```

```
    <output message="ogsi:CreateServiceOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="ServiceAlreadyExistsFault"
        message="ogsi:ServiceAlreadyExistsFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <sd:serviceData name="createServiceExtensibility"
                  type="ogsi:CreateServiceExtensibilityType"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="static"
                  modifiable="false"
                  nillable="false"/>
</gwsdl:portType>

<!-- NotificationSource PortType -->
<gwsdl:portType name="NotificationSource" extends="ogsi:GridService">
  <operation name="subscribe">
    <input message="ogsi:SubscribeInputMessage"/>
    <output message="ogsi:SubscribeOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="TargetInvalidFault"
        message="ogsi:TargetInvalidFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <sd:serviceData name="notifiableServiceDataName"
                  type="xsd:QName"
                  minOccurs="0"
                  maxOccurs="unbounded"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="subscribeExtensibility"
                  type="ogsi:OperationExtensibilityType"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="static"
                  modifiability="false"
                  nillable="false"/>
  <sd:staticServiceDataValues>
    <ogsi:subscribeExtensibility
        inputElement="ogsi:subscribeByServiceDataNames"/>
  </sd:staticServiceDataValues>
</gwsdl:portType>

<!-- Notification Sink PortType -->
<gwsdl:portType name="NotificationSink">
  <operation name="deliverNotification">
    <input message="ogsi:DeliverNotificationInputMessage"/>
  </operation>
</gwsdl:portType>
```

```
<!-- NotificationSubscription PortType -->
<gwsdl:portType name="NotificationSubscription"
    extends="ogsi:GridService">
  <sd:serviceData name="subscriptionExpression"
                  type="xsd:anyType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="sinkLocator"
                  type="ogsi:LocatorType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
</gwsdl:portType>

<!-- ServiceGroupEntry PortType -->
<gwsdl:portType name="ServiceGroupEntry" extends="ogsi:GridService">
  <sd:serviceData name="memberServiceLocator"
                  type="ogsi:LocatorType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="content"
                  type="ogsi:EntryContentType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false"/>
</gwsdl:portType>

<!-- ServiceGroup PortType -->
<gwsdl:portType name="ServiceGroup" extends="ogsi:GridService">
        <sd:serviceData name="membershipContentRule"
                        type="ogsi:MembershipContentRuleType"
                        minOccurs="1"
                        maxOccurs="unbounded"
                        mutability="constant"
                        modifiable="false"
                        nillable="false"/>
        <sd:serviceData name="entry"
                        type="ogsi:EntryType"
                        minOccurs="0"
                        maxOccurs="unbounded"
                        mutability="mutable"
                        modifiable="false"
                        nillable="false"/>
</gwsdl:portType>

<!-- ServiceGroupRegistration PortType -->
```

```
<gwsdl:portType name="ServiceGroupRegistration"
    extends="ogsi:ServiceGroup">
  <operation name="add">
    <input message="ogsi:AddInputMessage"/>
    <output message="ogsi:AddOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="ContentCreationFailedFault"
        message="ogsi:ContentCreationFailedFaultMessage"/>
    <fault name="UnsupportedMemberInterfaceFault"
        message="ogsi:UnsupportedMemberInterfaceFaultMessage"/>
    <fault name="AddRefusedFault"
        message="ogsi:AddRefusedFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="remove">
    <input message="ogsi:removeInputMessage"/>
    <output message="ogsi:removeOutputMessage"/>
    <fault name="ExtensibilityNotSupportedFault"
        message="ogsi:ExtensibilityNotSupportedFaultMessage"/>
    <fault name="ExtensibilityTypeFault"
        message="ogsi:ExtensibilityTypeFaultMessage"/>
    <fault name="MatchFailedFault"
        message="ogsi:MatchFailedFaultMessage"/>
    <fault name="RemoveFailedFault"
        message="ogsi:RemoveFailedFaultMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <sd:serviceData name="addExtensibility"
                  type="ogsi:OperationExtensibilityType"
                  minOccurs="0"
                  maxOccurs="unbounded"
                  mutability="static"
                  modifiable="false"
                  nillable="false"/>
  <sd:serviceData name="removeExtensibility"
                  type="ogsi:OperationExtensibilityType"
                  minOccurs="1"
                  maxOccurs="unbounded"
                  mutability="static"
                  modifiable="false"
                  nillable="false"/>
  <sd:staticServiceDataValues>
    <ogsi:removeExtensibility
        inputElement="ogsi:matchByLocatorEquivalence"/>
  </sd:staticServiceDataValues>
</gwsdl:portType>

</definitions>
```

19.2 http://www.gridforum.org/namespaces/2003/03/serviceData

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
targetNamespace="http://www.gridforum.org/namespaces/2003/03/serviceDat
```

```
a"
     xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
     xmlns="http://www.w3.org/2001/XMLSchema"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">

<attributeGroup name="occurs">
   <attribute name="minOccurs"
              type="nonNegativeInteger"
              use="optional"
              default="1"/>
   <attribute name="maxOccurs">
      <simpleType>
         <union memberTypes="nonNegativeInteger">
            <simpleType>
               <restriction base="NMTOKEN">
                  <enumeration value="unbounded"/>
               </restriction>
            </simpleType>
         </union>
      </simpleType>
   </attribute>
</attributeGroup>

<complexType name="ServiceDataType">
   <sequence>
      <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="name" type="NCName"/>
   <attribute name="type" type="QName"/>
   <attribute name="nillable"
              type="boolean"
              use="optional"
              default="false"/>
   <attributeGroup ref="sd:occurs"/>
   <attribute name="mutability" use="optional" default="extendable">
      <simpleType>
         <restriction base="string">
            <enumeration value="static"/>
            <enumeration value="constant"/>
            <enumeration value="extendable"/>
            <enumeration value="mutable"/>
         </restriction>
      </simpleType>
   </attribute>
   <attribute name="modifiable" type="boolean" default="false"/>
   <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<element name="serviceData" type="sd:ServiceDataType"/>

  <complexType name="ServiceDataValuesType">
    <sequence>
      <any namespace="##any" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
```

```
  <element name="serviceDataValues" type="sd:ServiceDataValuesType"/>
  <element name="staticServiceDataValues"
      type="sd:ServiceDataValuesType"/>
</schema>
```

19.3 http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
targetNamespace="http://www.gridforum.org/namespaces/2003/03/gridWSDLEx
tensions"

xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtens
ions"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified">

  <import namespace="http://schemas.xmlsoap.org/wsdl/"/>

  <element name="portType" type="gwsdl:PortTypeType"/>
  <complexType name="PortTypeType">
    <complexContent>
      <extension base="wsdl:portTypeType">
        <sequence>
          <any namespace="##other" minOccurs="0"
              maxOccurs="unbounded"/>
        </sequence>
        <attribute name="extends" use="optional">
          <simpleType>
            <list itemType="QName"/>
          </simpleType>
        </attribute>
        <anyAttribute namespace="##other"/>
      </extension>
    </complexContent>
  </complexType>
</schema>
```