WSMF/OGSI Re-factoring

Document prepared by:

Shivakumar Jayaraman
Bryan Murray
M. Homayoun Pourheidari
Akhil Sahai
Latha Srinivasan
Jem Treadwell
Aad van Moorsel
(Hewlett-Packard)

Revision: 1.0
Date: September 10, 2003

technical paper

(draft, open for comments)

Version:	1.0 edit 2
Last saved:	September 10, 2003 at 5:29 PM

History

Version	Date	Changes
1.0	8/23/2003	Initial version, draft

Table of Contents

1 Executive Summary 5
2 Introduction 5
3 Background 5
4 Refactoring Topics 7
4.1 GWSDL vs. WSDL1.1 8
4.2 Terminology and Naming of Operations 9
4.3 WSMF Foundation 9
4.3.1 GridService vs. ManagedObject and ManagedObjectIdentity 9
4.3.2 ServiceData vs. Attributes 11
4.3.3 ServiceGroupRegistration vs. Collection 11
4.3.4 Fault (exception) Handling 13
4.4 WS Events 13
4.5 Web Services Management (WSM) 19
4.6 End-to-End 19
5 Glossary 20
6 References 21
7 Refactored WSMF Foundation (GWSDL) 21

1 Executive Summary

The Open Grid Services Infrastructure (OGSI) defines a set of interfaces, behaviors and conventions on the use of Web Services. These behaviors can be leveraged by proposed management standards such as WSMF as a foundation for using Web Services.

This document, therefore, discusses re-factoring[1] of WSMF using OGSI 1.0. We discuss the technology issues, our preferred resolution, and the remaining open problems. It is expected that both WSMF and OGSI will work together to better support manageability requirements.

The first result of this re-factoring is an OGSI-compliant specification for 're-factored WSMF' expressed in terms of GWSDL. The second result is a WSDL1.1 equivalent derived from this re-factored WSMF.[2]

The main results are summarized in the table of Figure 1. The GWSDL description of re-factored WSMF Foundation is provided in Section 7.

This is a draft version; we invite comments and discussion.

2 Introduction

We believe that Grid technologies, as they are defined through OGSA and OGSI, are fundamental for service-oriented representations of resources that are part of distributed applications and IT environments. Proposed management standards, such as WSMF, should therefore interoperate with Grid and leverage its technologies.

WSMF is conceptually very close to OGSI as both introduce as core of their architecture a web service representation for resources. In WSMF this is the managed object, in OGSI the grid service. Standardization of WSMF thus inevitably results in resolving issues in ways that are similar to OGSI, and it is therefore to everyone's benefit to modify the originally proposed WSMF specification so it leverages Grid developments.

By definition, re-factoring WSMF using OGSI is adapting the originally proposed WSMF specification to leverage OGSI without changing WSMF's functionality. This re-factoring exercise is based on:

- WSMF version 2.0, as released July 16, 2003 [1], and updated July 28, 2003, and as obtainable from http://devresource.hp.com/wsmf.
- A technical note on dynamic attributes and meta information, available upon request [2].
- OGSI Version 1.0, Final, June 27, 2003, as available from https://forge.gridforum.org/projects/ogsi-wg/document/Final_OGSI_Specification_V1.0/en/1/Final_OGSI_Specification_V1.0.pdf [4].

There are various issues that we need to take into account to provide the best standardization plan for the web services management industry. Although strictly speaking re-factoring WSMF using OGSI implies a GWSDL specification with WSMF functionality, the industry requires more than that. In particular, we need a WSDL1.1 version of re-factored WSMF, a topic we discuss in detail in Section 4.1. Furthermore, re-factored WSMF must have the right abstraction to fit with the management standardization plans within WSDM, and the Global Grid Forum.

This document takes the first step towards aligning web services management and grid developments by providing a GWSDL and WSDL1.1 version of WSMF utilizing OGSI.

3 Background

The **Web Services Management Framework (WSMF)** [1] is a logical architecture for the management of resources, including Web services themselves, through Web services[3]. WSMF 2.0 is specified using WSDL1.1, and thus does not

[1] In the software engineering community, re-factoring is "improving the design of code without introducing new behavior" (http://industriallogic.com/xp/refactoring/). Here, we improve the WSMF specification using OGSI constructs, so we speak of 're-factoring WSMF using OGSI,' and the result of re-factoring is 're-factored WSMF.'

[2] The third result of re-factoring should be a re-factored WSMF that has the right abstraction so it can be a useful component in the future OGSA standardization plans—this is not the primary focus of this document, but we expect this can and will be done as a next step.

contain GWSDL extensions. WSMF is based on the notion of managed objects and their relationships. A managed object essentially represents a resource and exposes a set of management interfaces through which the underlying resource could be managed (see the overview document in [1]). Similarly, relationships among managed objects represent relationships among underlying resources. The management functions addressed by WSMF include: discovery of the management WSDL definitions; discovery of the topology of the managed objects; registrations and retrieval of notifications; monitoring, auditing, and controlling various aspects of managed objects by using the supported management operations.

WSMF has three parts:

- WSMF Foundation: the core pieces of WSMF
- WS-Events: an event subsystem with an advanced push model
- Web Service Management (WSM): the data model for managing the web services domain

The first two deal with management *using* web service, the latter deals with management *of* web services. Throughout this document we alternately use WSMF and WSMF2.0 to refer to the original proposal for management using web services published in [1], and use 're-factored WSMF' to denote either the GSWDL or WSDL1.1 version of WSMF redesigned using OGSI.

The **Open Grid Services Architecture (OGSA)** [3] addresses the challenges in integrating services across distributed, heterogeneous, dynamic "virtual organizations" formed from the disparate resources within a single enterprise and/or from external resource sharing and service provider relationships. Building on concepts and technologies from the Grid and Web services communities, this architecture defines a uniform exposed service semantics (the *Grid service*); defines standard mechanisms for creating, naming, and discovering potentially transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities.

The OGSA platform encompasses the **Open Grid Services Infrastructure (OGSI)**. From the completed OGSI draft [4]: "[…] 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."

The **Common Management Model (CMM)** is a working group within Global Grid Forum (GGF). From its charter [5]: "The purpose of this WG will be to define the Common Management Model in which [...] entities will be represented as manageable resources and services. Manageable resources and services can include any type of entity, ranging from hardware (such as a disk drive), to software components (such as a database or message queue), to complete solutions (such as a billing system or provisioning), and also to transient things such as print jobs. The Common Management Model will provide a set of port types that build on and supplement the GGF's OGSI Specification [4] that are of broad and general use for management in the Grid."

[3] The term Web services is used to describe the approach to loosely-coupled distributed computing in which interactions are carried out through the exchange of XML messages, exchanged in accordance to the Simple Object Access Protocol (SOAP) specification and described through the Web Services Description Language (WSDL).

4 Re-factoring Topics

Subject	OGSI	WSMF	Issue	Resolution	Remaining Open Issues
WSDL version	GWSDL based, WSDL1.1 compatible through GWSDL to WSDL1.1 transformation; anticipates migration to WSDL 2.0 (needs inheritance and information elements)	WSDL 1.1 based; anticipates migration to WSDL2.0 when it becomes available	- What description language to use for re-factored WSMF? - How to translate SDEs from GWSDL to WSDL1.1?	- Re-factored WSMF results in both a GWSDL and a WSDL1.1 spec. Translation tools used only if normative. Unification is expected to take place through WSDL2.0 (early 2004?) - Translate SDEs through corresponding complexType in WSDL1.1.	- When will normative GWSDL2WSDL1.1 be available? - When tool support? - When WSDL2.0?
Terminology and naming of operations			How to choose between different names for similar operations, and what to do with names that are unfamiliar in management domain?	General rule: do not use more than one operation name in spec, leave to tooling to generate added WSDL/APIs. Otherwise, decide case by case.	
resource representation	GridService represents a resource. Every OGSI service extends GridService.	ManagedObject represents a resources. Every ManagedObject exposes ManagedObjectIdentity.	- should re-factored WSMF use 'extends' ManagedObjectIdentity in various WSMF portTypes. - can we expect tools to prevent implementation overhead when using extends in every portType?	Re-factored WSMF adopts GridService, and extends ManagedObjectIdentity in all WSMF portTypes.	- do tools deal efficiently with a base port type that is extended more than once?
service data	ServiceData is used to represent state information.	Traditional use of attributes and Get/Set operations. The WSMF technical note provides a generalized Get/Set, allowing dynamic addition of attributes, and ways of obtaining subsets of attributes.	- shall we maintain Get/Set for individual attributes or replace attributes with SDEs? - how can we maintain strong typing? - can we add attributes at run-time (dynamic attributes)? - can we add meta information?	- GWSDL version will incorporate SDEs; for the WSDL1.1 version we suggest to add complexType for SDEs. - Introduce specific Get/Set in implementation as needed for strong typing by tooling support. - dynamic attributes supported in SDEs - meta information supported in SDEs, but insufficient	- how do tools generate the strongly typed operations—can they be configured? - WSDL1.1 translation of ServiceDataNames to deal with dynamic attributes is not well understood - how to progress with general solution for meta information?
group / collection	ServiceGroup	collections	- WSMF's Invoke is not present in Service Group - WSMF does not have all member operations	WSMF collection extends ServiceGroupRegistration, and we add "Invoke" operation	

Subject	OGSI	WSMF	Issue	Resolution	Remaining Open Issues
fault handling	Exceptions/Faults specified in interface definition, provides locator of the fault	Defined faults are listed in the WSMF technical doc	What fault model to adopt?	Re-factored WSMF adopts OGSI's fault handling, mapping WSMF error codes into appropriate OGSI faults and add new ones where appropriate.	

Figure 1. WSMF re-factoring topics

4.1 GWSDL vs. WSDL1.1

WSDL1.1 [6] is the default description language for web service interfaces, and web services tool support typically assumes WSDL1.1 specifications. GWSDL [7] is an extension of WSDL1.1 which provides features like portType inheritance and open content. It is used by OGSI as an interim description language until WSDL2.0 is released. WSDL2.0 (previously termed WSDL1.2) [8] is under development and is expected to subsume WSDL1.1. Similar to GWSDL, WSDL2.0 uses inheritance; WSDL2.0 also introduces 'information elements,' which can be seen as the counterpart of GWSDL's support for open content in portTypes used to support the service data construct. The hope and expectation therefore is that WSDL2.0 will replace and unify WSDL1.1 and GWSDL specifications, providing a single, common, description language for any grid or web service [10][11].

Until WSDL2.0 appears we provide GWSDL and WSDL 1.1 mappings for the WSMF interfaces.

Re-factored WSMF results in a normative GWSDL spec.

1. Re-factored WSMF provides a WSDL1.1 spec, generated from the GWSDL spec as follows:
    a. Once normative GWSDL2WSDL1.1[4] [9] is accepted or "good enough," we will use that approach to generate the WSDL1.1 version.
    b. Until normative GWSDL2WSDL1.1 is accepted or "good enough," we will create the WSDL1.1 spec 'by hand.'
    c. We will not use GWSDL2WSDL1.1 translation tools (such as those currently available) that are not based on a normative GWSDL2WSDL1.1 spec, since they produce unsatisfactory results.
2. Once WSDL2.0 is accepted, we expect to retire both the GWSDL and WSLD1.1 version of re-factored WSMF, and instead create a normative WSDL2.0 version of re-factored WSMF.

We also have the following topics and suggestions for discussion in translating GWSDL to WSDL1.1:

- **Service data:** the most elegant solution for service data elements in WSDL 1.1 seems to be to declare servicedata elements in the types section of the WSDL 1.1 document. The WSDL 1.1 portType element can then refer to the QNames of these elements. This is possible because the WSDL schema available at the URL http://schemas.xmlsoap.org/wsdl supports attribute extensibility for portTypes. It should be however noted that the WSDL schema specified in the WSDL 1.1 specification at http://www.w3.org/TR/wsdl does not support attribute extensibility in portTypes. We would like to suggest this solution to the group working on normative GWSDL2WSDL1.1 [9].
- **Dynamic attributes:** as we mention under the dynamic attributes discussion in Section 4.3.2, the implications of translation of *ServiceDataNames* is uncertain to us at this moment.
- **Factory references in GridService:** OGSI includes a factory concept used for creating and destroying Grid services, and includes a handle in *GridService* to interact with the factory that created this service (*factoryLocator* SDE). A service that was not created by a factory will expose this SDE with a value of *xsi:nil*. A translation from GWSDL to WSDL1.1 of SDEs will contain this attribute with a value of nil. However, we do not see this as a serious flaw, and propose therefore to leave the *factoryLocator* in the WSDL1.1 version, setting *nillable* to true in

[4] The most elegant solution for SDEs in normative WSDL1.1 seems to be to add SDEs as complexType, and let the portType call out QName of the SDE. We would like to suggest this to the group working on normative GWSDL2WSDL1.1 [9].

the SDE. Nonetheless, we believe that future versions of OGSI can be componentized in a way that will not include any factory information in deployments that do not require a factory or are not Grid based.

4.2 Terminology and Naming of Operations

Each standard as well as each application domain has its preferred terminology usage, and when re-factoring WSMF, one often comes across terminology differences between OGSI, WSMF and the management software community, e.g., source/sink versus producer/consumer, EventType versus NotificationTopic, Grid service versus managed object, etc.

We have to make decisions as needed, following the rule:

- We will not use multiple operation names for the same functionality (e.g., use OGSI and WSMF names side by side), since we want to keep the standard as lean as possible. Instead, we rely on tooling to provide client or server with desired WSDL/API extensions.

4.3 WSMF Foundation

Re-factoring WSMF Foundation is in large part adopting OGSI *GridService* as the base abstraction of managed objects, and utilizing its service data feature to represent attributes. Most of this section is therefore devoted to *GridService* and *ServiceData*.

To utilize OGSI, we include the OGSI namespaces:

```
<definitions name="WSMF-OGSI"
 targetNamespace=”http://devresource.hp.com/drc/
   specifications/wsmf/2003/07/wsmf-ogsi-foundation”
   xmlns=”http://schemas.xmlsoap.org/wsdl/”
   xmlns:gwsdl=”http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions”
   xmlns:ogsi=”http://www.gridforum.org/namespaces/2003/03/OGSI”
   xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:sd=”http://www.gridforum.org/namespaces/2003/03/serviceData”
   xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
   xmlns:wsmf="http://devresource.hp.com/drc/specifications/wsmf/2003/07/
      wsmf-ogsi foundation"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
   <import location="../../ogsi/ogsi.gwsdl"
      namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
   …
</definitions>
```

Figure 2. name spaces in re-factored WSMF, GWSDL specification

4.3.1 GridService vs. ManagedObject and ManagedObjectIdentity

WSMF Foundation defines a managed object for each managed resource, and each managed object must implement the WSMF *ManagedObjectIdentity* interface. Conceptually similar is OGSI's *GridService*, which any OGSA service must extend. Re-factoring implies taking *GridService* as the base, and modifying *ManagedObjectIdentity* and *ManagedObject* so they benefit from *GridService* constructs.

The following piece of refactored WSMF shows how *ManagedObjectIdentity* extends *GridService*, using GWSDL.

```
<gwsdl:portType name="ManagedObjectIdentity" extends="ogsi:GridService">
   <sd:serviceData name="ID"
      type="wsmf:EntityReferenceType"
      minOccurs="1"
      maxOccurs="1"
      mutability="mutable"
      modifiable="false"
      nillable="false" />
</gwsdl:portType>
```

Figure 3. ManagedObjectIdentity extends GridService

The objective of *ManagedObjectIdentity* is to provide identification information about the managed object and the resource it represents (hence, the name equals "ID"). Through the inheritance of *GridService*, we can now utilize service data accessor and modifier operation (*FindServiceData* and *SetServiceData*), and have a mechanism to manage the life cycle of the managed object (through Grid's *goodUntil*, etc.).

In OGSI, use of *GridService* is enforced through the *extend* construct, which all port types use. In WSMF, the WSDL specifies no such enforcement, although it is explicitly stated that managed objects MUST implement *ManagedObjectIdentity*. For other port types in WSMF, such as *Monitoring*, *Configuration*, *Control*, etc., we have chosen to take the same approach as OGSI, and use *extends* to enforce that *ManagedObjectIdentity* is leveraged. The extension of the *ManagedObjectIdentity* portType by all WSMF portTypes will enforce implementation of the *ManagedObjectIdentity* portType -- the one required portType that will provide the essential pieces of information for a managed object. However, if improperly implemented it could also bring about extra overhead for the WSMF portTypes. It is recommended that implementers pay special attention to this issue such that their products do not end up with a large footprint. As an example, we give the Configuration portType, which is meant to give information about the managed object itself.

```
   <gwsdl:portType name="Configuration" extends="wsmf:ManagedObjectIdentity">
      <sd:serviceData name="Name"
         type="s:string"
         minOccurs="1"
         maxOccurs="1"
         mutability="mutable"
         modifiable="true"
         nillable="false" />
      <sd:serviceData name="Type"
         type="wsmf:ManagedObjectType"
         minOccurs="1"
         maxOccurs="1"
         mutability="mutable"
         modifiable="true”
         nillable="false" />
      <sd:serviceData name="Description"
         … />
      <sd:serviceData name="Owner"
         … />
      <sd:serviceData name="Vendor"
         … />
      <sd:serviceData name="Version"
         … />
      <sd:serviceData name="ManagedObjectVersion"
         … />
      <sd:serviceData name="CreatedOn"
         … />
      <sd:serviceData name="HostName"
         … />
      <sd:serviceData name="ManagedObjectHostName"
         … />
   </gwsdl:portType>
```

Figure 4. Other WSMF port types extend ManagedObjectIdentity, and use ServiceData

As one can see from the above example, re-factoring WSMF Foundation is relatively straightforward. The only portType requiring special consideration is the collection interface, which we discuss in Section 4.3.3.

Handle resolution: WSMF provides managed object interfaces through which their unique identity and references can be exposed. This aligns well with the OGSI notion of grid service handles and grid service references, so re-factored WSMF can rely on the corresponding OGSI constructs. In doing so, the resulting set of interfaces in re-factored WSMF supports various implementations to obtain unique managed object identities.

4.3.2 ServiceData vs. Attributes

A critical construct in OGSI is *ServiceData*, which is used to access state information. *ServiceData* is a set of named values of type pre-specified in XSD, and can be queried and modified. One can also subscribe to receive events about changes in *ServiceData*.

Currently WSMF provides strongly typed get/set operation pairs for each attribute. But this is not a very scalable model and additionally does not support runtime attribute additions (dynamic attributes, see below). Hence WSMF is moving towards a generic get/set operation that will provide attribute values for all attributes, whether they are defined at design time or run-time, see the technical note [2]. This is similar to OGSI's current behavior and its SDE retrieval operation—*findServiceData*, and we therefore utilize OGSI's service data concept to the fullest in re-factored WSMF.

Figure 4 illustrates the use of *ServiceData* in refactored WSMF. We have chosen to introduce a service data element for every element in each portType. For all these SDEs, we then determine the values of *minOccurs, maxOccurs,* etc.

Although re-factoring WSMF using SDEs is relatively straightforward, there are several issues related to accessing and specifying 'information' that require further investigation:

Strong typing. The traditional Get/Set approach has the advantage that it offers strong typing. This is sometimes important, since it helps compile time verifications. If that is the case, we introduce operations that support interactions with some pieces of the management data even though the same data can be retrieved through the generic operations. An example could be the state information and support for a *getState* operation.

Dynamic attributes. To support modifications of attribute values in addition to creation of new attributes at runtime, WSMF introduces dynamic attributes in its technical note [2]. OGSI supports such a notion as well. Updates to SDE attributes are handled through notifications that are defined on SDEs. New SDEs can be discovered by subscribing to notifications on the *serviceDataName* SDE where all the SDE names are captured. Once this notification is detected, a client can deduce the new SDE names and then query their values through the *findServiceData* operation. Re-factored WSMF thus can leverage this approach. **GWSDL to WSDL1.1 translation of service data.** See Section 4.1.

Meta information. Meta information is additional information on managed objects and their elements, beyond the name and value information. As an example, the *Name* attribute in the *Configuration* portType of Figure 4 can have a GUI display version of the name, or names in different languages, or access rights. The technical note [2] describes a possible solution for meta information in detail.

OGSI does not provide special support for meta information for a grid service and its components. However, there are ways to capture some of the meta information by introducing additional SDEs in each portType to capture meta information related to attributes and operations. The attribute meta information SDE would have one entry for each of the *serviceData* elements that has meta information. It could contain elements such as dataType, name, description, updateFrequency, and have an "any" construct to allow extension. It is not clear whether updates and access to this information at runtime is supported.

Since we consider meta information very important for manageability models and management applications, we prefer a generic solution for meta information, possibly through additional standardization efforts. This may imply augmentation of SDEs to include meta information, a topic that should be further discussed.

4.3.3 ServiceGroupRegistration vs. Collection

There are cases where an application wants to take the same action on several managed objects (see Section 4.6 of WSMF Foundation [1]). The *ManagedObjectCollection* management interface allows one managed object to act as a proxy for several other managed objects. A collection is a set of managed objects that may be accessed through a *ManagedObjectCollection* management interface implemented by one managed object.

OGSI uses the notion of *ServiceGroup*, a portType that extends *GridService* and maintains information about a group of other grid services (see Section 13 in [4]). One intended use of *ServiceGroup* is that it forms the basis to implement a traditional registry.

In re-factored WSMF, the *Collection* portType inherits from the *serviceGroupRegistration* portType. This allows the WSMF *Collection* portType to provide operations to support addition/modification of collection members and introspection of the collection list.

The part from *ManagedObjectCollection* that is missing from *ServiceGroup* is the *Invoke* operation. The *Invoke* operation allows a manager to invoke the same operation on all or a subset of members within the collection, for instance to query or set *serviceData* values, or to perform some control operation. The *Select* argument to *Invoke* specifies an XPath expression to identify the collection members to take part in the operation, the *Interface* and *Name* arguments specify which operation, and the *ArgumentList* holds the parameters to be sent to the operation. The response to the *Invoke* operation is a list of elements, one for each member matching the *Select* expression in the request, that contain a reference to a member, and the response or fault returned from the member when sent the requested operation. Since the types, messages, and portType do not exist in OGSI, they need to be included in a portType which extends ServiceGroupRegistration. The result is the following piece of refactored WSMF in GWSDL:

```
<types>
   <s:schema targetNamespace=http://devresource.hp.com/drc/specifications/wsmf/
      2003/07/wsmf-ogsi-foundation
      attributeFormDefault="qualified"
      elementFormDefault="qualified">
      <s:complexType name="SelectExpressionType">
         <s:restriction base="s:string" />
      </s:complexType>
      <s:element name="Select" type="wsmf:SelectExpressionType" />
      <s:complexType name="ManagedObjectResponseInformationType">
         <s:sequence>
            <s:element name="ManagedObjectReference" type="ogsi:HandleType" />
            <s:sequence>
               <s:choice>
                  <!-- Either the response from the Invoke operation or a fault. The
                     response from the Invoke operation would be the children of the
                     response SOAP Body. A fault will be the WSMF defined Fault element
                     which will contain the SOAP Fault element or other transport or
                     message protocol fault information.
                  -->
                  <s:any namespace="##any" minOccurs="0" maxOccurs="unbounded"
                        processContents="lax" />
                  <s:element name="Fault">
                     <s:complexType>
                        <s:sequence>
                           <s:any namespace="##other" minOccurs="0"
                                 maxOccurs="unbounded"
                                 processContents="lax" />
                        </s:sequence>
                     </s:complexType>
                  </s:element>
               </s:choice>
            </s:sequence>
            <s:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                  processContents="lax" />
         </s:sequence>
         <s:anyAttribute namespace="##other" processContents="skip" />
      </s:complexType>
      <s:element name="ManagedObjectResponseInformation"
            type="wsmf:ManagedObjectResponseInformationType" />
      <s:complexType name="ManagedObjectResponseInformationListType">
         <s:sequence>
            <s:element ref="wsmf:ManagedObjectResponseInformation" minOccurs="0"
                  maxOccurs="unbounded" />
         </s:sequence>
      </s:complexType>
      <s:element name="ManagedObjectResponseInformationList"
            type="wsmf:ManagedObjectResponseInformationListType" />
      <s:element name="Invoke">
         <s:complexType>
            <s:sequence>
               <s:element ref="wsmf:Select" minOccurs="0" />
               <s:element name="Interface" type="s:QName" />
               <s:element name="Name" type="s:NMTOKEN" />
               <s:element name="ArgumentList" minOccurs="0">
```

```
                     <s:complexType>
                        <s:sequence>
                           <s:any minOccurs="1" maxOccurs="unbounded"
                              processContents="lax" />
                        </s:sequence>
                     </s:complexType>
                  </s:element>
               </s:sequence>
            </s:complexType>
         </s:element>
         <s:element name="InvokeResponse">
            <s:complexType>
               <s:sequence>
                    <s:element ref="wsmf:ManagedObjectResponseInformationList" />
               </s:sequence>
            </s:complexType>
         </s:element>
      </s:schema>
   </types>
   <message name="InvokeInput">
      <part name="document" element="wsmf:Invoke" />
   </message>
   <message name="InvokeOutput">
      <part name="document" element="wsmf:InvokeResponse" />
   </message>

   <gwsdl:portType name="Collection" extends="ogsi:ServiceGroupRegistration" >
      <operation name="Invoke">
         <input message="wsmf:InvokeInput" />
         <output message="wsmf:InvokeOutput" />
      </operation>
   </gwsdl:portType>
```

4.3.4 Fault (exception) Handling

OGSI defines a base XSD type, FaultType, for all fault messages, which must be returned by all Grid services. A fault message may include a text description of the fault, a locator for the service instance that raised the fault, a timestamp, a fault cause, a fault code and extension information. Multiple fault types can be nested or chained in a single message, so that fault information can reflect the service invocation stack, allowing a recipient to drill down throught the causes to learn more about the reasons for the fault. Services may extend FaultType to provide more detailed fault information.

The WSMF FaultDetail element contains one or more Error elements, each of which contains an ErrorCode element and an optional text message. An extension element is provided to allow managed objects to add more detailed information about the fault.

Since the fault handling of OGSI is more comprehensive than that of WSMF, we propose to use the OGSI model. We will translate the WSMF error codes and/or add new fault types as appropriate.

4.4 WS Events

4.4.1 Introduction

WSMF WS-Events is an extensible XML schema and a set of publishing rules for advertising, subscribing, producing and consuming Web Service events using push or pull modes. OGSI supports notifications through the NotificationSource, Notification Sink and NotificationSubscription portTypes

WS-Events defines an 'event' as a 'change in the state of a resource'. WS-Events also defines an XML schema to represent event information. OGSI defines state in the form of service data elements (SDEs), and provides the support to asynchronously notify interested parties (subscribers), when the value of these SDEs change. We propose merging these two concepts by describing every 'event' as an SDE. The 'type' of these events will be similar to the event schema as defined by WS-Events.

The OGSI model is primarily tailored towards the push mode of subscribing and consuming notifications. OGSI also does not directly support event filtering. We propose the following extensions to the OGSI 1.0 Notification model in order to support event filtering and pull notifications akin to WS-Events.

The namespace conventions used throughout this section are described below

```
<definitions name="WSEvents-OGSI"
   targetNamespace=”http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
events”
      xmlns=”http://schemas.xmlsoap.org/wsdl/”
      xmlns:gwsdl=”http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions”
      xmlns:ogsi=”http://www.gridforum.org/namespaces/2003/03/OGSI”
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:sd=”http://www.gridforum.org/namespaces/2003/03/serviceData”
      xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
      xmlns:events="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
events">

      <import location="../../ogsi/ogsi.gwsdl"
              namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
```

4.4.2 Discovering the list of notifications

- o WS-Events defines a ‘Discovery’ portType, which has a ‘GetAllEventTypes’ operation. This operation is used by clients to discover the list of event types supported by an implementation. The operation returns a list of EventTypes (which are essentially URIs) supported.
- o OGSI defines a ‘NotificationSource’ portType which has an SDE called ‘notifiableServiceDataName’. The values of this SDE list a set of SDE names (which are QNames), which can be subscribed to. Changes to the values of the subscribed SDEs would trigger notifications.
- o The discovery approaches used by WS-Events and OGSI are functionally identical.

The proposal is to reuse the OGSI mechanisms for notifications. We plan to extend the ogsi:NotificationSource portType with a new port type (events:BufferedNotificationSource). ManagedObjects that support WSMF events need to implement this portType or an extension of it. Every supported notification type would be declared as an SDE. The ‘notifiableServiceDataName’ SDE in the ogsi:NotificationSource portType would be used to discover supported notifications.

Discovering the notifications supported by an implemention will now be accomplished by performing a findServiceData operation on implementations of the above portType, using a ‘queryByServiceDataNames’ extensible operation and by using ‘ogsi:notifiableServiceDataName’ as the name of the SDE to be queried. A list of the QNames of supported notifications would be returned.

The gwsdl snippet describing the BufferedNotificationSource portType is shown in section 4.4.3

4.4.3 Discovering Meta Information about notifications

- o WS-Events supports a ‘GetEventTypeDefinition’ operation in its ‘Discovery’ portType, which returns more information about a specified list of EventTypes.
- o OGSI does not directly define a mechanism to discover Meta information about SDEs that support notifications, though this feature could be easily added using more SDEs.
- o Since the OGSI model does not define direct support for meta information, we have to add support for this feature to match the capabilities of WS-Events.

The proposal is to define a new SDE called 'notificationMetaInfo' in the events:BufferedNotificationSource portType. This SDE will have a schema type of NotificationMetaInfoType. Values for this SDE will include Meta information corresponding to all the notifications supported. The following schema and gwsdl snippets demonstrate this. The NotificationMetaInfoType is modeled exactly similar to the WS-Events schema type 'EventTypeDefinitionType'.

```
<xsd:complexType name="NotificationMetaInfoType">
    <xsd:sequence>
        <xsd:element name="Notification" type="xsd:QName"/>
        <xsd:element name="SchemaLocation" type="xsd:anyURI"/>
        <xsd:element name="Description" type="string" minOccurs="0"/>

        <xsd:element name="Causes" type="events:NotificationMetaInfoListType"
                      minOccurs="0"/>

        <xsd:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
                  processContents="lax"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="ogsi:LifeTimePropertiesGroup"/>
    <xsd:anyAttribute namespace="##other" processContents="skip"/>
</xsd:complexType>

<xsd:complexType name="NotificationMetaInfoListType">
    <xsd:sequence>
        <xsd:element name="NotificationMetaInfo" type="events:NotificationMetaInfoType"
                     minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>



<gwsdl:portType name="BufferedNotificationSource" extends="ogsi:NotificationSource">

    <sd:serviceData name="notificationMetaInfo" type="events:NotificationMetaInfoType"
                    minOccurs="0"  maxOccurs="unbounded"/ >
 … …
</gwsdl:portType>
```

Below is an explanation of the various elements and attributes used in the previous snippet. (These elements are very similar in nature to those defined in the WS-Events specification. Please refer to that specification for more information).

1. Notification element in NotificationMetaInfoType refers to the QName of the SDE representing the notification.
2. SchemaLocation refers to the URI to a XML schema that the subscriber could use to validate notifications against.
3. Description is an human readable string describing the purpose and semantics of the notification.
4. Causes refers to a list of other notifications that the current notification was a result of. This could be used for correlation purposes.

Queries on the 'notificationMetaInfo' SDE using findServiceData by name would return the meta -information corresponding to all the notifications supported. To support meta-information queries on specific notifications, an XPath based query expression could be used as the extensible operation input for findServiceData. We propose declaring XPath based expressions for the 'findServiceDataExtensibility' SDE declared in ogsi:GridService For example, an expression like '//events:notificationMetaInfo/[Notification="app:FileCreatedNotification"], used on the notificationMetaInfo SDE would return the meta information corresponding to the FileCreatedNotification notification.

4.4.4 Notifications about Notifications

- WS-Events supports two special kinds of notifications that all compliant ManagedObjects need to support. These two notifications are to announce
  1. The inclusion of a new dynamic notification type (NewEventTypeNotification)
  2. The change to the meta information of an existing notification type (EventTypeUpdatedNotification)
- OGSI does not define special types of notifications to describe other notifications.
- Since OGSI does not support this, we have to add support for special notification types to match the capabilities of WS-Events.

The proposal for including this feature in the new model is to add 2 SDEs to the BufferedNotificationSource portType corresponding to these notification types. The schema types for these notifications would correspond to a base NotificationType that we define that forms a template for all notifications. The gwsdl portType for events:BufferedNotificationSource thus looks like below

```
<xsd:complexType name=”NotificationType”>
    <xsd:sequence>
          <xsd:element name=”Originator” type=”ogsi:LocatorType”/>
          <xsd:element name=”UUID”  type=”xsd:anyURI”/>
          <xsd:element name=”extensibilityElement” type=”ogsi:ExtensibilityType”
                       minOccurs=”0” maxOccurs=”unbounded”/>
          <xsd:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
                  processContents="lax"/>
    </xsd:sequence>
    <xsd:attributeGroup ref=”ogsi:LifeTimePropertiesGroup”/>
</xsd:complexType>

<gwsdl:portType name=”BufferedNotificationSource” extends=”ogsi:NotificationSource>

   ...

    <sd:serviceData name=”NewEventTypeNotification”  type=”events:NotificationType”
                    minOccurs=”0”  maxOccurs=”1” / >
    <sd:serviceData name=”EventTypeUpdatedNotification”  type=”events:NotificationType”
                    minOccurs=”0”  maxOccurs=”1” / >
</gwsdl:portType>
```

Notification would be a wrapper type for notifications. The ‘extensibilityElement’ element of type ogsi:ExtensibilityType is a simple wrapper around any XML content which can be utilized by the event source to wrap any application specific information. The any element in the schema could be used to support extended types.

Note: We could use events:NotificationType for the special notifications mentioned above or even declare extensions of these types if needed. Currently we don’t see a need to specialize types for specific events and therefore use the basic type.

Clients who subscribe to these SDEs will be notified if new notification types are added to the system or if any Meta information of existing Notifications changes.

4.4.5 Subscription

- WS-Events supports the ability to subscribe to a set of events. The subscribe operation defined in the Subscription portType allows clients to specify the following inputs
    1. A regex based subscription expression specifying the list of events to subscribe to
    2. An expiration time that specifies the initial value for the lifetime of the subscription
    3. An optional filter that could further select events that the client needs notifications on.
    4. An optional callback address which specifies a destination where 'push' type notifications could be asynchronously delivered
    5. Absence of the callback address means a 'pull' type subscription. In this case the managed object buffers the events for the client until the client asks for it through specific 'pull' operations. No asynchronous notifications occur.
    6. The response to a subscribe operation is a subscriptionId that uniquely identifies the subscription which can be used to control the lifetime of the subscription using the ExtendSubscription and CancelSubscription operations.
- OGSI NotificationSources support a 'subscribe' operation which allows the following inputs
    1. An extensible subscription expression. NotificationSources have to support one variant which allows subscribers to specify a list of SDEs and the intended periodicity of notifications.
    2. A Locator to a NotificationSink specifying the destination for asynchronous notifications.
    3. An expiry time indicating the initial lifetime of the subscription
    4. The response to the subscribe operation is the Locator to a GridService instance implementing the NotificationSubscription portType which can be used to control the lifetime of the subscription.
- OGSI does not support 'pull' type notifications. It also does not support the notion of event filters directly (though this can be added as an extension). We therefore have to add support for these features to match the capabilities of WS-Events.

Filters are selectors that the NotificationSource could use to determine at runtime whether a subscriber should be notified about a specific notification instance. In general it should provide a way to define a piece of logic that can be applied by the source of notifications to attributes available to it at runtime. This could mean that the logic could provide a specific condition under which a subscriber would like to receive an event, to something more complex like how to gather performance information and do measurements and when to let clients know that some state or threshold has been reached. Filters could be as simple as a template that the notification instance could be matched against or could be as complex as an expression based on runtime properties that would evaluate to a Boolean value, which would help the source decide whether the notification should be discarded or dispatched. An example of a complex filtering expression could be an expression that specifies something like, if events 1, 2 and 3 happen and then event 4 happens within one hour, then send a notification.

The OGSI specification defines the 'subscription expression' as an XML element that describes what messages should be sent from the notification source to the notification sink. In keeping with this definition, we think that an ideal place to specify filters would be the subscription expression itself. Since this expression is an extensible operation whose allowable values can be queried by the client, filters that the source supports can be determined by the client using findServiceData. Specific filters would be covered in a later document.

The proposal is to adopt the OGSI way of subscribing to notifications, with slight changes introduced to support the 'pull' type of buffered notifications supported by WS-Events. The following points describe the subscribe semantics of events:BufferedNotificationSource portType.

1. We propose more variants for the subscription expression, which let the client specify filters to further select notifications. We need this to provide the 'filtering' functionality those WS-Events supports. We could also use the subscription expression to indicate profiles so that the events which get sent could be adapted for different kinds of devices (like PDAs etc). Also the notification semantics (push| pull) could be specified in the expression itself.

2. The response to a subscription would be a Locator to a GridService implementing the ogsi:BufferedNotificationSubscription portType.
3. We introduce the notion of a new portType called 'events:BufferedNotificationSubscription'. This portType would extend the ogsi:NotificationSubscription portType
4. Clients could now perform pulls from this buffer using 'findServiceData'. We propose specifying more variants for findServiceData (by adding more staticServiceDataValues for its extensible operation), that support richer pulls the way WS-Events does. WS-Events allows clients to pull events that happened after a given absolute time or that happened in a particular time range.
5. Explicit pull operations would also be supported on the BufferedNotificationSubscription to make the pull intent more clear than what findServiceData accords.
6. The client can control the lifetime of their subscriptions by controlling the lifetime of the BufferedNotificationSubscription that manages their subscription
7. The new portType's bufferSize SDE is used to support the GetEventInstanceInfo operation that WS-Events supports to allow queries on buffer sizes.

```
<gwsdl:portType name="BufferedNotificationSubscription"
                extends="ogsi:NotificationSubscription">

    <sd:serviceData  name="notificationBuffer" type="events:NotificationType"
                     minOccurs="0"   maxOccurs="unbounded"/ >
    <sd:serviceData  name="bufferSize" type="xsd:int" / >

</gwsdl:portType>
```

4.4.6 Faults

OGSI fault mechanisms will be adopted to describe all faults resulting from this subsystem.

4.4.7 Example

The following example describes an implementation of a FileWatcher managed object that supports three kinds of notifications

1. FileCreated
2. FileDeleted
3. FileUpdated

The implementation could declare a new portType that extends the BufferedNotificationSource portType and specify SDEs for the notifications it supports

```
<gwsdl:portType name="FileWatcher"
                extends="events:BufferedNotificationSource">

    <sd:serviceData  name="FileCreated" type="events:NotificationType"
                     minOccurs="0"   maxOccurs="1"/ >
    <sd:serviceData  name="FileDeleted" type="events:NotificationType"
                     minOccurs="0"   maxOccurs="1"/ >
    <sd:serviceData  name="FileUpdated" type="events:NotificationType"
                     minOccurs="0"   maxOccurs="1"/ >
    ..
</gwsdl:portType>
```

The managed object would be a GridService implementing this portType. The notifiableServiceDataName SDE that the MO would inherit by virtue of the portType extending ogsi:NotificationSource, will be set by the implementation to the list of 'notifications' that it supports. If the ManagedObject wants to support the two special notifications about notifications (described in section 4.4.3), it could do that too. Meta information about these notifications will be set in the notificationMetaInfo SDE which can then be queried by clients to obtain meta information about notifications. The client can then subscribe to notifications using either the pull or push mode. In the case of push modes, the Locator sink specified during the call to subscribe would be notified when the events occur. In response to a successful subscription clients receive a Locator to a BufferedNotificationSubscription, which queues notifications for the client and publishes the buffer itself as a SDE.

4.5 Web Services Management (WSM)

To be provided.

4.6 End-to-End

To be provided.

5 Glossary

conversation. A managed object that implements one or more of the *Conversation* management interfaces which represents one service's view of the state associated with a set of related messages.

Common Management Model (CMM). CMM is a working group in GGF to provide management interface definitions in Grid.

data model. See model, or information model.

event. An *event* is a change in the state of a resource or request for processing.

Grid. GGF standards, in particular OGSI and OGSA.

Grid service. Grid service is a service representation of a managed entity. It prescribes how to deal with state of managed entities through Service Data, and provides interfaces to manipulate the lifetime of the Grid Service. A Grid service does not further specify management, control or other port types, that is up to extensions like CMM.

hosting environment. The platform on which WSMF or Grid is running, e.g., J2EE, TIBCO, etc. For WSMF (as well as for CMM), an implementation of OGSI can be considered the hosting environment. Also called run-time environment.

infrastructure services and infrastructure layer. The Grid service extensions that deal with the lifecycle management of Grid services: creation, registration, discovery, deletion. In this document, infrastructure services include both OGSI lifecycle services and OGSA platform services.

information model. See model, or data model.

interface collection. An *interface collection* is a group of management interfaces that expose the management capabilities of a type of managed object.

lifecycle services. See services lifecycle layer.

managed entity. A managed entity is anything that is represented through a Grid service (in Grid) or Managed Object (in WSMF). A managed entity can be a hardware component, a software executable, a logical device such as an SLA contract, or an abstract notion such as 'solution.'

management interface. A management interface exposes management capabilities of a resource. A management interface is presented as a set of attributes, operations, and notifications to be accessed through a WSDL portType.

managed object. A managed object is a management representation of a resource. A managed object implements one or more management interfaces to provide a means to monitor and/or control the underlying resource.

management interface. A management interface exposes management capabilities of a resource. A management interface is presented as a set of attributes, operations, and notifications to be accessed through a WSDL portType.

meta model. A meta model specifies the available constructs in the model. A meta model does not specify instances of those constructs. For instance, the meta model for CIM has relationships in it, but does not specify what relationships are possible (that's left to the domain-specific information models, which have introduced hundreds of different relationships over the years.)

model. A model is a set of objects, properties, and their relationships.

notification. A notification is a message that is sent to or retrieved by one or more subscribers to inform them that an event has occurred.

platform services and platform services layer. The Grid service extensions in OGSA that deal with life cycle management of Grid services, in particular registry, authorization. See also infrastructure services.

resource. A resource is a component of a deployed environment. We preferably use managed entity instead of resource.

relation. A relation is a type of association between two managed objects.

relationship. A relationship specifies two managed objects and the relation to define how two specific objects are associated.

run-time environment. See hosting environment.

service. A managed object that implements the *Service* management interfaces which represents the management capabilities of a Web service. This Web service may be acting as the provider and/or the consumer of Web service messages.

service lifecycle layer. The elements in the OGSI specification that deal with lifecycle management of Grid services, in particular handle resolution, creation through factory. See also infrastructure services.

service representation. The service representation of a managed entity is the Grid service or WSMF managed object associated with the managed entity.

Simple Object Access Protocol (SOAP). SOAP is a transport protocol used in WSDL specifications to bind web service communication to a particular protocol (such as e-mail, http, remote procedure call)

subscriber. A subscriber is an entity that is interested in selected notifications from managed objects. These notifications contain information about the state change in a managed object.

virtual organization. A virtual organization (VO) is a group of individuals and/or organizations sharing a defined set of resources, under controlled conditions, for some collaborative purpose. VOs are often temporary, and their membership and the set of resources assigned to them may vary over time.

web service. A web service is defined by its WSDL specification, and uses SOAP as its transport protocol.

Web Services Description Language (WSDL). WSDL is a standard that provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered. The WSDL specification defines a language for describing the abstract functionality of a service as well as a framework for describing the concrete details of a service description. [From draft WSDL 1.2 Core Language Specification, June 2003]

Web Services Execution Environment (WSEE). A managed object that implements the WSEE management interfaces which encapsulates the management capabilities of a Web service execution environment.

Web Service Management Framework (WSMF). Set of WSDL1.1 compliant interface, attribute and port type definitions, for the purpose of managing any type of entity. WSMF also comes with a management information model for the web services domain.

6 References

1. **Web Service Management Framework.** N. Catania, P. Kumar, B. Murray, H. Pourheidari, W. Vambenepe, K. Wurster, http://devresource.hp.com/wsmf
2. **Dynamic Attributes and Meta Information.** B. Murray, H. Pourheidari, W. Vambenepe, available upon request
3. **The Physiology of the Grid—An Open Grid Services Architecture for Distributed Systems Integration.** I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf
4. **Open Grid Services Infrastructure.** S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, https://forge.gridforum.org/projects/ogsi-wg/document/Final_OGSI_Specification_V1.0/en/1/Final_OGSI_Specification_V1.0.pdf
5. **Open Grid Service Common Management Model Working Group.** https://forge.gridforum.org/projects/cmm-wg/document/Charter_21apr2003/en/1/Charter_21apr2003.pdf
6. **Web Services Description Language 1.1,** WSDL1.1, http://www.w3.org/TR/wsdl.
7. **Grid WSDL,** GWSDL, http://www.gridforum.org/ogsi-wg/.
8. **Web Services Description Working Group,** WSDL1.2 drafts, http://www.w3.org/2002/ws/desc/.
9. **GWSDL to WSDL 1.1 Transformation (GWSDL2WSDL1.1)**, T. Maguire, T. Sandholm, M. Williams, J. Joseph, http://www-unix.gridforum.org/mail_archive/ogsi-wg/2003/06/doc00000.doc.
10. **A developer's overview of OGSI and OGSI-based Grid computing,** J. Joseph, http://www-106.ibm.com/developerworks/grid/library/gr-ogsi/.
11. **GWSDL vs. WSDL1.2**, a grid forum e-mail thread, http://www-unix.gridforum.org/mail_archive/ogsi-wg/2003/02/msg00063.html.

7 Refactored WSMF Foundation (GWSDL)

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
Copyright (c) 2003 Hewlett-Packard Development Company, L.P.
PERMISSION TO COPY AND DISPLAY THIS WSMF PAPER, IN ANY MEDIUM WITHOUT FEE OR ROYALTY,
IS HEREBY GRANTED PROVIDED THAT YOU INCLUDE THE ABOVE COPYRIGHT NOTICE ON *ALL* COPIES
OF THIS WSMF SPECIFICATION, OR PORTIONS THEREOF, THAT YOU MAKE.

DISCLAIMER OF WARRANTEES. USER ACKNOWLEDGES THAT THE SPECIFICATION MAY HAVE
ERRORS OR DEFECTS AND IS PROVIDED "AS IS." HEWLETT-PACKARD MAKES NO EXPRESS
OR IMPLIED WARRANTIES OF ANY KIND WITH RESPECT TO THE SPECIFICATION, AND
SPECIFICALLY DISCLAIM THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE, EVEN IF THAT PURPOSE IS KNOWN TO HEWLETT-PACKARD.
```

```
NO LICENSE, EXPRESS OR IMPLIED, IS PROVIDED TO ANY PATENT OR TRADEMARK RIGHT.

LIMITATION OF LIABILITY. HEWLET-PACKARD SHALL NOT BE RESPONSIBLE FOR ANY LOSS
TO ANY THIRDS PARTIES  CAUSED BY USING THE SPECIFICATION IN ANY MANNER
WHATSOEVER. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT
OR ANY OTHER LEGAL THEORY, ARISING OUT OF ANY USE OF THE SPECIFICATION OR ANY
PERFORMANCE OF HEWLETT-PACKARD RELATED TO THIS SPECIFICATION. USER FURTHER
ACKNOWLEDGES THAT THE SPECIFICATION IS PROVIDED FOR EVALUATION PURPOSES ONLY,
AND USER ASSUMES ALL RISKS ASSOCIATED WITH ITS USE.
-->
<!-- CVS $Revision: 1.10.2.3 $ $Date: 2003/07/18 21:30:34 $ -->
<definitions name="WSMF-OGSI"
       targetNamespace="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-
ogsi-foundation"
       xmlns:wsmf="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
foundation"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
    xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
    xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData">
    <!--
    Todo: Add this extension to ServiceDataType later
    xmlns:ogsi-wsmf-sd="WSMF_OGSI_ServiceData.xsd"
    -->

    <import location="../../ogsi/ogsi.gwsdl"
           namespace="http://www.gridforum.org/namespaces/2003/03/OGSI" />
  <documentation>
This document creates an OGSI compliant version of WSMF's Managed Object Interface v2.0.

The ManagedObject interface collection supports the following event types with the
respective URIs:

http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/relationship-
added/

http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/relationship-
removed/
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/state-
changed/
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/attribute-
value-changed/
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/collection-
member-added/
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/event/collection-
member-removed/

The ManagedObject interface collection supports the following values for the State
attribute:
  State:
    http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/state/up/
    http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/state/down/
    http://devresource.hp.com/drc/specifications/wsmf/2003/07/managedobject/state/unknown/

The ManagedObject interface collection defines the following relations with the respective
URIs:
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/relations/manager-of-
collection/
  http://devresource.hp.com/drc/specifications/wsmf/2003/07/relations/member-of-collection/
  </documentation>

  <types>
```

```
         <s:schema
targetNamespace="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
foundation"
       attributeFormDefault="qualified"
        elementFormDefault="qualified">

      <s:include schemaLocation="WSMF-Foundation.xsd" />

      <s:complexType name="SelectExpressionType">
        <s:restriction base="s:string" />
      </s:complexType>
      <s:element name="Select" type="wsmf:SelectExpressionType" />

      <s:element name="Invoke">
        <s:complexType>
          <s:sequence>
               <!-- If the Select element is not present, all collection members are
selected. -->
            <s:element ref="wsmf:Select" minOccurs="0" />
            <s:element name="Interface" type="s:QName" />
            <s:element name="Name" type="s:NMTOKEN" />
            <s:element name="ArgumentList" minOccurs="0">
              <s:complexType>
                <s:sequence>
                    <!-- TODO: Can this be more specifically defined? It can be an argument
value
                    which may be complex (e.g. <MyArg><A>something</A><B>else</B></MyArg>).
-->
                  <s:any minOccurs="1" maxOccurs="unbounded" processContents="lax" />
                </s:sequence>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="InvokeResponse">
        <s:complexType>
          <s:sequence>
            <s:element ref="wsmf:ManagedObjectResponseInformationList" />
          </s:sequence>
        </s:complexType>
      </s:element>

    </s:schema>
  </types>

  <message name="InvokeInput">
    <part name="document" element="wsmf:Invoke" />
  </message>
  <message name="InvokeOutput">
    <part name="document" element="wsmf:InvokeResponse" />
  </message>

  <!-- Note that right now, the Identity portType contains nothing more than
  what is already present in the base GridService portType. However, we are
  retaining this because we may introduce newer SDEs in the future -->

  <gwsdl:portType name="ManagedObjectIdentity" extends="ogsi:GridService">
         <sd:serviceData name="ID"
               type="wsmf:EntityReferenceType"
               minOccurs="1"
               maxOccurs="1"
               mutability="mutable"
               modifiable="false"
               nillable="false" />
  </gwsdl:portType>

  <gwsdl:portType name="Configuration" extends="wsmf:ManagedObjectIdentity">
         <sd:serviceData name="Name"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="true"
              nillable="false" />
              <!--
              Policy="<wsp:Policy>...</wsp:Policy>"
              -->
       <sd:serviceData name="Type"
              type="wsmf:ManagedObjectType"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="true"
              nillable="false" />
       <sd:serviceData name="Description"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="true"
              nillable="true" />
       <sd:serviceData name="Owner"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="true"
              nillable="false" />
       <sd:serviceData name="Vendor"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="true"
              nillable="true" />
       <sd:serviceData name="Version"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="false"
              nillable="false" />
       <sd:serviceData name="ManagedObjectVersion"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
              mutability="mutable"
              modifiable="false"
              nillable="false" />
       <sd:serviceData name="CreatedOn"
              type="s:dateTime"
              minOccurs="1"
              maxOccurs="1"
              mutability="static"
              modifiable="false"
              nillable="false" />
       <sd:serviceData name="HostName"
              type="s:string"
              minOccurs="1"
              maxOccurs="1"
```

```
             mutability="mutable"
             modifiable="true"
             nillable="true" />
      <sd:serviceData name="ManagedObjectHostName"
             type="s:string"
             minOccurs="1"
             maxOccurs="1"
             mutability="mutable"
             modifiable="true"
             nillable="true" />

  </gwsdl:portType>

  <gwsdl:portType name="Monitoring" extends="wsmf:ManagedObjectIdentity" >
      <sd:serviceData name="State"
             type="wsmf:StateType"
             minOccurs="1"
             maxOccurs="unbounded"
             mutability="mutable"
             modifiable="true"
             nillable="true" />
             <!--
             Policy="<wsp:Policy>...</wsp:Policy>"
             -->
      <sd:serviceData name="SupportedStates"
             type="wsmf:StateListType"
             minOccurs="1"
             maxOccurs="unbounded"
             mutability="mutable"
             modifiable="true"
             nillable="true" />

  </gwsdl:portType>

  <gwsdl:portType name="Discovery" extends="wsmf:ManagedObjectIdentity" >
        <sd:serviceData name="Relationship"
             type="wsmf:RelationList"
             minOccurs="0"
             maxOccurs="unbounded"
             mutability="mutable"
             modifiable="true"
             nillable="true" />
      <sd:serviceData name="SupportedRelations"
             type="wsmf:RelationshipListType"
             minOccurs="0"
             maxOccurs="unbounded"
             mutability="mutable"
             modifiable="true"
             nillable="true" />
  </gwsdl:portType>

  <gwsdl:portType name="Control"  extends="wsmf:ManagedObjectIdentity" >
       <sd:serviceData name="State"
             type="wsmf:StateType"
             minOccurs="0"
             maxOccurs="unbounded"
             mutability="mutable"
             modifiable="true"
             nillable="true" />
  </gwsdl:portType>

  <gwsdl:portType name="Collection" extends="ogsi:ServiceGroupRegistration" >
      <operation name="Invoke">
     <input message="wsmf:InvokeInput" />
     <output message="wsmf:InvokeOutput" />
    </operation>
```

```
 </gwsdl:portType>

</definitions>
```

8 Refactored WS-Events (GWSDL)

```
<definitions
    name="WSEvents-OGSI"
    targetNamespace="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
events"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
    xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:events="http://devresource.hp.com/drc/specifications/wsmf/2003/07/wsmf-ogsi-
events">

    <import
        location="../../ogsi/ogsi.gwsdl"
        namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

        <types>
            <xsd:schema>
                <!-- The base notification schema -->
                <xsd:complexType name="NotificationType">
                    <xsd:sequence>
                          <xsd:element name="Originator" type="ogsi:LocatorType"/>
                          <xsd:element name="UUID"  type="xsd:anyURI"/>
                          <xsd:element name="extensibilityElement"
                                       type="ogsi:ExtensibilityType"/>
                          <xsd:any minOccurs="0" maxOccurs="unbounded"
                                   namespace="##other"
                                   processContents="lax"/>
                    </xsd:sequence>
                    <xsd:attributeGroup ref="ogsi:LifeTimePropertiesGroup"/>
                </xsd:complexType>

                <!-- The schema for representing meta info about notifications -->
                <xsd:complexType name="NotificationMetaInfoType">
                    <xsd:sequence>
                        <xsd:element name="Notification" type="xsd:QName"/>
                        <xsd:element name="SchemaLocation" type="xsd:anyURI"/>
                        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>

                        <xsd:element name="Causes"
                                     type="events:NotificationMetaInfoListType"
                                     minOccurs="0"/>

                        <xsd:any minOccurs="0" maxOccurs="unbounded"
                                 namespace="##other"
                                 processContents="lax"/>
                    </xsd:sequence>
                    <xsd:attributeGroup ref="ogsi:LifeTimePropertiesGroup"/>
                    <xsd:anyAttribute namespace="##other" processContents="skip"/>
                </xsd:complexType>

                <!-- A list of meta information -->
                <xsd:complexType name="NotificationMetaInfoListType">
                    <xsd:sequence>
                        <xsd:element
                            name="NotificationMetaInfo"
                            type="events:NotificationMetaInfoType"
                            minOccurs="0" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>

                <!-- Some types used for extensibility elements -->
                <xsd:complexType name="FilterType">
                    <xsd:choice>
                        <xsd:element name="Filter" type="xsd:QName"/>
                        <xsd:any processContents="lax"/>
                    </xsd:choice>
                </xsd:complexType>

                <xsd:simpleType name=”SubscriptionModeType”>
                     <xsd:restriction base=”xsd:token”>
                         <xsd:enumeration value=”push”/>
                         <xsd:enumeration value=”pull”/>
                         <xsd:enumeration value=”pushAndPull”/>
                     </xsd:restriction>
                </xsd:simpleType>

                <!-- Extensibility element for supporting meta info queries on
                     notifications -->
                <!-- Only supported on the notificationMetaInfo SDE of
                     BufferedNotificationSource-->
                <xsd:element name="GetNotificationMetaInfo">
                    <xsd:complexType name="GetNotificationMetaInfoType">
                        <xsd:sequence>
                            <xsd:element name="Xpath" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>

                <!-- Extensibility element for supporting richer subscribe variants -->
                <xsd:element name="NotificationSubscriptionExpression">
                    <xsd:complexType name="NotificationSubscriptionExpressionType">
                        <xsd:complexContent>
                          <xsd:extension base="ogsi:SubscribeByNameType">
                              <xsd:sequence>
                                  <xsd:element name="Filter" type="events:FilterType"/>
                                  <xsd:element name=”SubscriptionMode”
                                               type=”events:SubscriptionModeType”/>
                              </xsd:sequence>
                          </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>

                <!-- Extensibility elements for pull variants -->
                <xsd:element name="GetNotificationsSinceDate">
                    <xsd:complexType name="GetNotificationSinceDateType">
                        <xsd:sequence>
                            <xsd:element name="Date" type="xsd:dateTime"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>

                <xsd:element name="GetNotificationsSinceUUID">
                    <xsd:complexType name="GetNotificationsSinceUUIDType">
                        <xsd:sequence>
                            <xsd:element name="UUID" type="xsd:anyURI"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>

                <xsd:element name="GetNotificationsRangeByDate">
```

```
                    <xsd:complexType name="GetNotificationsRangeByDateType">
                        <xsd:sequence>
                            <xsd:element name="BeginDate" type="xsd:dateTime"/>
                            <xsd:element name="EndDate" type="xsd:dateTime"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name=”NotificationList”>
                    <xsd:complexType>
                      <xsd:sequence>
                          <xsd:element name=”Notification” type=”events:NotificationType”
                                       minOccurs=”0” maxOccurs=”unbounded”/>
                      </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:schema>
        </types>

       <message name="PullByDateInput">
              <part name="document" element="events:GetNotificationsSinceDate" />
       </message>
       <message name="PullByUUIDInput">
               <part name="document" element="events:GetNotificationsSinceUUID" />
       </message>
       <message name="PullByDateRangeInput">
               <part name="document" element="events:GetNotificationsRangeByDate" />
       </message>
       <message name=”PullOutput”>
              <part name=”document” element=”events:NotificationList”/>
       </message>



        <gwsdl:portType name="BufferedNotificationSource"
                        extends="ogsi:NotificationSource">
            <sd:serviceData name="notificationMetaInfo"
                            type="events:NotificationMetaInfoType"
                            minOccurs="0"  maxOccurs="unbounded"/>
            <sd:serviceData name="NewEventTypeNotification"  type="events:NotificationType"
                            minOccurs="0"  maxOccurs="1" />
            <sd:serviceData name="EventTypeUpdatedNotification"
                            type="events:NotificationType"
                            minOccurs="0"  maxOccurs="1" />

            <sd:staticServiceDataValues>
                <!-- extensibility element for supporting meta info queries on
                     notifications -->
                <ogsi:findServiceDataExtensibility
                      inputElement="events:GetNotificationMetaInfo"/>

                <!-- extensibility element for supporting richer subscribe variants -->
                <ogsi:subscribeExtensibility
                      inputElement="events:NotificationSubscriptionExpression"/>
            </sd:staticServiceDataValues>
        </gwsdl:portType>

        <gwsdl:portType name="BufferedNotificationSubscription"
                        extends="ogsi:NotificationSubscription">
            <operation name=”PullByDate”>
               <input message=”events:PullByDateInput”/>
               <output message=”events:PullOutput”/>
            </operation>
            <operation name=”PullByUUID”>
               <input message=”events:PullByUUIDInput”/>
               <output message=”events:PullOutput”/>
            </operation>
            <operation name=”PullByDateRange”>
               <input message=”events:PullByDateRangeInput”/>
               <output message=”events:PullOutput”/>
            </operation>
            <sd:serviceData  name="notificationBuffer" type="events:NotificationType"
                             minOccurs="0"   maxOccurs="unbounded"/>
            <sd:serviceData  name="bufferSize" type="xsd:int" />

            <!-- extensibility elements for supporting the different WS-Events pull
                 variants -->
            <sd:staticServiceDataValues>
                <ogsi:findServiceDataExtensibility
                       inputElement="events:GetNotificationsSinceDate"/>
                <ogsi:findServiceDataExtensibility
                       inputElement="events:GetNotificationsSinceUUID"/>
                <ogsi:findServiceDataExtensibility
                       inputElement="events:GetNotificationsRangeByDate"/>
            </sd:staticServiceDataValues>
        </gwsdl:portType>

</definitions>
```