E.Stokes, IBM
N. Butler, IBM
17 February 2003

Common Resource Model (CRM)

Status of This Memo

This document provides information to the Grid community regarding the OGSA Common Resource Model specification. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

Manageable resources are exposed as Grid services in OGSA. This specification defines a Common Resource Model to describe the structure of a manageable resource as an OGSA service. The Common Resource Model builds upon existing resource model standards such as DMTF's Common Information Model meta-model (http://www.dmtf.org/standards/standard_cim.php) and IETF's Management Information Base (MIB) (http://www.ietf.org/RFCxxxx). It is assumed that the reader is familiar with the Global Grid Forum's *Grid Service Specification* (http://www.globalgridforum.org/ogsi-wg/drafts/GS_Spec_draft15-2003-02-13.pdf) and the W3C's XML Schema specifications (http://www.w3c.org/TR/xmlschema-1 and http://www/w3c.org/TR/xmlschema-2). Concepts from IBM's Solution Management work are also incorporated.

Contents

1. 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 is not yet done consistently.
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/OGSI”
gwsdl	"http://www.gridforum.org/namespaces/2003/gridWSDLExtensions"
crm	"http://www.gridforum.org/namespaces/2003/17/crm"
wsdl	"http://www.w3.org/2002/07/wsdl"
sd	"http://www.gridforum.org/namespaces/2003/serviceData"
http	"http://www.w3.org/2002/06/wsdl/http"
xsd	"http://www.w3.org/2001/XMLSchema"
xsi	“http://www.w3.org/2001/XMLSchema-instance”

Unresolved issues with the specification are interspersed in appropriate locations through this specification, are highlighted in yellow.

Some of the XML in this document has not been checked with a parser yet, so may be syntactically incorrect.

2. Overview

In an IT system there are many entities that have some form of state (runtime state, configuration, etc) and on which management operations can be performed. These are known in this specification as *manageable resources*. Manageable resources 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), and also to transient things such as print jobs.

The purpose of this specification is to define how to model the manageability of these manageable resources. *Manageability* defines information that is useful for managing a resource. It details the aspects of a resource that support management including the instrumentation that allows a management tool to interact with a resource. *Management* is the active process of monitoring, modifying, and making decisions about a resource including the capabilities that use manageability information to perform activities or tasks associated with managing IT resources.

The CRM describes how the management interface of a manageable resource is exposed through Web services. A manageable resource is a stateful object – its runtime state, configuration, etc – and so the web service that represents it is a stateful web service. The CRM uses the OGSA grid service model for stateful services – every manageable resource is represented by a grid service: each resource instance has a unique identifier, its state data is exposed via Service Data elements and its operations are web service operations. The interface to a manageable resource (its Service Data elements and its operations) is described using the Web Services Description Language (WSDL).

The grid service that represents a manageable resource allows the resource to be managed throughout its runtime lifecycle, allowing it to be stopped and started, for example. For this reason, this grid service is always separate from the resource, even if the resource is itself a web or grid service. The grid service that represents the resource is, at least logically, created at the time the resource is created and removed when the resource is removed. The CRM addresses only the grid services that represent the manageable resources, not other web service interface(s) that the resource may expose for some domain-specific functional use. The identifier used in the CRM for the manageable resource is the Grid Service Handle (GSH) of the grid service that provides the management interface. This is illustrated in Figure 1.

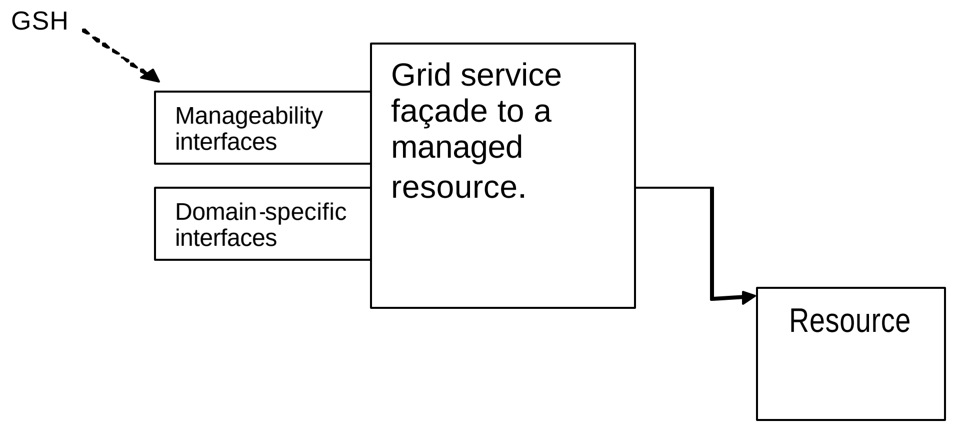
GSH
Manageability interfaces
Domain-specific interfaces
Grid service façade to a managed resource.
Resource

Figure 1: Manageable resource

This specification defines the OGSA Common Resource Model (CRM), a model of manageable resources as OGSA services. The Common Resource Model uses existing resource model standards such as DMTF's Common Information Model [cimschema] as a base of information and experience. WSDL [wsdl], GSDL which is part of the OGSi specification [gsspec], and XSD schema [xmldatatypes, xmlstructures] are used to define a manageable resource as a service.

There are three main aspects to manageability in OGSA CRM:
- XML schema (XSD) for modeling resource manageability information
- a collection of manageability portTypes
- guidelines for modeling resources

These are introduced in the following sections.

2.1 Schema

A resource's manageability information is modeled using XML schema. Extensions in the form of additional data types and XML attributes are defined to allow those manageable resources to provide additional information to the application or management tool in order to be better managed.

The additional data types defined are:
- counter
- gauge

These are refinements of data type integer to convey the meaning of the integer data as well as the range of valid values. These are described in Section 11.2.

The XML attributes defined are:

- version
- deprecated
- experimental
- units
- valid
- changeable
- volatile
- latency

The first three attributes listed (version, deprecated and experimental) provide general and useful versioning capabilities[1], and are described in Section 10.1. The fourth (units) communicates the unit of measure (e.g. kilobytes) of the data so it can be meaningfully be processed/displayed an application or management tool. This attribute is described in Section 10.2. The last four (valid, changeable, volatile and latency) are for use in describing various lifecycle characteristics of a resource, and are described in Section10.3.

2.2 Manageability portTypes

The interface to an OGSA service is described in one or more port types. The port type for a manageable resource is analogous to a class with properties and methods. The attributes of the resource are expressed as service data, and its methods are expressed as operations.

Port types that define part of the manageability interface for a manageable resource are called manageability port types. WSDL 1.2 introduces port type inheritance, which allows one port type to say that it extends one or more other port types: in this way the new port type also includes all the service data and operations defined in the extended port types. The complete interface for a manageable resource is thus defined in a single port type that extends various other port types to include all the management function of the resource. While WSDL does not require that there is one single most derived port type, we make this restriction so that port type can be used to identify the type of a manageable resource. This port type is called the manageable resource type.

It is an objective of resource modeling according to the CRM to factor out and define port types for commonly used related functions that may be applied to multiple types of resource: for example, start, stop and restart are a set of operations that will be used frequently. These sorts of port types are termed *canonical port types*, and using them wherever possible in the modeling of manageable resources allows management applications to more easily provide consistent management across resource types.

The port type for a manageable resource may, of course, include canonical port types, for common functions, and its own operations, service data and port types for resource specific functions. A port type that contains operational operations (such as start, stop, resume, pause) is an example of a canonical port type.

One of the port types that a Grid service, and hence manageable resource services, must implement is the GridService port type. This provides some basic service data elements and operations – such as FindServiceData, which allows the caller to retrieve selected pieces of service data.

The CRM defines a single new port type, BaseManageableResource.

[1] These attributes will be discussed for inclusion in the Grid services spec [gsspec] post V1.0 of the GS Spec, as they really have applicability beyond management and manageability in the CRM.

The BaseManageableResource port type contains service data elements that must be implemented by each and every manageable resource. This port type extends the GridService port type and adds service data that allows a managed resource to describe its lifecycle data, relationships to other resource instance and types, to which resource group type it belongs, and identify which service data elements are likely to be used for finding this resource.

The CRM also makes use of other Grid service port types defined in the Grid Service Specification. Most notably the ServiceGroup port type is used to help in the location of fine-grained manageable resources: for details see Section 7.

3. Model Design

3.1 Structure Description

This section describes the structure of a service that represents a resource. The structure uses the Grid service specification as its base and then extends it to express what's needed for a resource to be expressed as a service. Concepts are borrowed from DMTF's CIM given the extensive resource modeling and meta-model that exist.[2] The CRM is not a strict algorithmic mapping for any one model. But existing models are mappable to CRM; those existing models can be service implementations of CRM.

The model does not address any specifics with respect to the binding and service implementation constructs.

3.1.1 Structure of a Grid Service

Below is the basic outline of a Grid Service. Refer to the *Grid Service Specification* [gsspec] for the details. [Note: the definition is included below for easy reference and to show the relationships of the wsdl/gsdl fragments discussed in the grid service specification. There may be multiple bindings for a given interface (port type) and multiple service implementations for a given binding. Bindings provide a variety of transport, QoS, and security opportunities upon which to deploy the grid service that models the resource.

<Insert outline of a Grid service here – or get the Grid service spec authors to pull the fragments in the grid spec into the outline of a grid service for easy reference.>

3.1.2 Structure of a Resource as a Grid Service

The primary components of the Common Resource Model (CRM) are data types, additional XML attributes, service data and their associated service data descriptions, and port types.

In general, a resource type is represented as a port type, managed properties of a resource are represented as service data of that port type, and methods of a resource are represented as operations of that port type.

A port type may have one or more bindings for access. A binding may have one or more service implementations.

Resources defined using the CRM are generally coarser-grained services than that which a granular or normalized resource model defines. That is, a service is fairly self-contained and is

[2] Need to check SNMP, JMX, and CMIP to see any concepts from those models make sense to incorporate here.

composed of pieces of normalized resource models and contains a few relationships to other services.

For example, a disk resource may have many discrete model parts such as a model for the manageability characteristics of the disk, a model for the set of disk error statistics, a model for its disk metrics, and a relationship model to a computing system resource. When the disk resource is expressed as a service, the service is composed of (aggregate) of the manageability characteristics, error statistics, and metrics models and expresses a contained relationship to the computer system.

See Section 12 for hints/tips on modeling manageable resources as grid services.

4. Service Data

Properties of a manageable resource are expressed as service data – this includes the configuration of the manageable resource. The Grid Service specification [gsspec] defines the XML attributes and operations of service data (FindServiceData for get/query and SetServiceData for set).

If an application is permitted to use the SetServiceData operation to change the value(s) of a property, then the service data that represents that property has the XML attribute modifiable set to 'true'. Otherwise, the service data is read-only from the point of view of SetServiceData. The SetServiceData operations are used to modify to those properties specified as modifiable (there may also be operations specific to the manageable resource that modify that resource's service data elements).

FindServiceData and SetServiceData are part of the GridService port type. Every manageable resource extends from the GridService port type so service data and its get/set operations are available for every manageable resource.

5. Base Manageable Resource Port Type

The BaseManageableResource port type contains service data elements that must be implemented by each and every manageable resource. This port type extends the GridService port type and adds service data that allows a managed resource to describe its global manageability information: lifecycle data, instance and type relationships to other resources, to which resource group it belongs, and which service data elements are likely to be used for finding this resource.

The BaseManageableResource port type is not instantiable by itself. Every manageable resource must extend this port type to describe the aggregation of its specific manageability information, its GridService port type behavior, and its global manageability information.

The WSDL for BaseManageableResource port type can be found in Appendix B.

5.1 BaseManageableResource: Service Data Declarations

A number of service data elements are defined in this specification, and all are part of the BaseManageableResource port type.

- lifecycleModel and currentLifecycle state are related to the lifecycle management of the service.
- serviceGroupType and searchProperty are provided to help in the location of manageable resources.

  - relatedInstance and relatedType are used to describe relationships and dependencies between resource instances or resource types.

- lifecycleModel

  Describes the lifecycle states that a resource may be in or pass through, along with any associated sub-states. For details see Section 6.2.

- currentLifecycleState

  Returns the current state that the resource is in, and sub-state if applicable. For details see Section 6.3.

- serviceGroupType

  The port type of the manageable resource that provides the ServiceGroup function for manageable resources of this type. This is a static value that is set in the WSDL for the manageable resource type. Its value must only be set in the most derived port type, i.e. the one that represents the manageable resource type. If this value is not set, then there is no specific ServiceGroup for this type.

  Should it be possible to specify more than one serviceGroupType, indicating that a resource type might be found in more than one type of resource group?

```
<sd:serviceData
    name="serviceGroupType"
    type="xsd:QName"
    minOccurs="0"
    maxOccurs="1"
    mutability="static" />
```

- searchProperty

  Zero or more service data elements that are likely to be used for searching for a manageable resource, and are thus worth caching in the manageable resource registry. This is a static SDE, so values can only be specified in the port type definitions. Values are likely to be specified at various points in the port type inheritance hierarchy. For example a base port type might specify a property, such as IP address, that must be provided by all manageable resources, and that it is to be a search property for all manageable resources. It would define the IP address SDE and specify this property in a searchProperty element.
  Note that searchProperty is a separate service data element, rather than an attribute that can be applied to any service data element. This allows port types to add search properties that come from other port type definitions.

```
<sd:serviceData
    name="searchProperty"
    type="xsd:QName"
    minOccurs="0"
    maxOccurs="unbounded"
    mutability="static" />
```

- relatedInstance

  Expresses the relationships that a manageable resource has with other manageable resources, by relating the service instances.

```
   <sd:serviceData
        name="relatedInstance"
        type="crm:relatedInstance"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="mutable"/>
```

```
  <xsd:complexType name="relatedInstance">
    <xsd:sequence>
      <xsd:element name="relationshipType" type="xsd:string"/>
      <xsd:choice>
        <xsd:annotation>
          <xsd:documentation>
            The source or sink is the grid service handle of the related service instance.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:element name="source" type="gsdl:serviceLocator"/>
        <xsd:element name="sink" type="gsdl:serviceLocator"/>
      </xsd:choice>
      <xsd:element name="requirement" minOccurs="0" maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            This optional element is used to specify a list of requirements that this
            service instance has upon the related service instance (identified as the
            source or sink).
          </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:annotation>
            <xsd:documentation>
            <xsd:documentation>
              The requirement identifies the specific nature of the dependency.  The details
               of the schema are under discussion and will be added later: until then an
              extension point allows any XML content to be used.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

<u>relatedInstance example</u>

A database, “myDatabase”, hosts a database table, “myTable”.

The relationship service data element in the service for “myDatabase” will contain an element like this:

```
<sd:serviceDataValues>
   …
   <crm:relatedInstance>
      <crm:relationshipType>hosts</crm:relationshipType>
      <crm:sink>..GSH of myTable..</crm:sink>
   </crm:relatedInstance>
   …
</sd:serviceDataValues>
```

The relationship service data element in the service for “myTable” will contain an element like this:

```
<sd:serviceDataValues>
   …
   <crm:relatedInstance>
      <crm:relationshipType>hosts</crm:relationshipType>
      <crm:source>..GSH of myDatabase..</crm:source>
   </crm:relatedInstance>
   …
</sd:serviceDataValues
```

- relatedType

  Expresses the relationships that exist between manageable resource (or other service) types. The service type is identified by a port type that it implements. In the case of manageable resources, this would be the most derived port type that describes the manageability interface of the resource type.

```
      <sd:serviceData
            name="relatedType"
            type="crm:relatedType"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability=”static”/>
```

```
   <xsd:complexType name="relatedType">
      <xsd:sequence>
         <xsd:element name="relationshipType" type="xsd:string"/>
         <xsd:choice>
            <xsd:annotation>
               <xsd:documentation>
                  The source or sink is qname of the related port type.
               </xsd:documentation>
            </xsd:annotation>
            <xsd:element name="source" type="xsd:QName"/>
            <xsd:element name="sink" type="xsd:QName"/>
         </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>
```

relatedType example

In the database example, database tables are hosted by databases, and databases are hosted by database servers.

The GSDL definition of the Table port type would contain the following initial value of a relatedType SDE:

```
<gwsdl:portType name="Table">
   …
   <sd:staticServiceDataValues>
      <crm:relatedType>
         <relationshipType>hosts<relationshipType>
         <source>database:Database</source>
      </crm:relatedType>
   </sd:staticServiceDataValues>
   …
</gwsdl:PortType>
```

The GSDL definition of the Database port type, would contain the following initial value of a relatedType SDE:

```
<gwsdl:portType name="Database">
   …
   <sd:staticServiceDataValues>
      <crm:relatedType>
         <relationshipType>hosts<relationshipType>
         <source>database:DatabaseServer</source>
      </crm:relatedType>
   </sd:staticServiceDataValues>
   …
</gwsdl:PortType>
```

5.2 BaseManageableResource: Operations

The GroupEntry portType defines no operations. The operations inherited from the GridService portType SHOULD be used to query the SDEs of the BaseManageableReource

6. Lifecycle

Resources exist from the time they are created until they are destroyed and are in a variety of states in between. Resources can be (and in most cases, are) managed differently at different stages of their lifetime. Because different kinds of resources and web services will have different lifecycles and states, there is no one single lifecycle model that can be defined. Instead, we define a generic method that allows models to be defined to match the needs of a resource. We recognize that classes or types of resources will use the same model, in fact this is strongly encouraged where possible. So, in Appendix A, we define one such typical model that resources can use. We expect other basic lifecycle models to be defined.

Lifecycle is a set of states that a resource can be in and the valid transitions between those states. This specification currently only addresses describing the valid states, and not the transitions and the operations that effect those transitions. The lifecycle XML attributes describe the meaningful lifecycle state for service data and operations. An application or management tool uses the lifecycle attributes to obtain information about lifecycle state to better manage that service. To incorporate the concept of lifecycle into a Grid service, the following are defined:

- XSD that describes the structure of a lifecycleState element
- A service data element that defines the lifecycle model used by the resource
- A service data element that holds the current lifecycle value of the resource
- XML attributes that describe the lifecycle characteristics of the service for use by an application or management tool, specifically the changeability, validity, volatility, and latency of the pieces comprising the service. These attributes are defined in Section 10.3.

Note: the Grid Service specification definitions of lifetime declaration properties (goodFrom, goodUntil, and availableUntil) are insufficient for managing resources. These three properties are intended to refer to the validity of values that have been cached somewhere in the system. These are legal to appear in values associated with CRM service data elements. <We need to offer guidance on the use of these temporal attributes and also on the implications of soft-state.>

6.1 Lifecycle state

The lifecycle model of a resource type contains one or more states, and each state can have zero or more sub-states that provide additional information about that state. The states and sub-states are described using lifecycleState elements within the lifecycleModel service data element of BaseManageableResource. The definition of lifecycleState element is:

```
<xsd:element name="lifecycleState" type="lifecycleStateType">

<xsd:complexType name="lifecycleStateType">
  <xsd:sequence>
    <xsd:element name="subState" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:NCName"></xsd:attribute>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:NCName"/>
</xsd:complexType>
```

Its mutability is static; the lifecycle states are defined once when the resource's port type is defined and are the same for all instances of a resource.

6.2 Lifecycle model

The lifecycle model of a resource type is defined in the lifecycleModel service data element. This contains the definition of the lifecycle states and sub-states.

A management application needs to understand how a state model works – the possible transitions and the operations that go with those. Those are currently described through the words of a specification and not a formal state model: for this reason it is not practical to allow arbitrary new states or sub-states to be added to an existing model. This is the reason for introducing a lifecycleModel service data element as a container for the states, rather than just using a lifecycleState service data element: it is only possible to replace the entire model at once. Its mutability is static; the lifecycle states are defined once when the resource's port type is defined and are the same for all instances of a resource.

We need to consider how to specify operations associated that transition the resource from one state to its next expected state and/or add metadata to operations such that their relationship or effect on lifecycle state is well understood.

The definition of lifecycleModel is:

```
<xsd:complexType name="lifecycleModelType">
  <xsd:sequence>
    <xsd:element ref="crm:lifecycleState" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
```

```
</xsd:complexType>

<sd:serviceData
  name="lifecycleModel"
  type="crm:lifecycleModelType"
  minOccurs=”1”
  maxOccurs=”1”
  nillable=”true”
  mutability=”static”/>
```

So, for example, one model that a resource might use (from Appendix A) is:

```
<sd:staticServiceDataValues>
  <lifecycleModel>
    <crm:lifecycleState name="down">
      <subState name="restartable"/>
      <subState name="recovered"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="starting">
      <subState name="OK"/>
      <subState name="error"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="up">
      <subState name="idle"/>
      <subState name="busy"/>
      <subState name="degraded"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="stopping">
      <subState name="OK"/>
      <subState name="error"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="failed">
      <subState name="dependencyFailure"/>
      <subState name="nonrecoverableError"/>
    </crm:lifecycleState>
  </lifecycleModel>
</sd:staticServiceDataValues>
```

6.3 Current lifecycle state

The currentLifecycleState service data element contains the current state, and sub-state if applicable, that the resource is in.

```
<sd:serviceData
  name="currentLifecycleState"
  type=”lifecycleStateType"
  minOccurs=”1”
  maxOccurs=”1”
  mutability=”mutable”/>
```

An example is

```
<sd:staticServiceDataValues>
  <crm:currentLifecycleState name=”Up”>
    <subState name=”OK”/>
  </crm:currentLifecycleState>
```

```
</sd:staticServiceDataValues>
```

7. Identifying and finding manageable resources

Manageable resources are represented by Grid services. Each resource instance has an associated service instance and an associated Grid Service Handle. The GSH provides the unique identifier for the resource. It can be passed around within and between management applications, and used to look up the Grid Service Reference (GSR) whenever it is required to interact with the resource.

In theory, it should be sufficient to leave identity at that: it should be possible to use any valid form of GSH, and use existing service data, and service data queries to find instances of resources. However, the number of manageable resources and the requirements to be able to search for specific resources suggest that additional information should be provided is provided to help in finding those resources.

7.1 Resource groups

An important requirement for a management system is to be able to find instances of the manageable resources in the system. The number of resource instances that may exist in a system can range from a few thousand, for a small installation, up to many, many millions in an enterprise installation. When used across the Grid, the number of resource could be even higher. To have a registry that holds the handles for every resource in the system would be impractical: it would be very large, and it would be updated frequently as resources were created and removed from the system.

So the problem is how to locate the fine-grained resources in a system. This is not a problem that is limited to services that represent manageable resources, and work is underway elsewhere to address this, using Service Domains, supported by the OGSI notion of a ServiceGroup (see the OGSI spec). This will result in some form of hierarchy or federation of registries, but this still does not address the fine-grained resources where it is not practical to place references in a separate registry.

However, there is a natural grouping of resources at a local level (within one machine or a small group of machines) where one relatively coarse grained resource provides some form of container for one or more types of fine grained resource. These “container” resources will often have some responsibility for management of the objects within them. For example, a database server hosts databases, and the databases in turn host tables. These groups will typically be defined by the boundaries of a software or hardware product, or component of a product, rather than spanning across products.

CRM exploits this natural grouping to assist with the task of locating resources. The container resource and its contained resources form a **resource group**. The Grid Service Specification defines the ServiceGroup Grid service port type that is used to maintain information about a group of service instances, having an entry SDE for each member of the group. The CRM uses ServiceGroups to hold information about the manageable resources within a resource group.

This is illustrated in **Error! Reference source not found.** with an example of a database server that contains two databases, which each contain some database tables. The database server is the container resource type and provides the ServiceGroup implementation for the databases and tables it contains. There is an entry SDE in the ServiceGroup for each of the databases and tables.

ServiceGroup defines two service data elements that are relevant to resource groups, entry and membershipContentRule. The entry SDE includes a content element that contains information about the member service. In a resource group, the content will contain the values of the SDEs from the member resource that are identified by the searchProperty SDE of the type of the resource.
The membershipContentRule SDE type identifies the port types that may be members of the ServiceGroup, along with the definition of the content associated with each port type. In a resource group, the port types will include those that represent the types of manageable resources within the group. In the database example, there would be membershipContentRule elements for each for each of db:Database and db:Table.

More detail is needed in this specification describing how ServiceGroups are used for resource groups, in particular in defining the content model to be used. This may result in the definition of a new port type that extends ServiceGroup.

When a client wants to find one of the fine-grained resources, they have to perform the search in two steps:

1. Find ServiceGroups that have members of the type of the fine-grained resource type. For example, to find instances of db:Table, the client must first find instances of db:DatabaseServer.
   This can be done in one of two ways:
   1. Search for ServiceGroup services with a membershipContentRule SDE where the memberInterface element contains the value 'db:Table'
   2. Look at the ServiceGroupType SDE in the WSDL for db:Table, which will be set to 'db:DatabaseServer', and then find instances of db:DatabaseServer.
2. Use the FindServiceData operation on the ServiceGroup to find entries that match the required criteria – i.e. have the required values in the content.

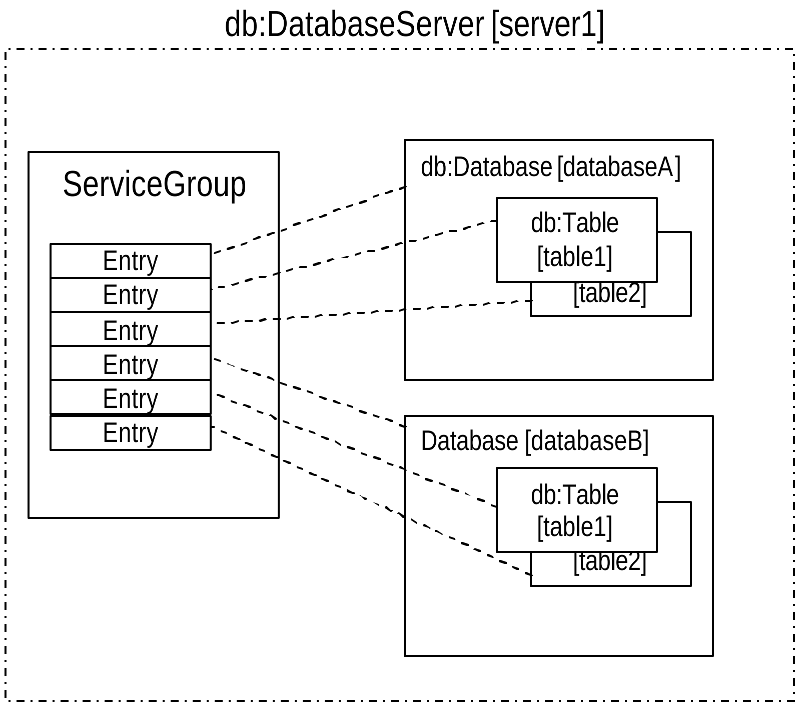

Figure 2: Resource group

7.2 Manageable resource types

Every manageable resource instance has a manageable resource type. The type identifies the management interface of the resource, defined through a single unique port type. This port type uses the WSDL 1.2 port type extension mechanism to aggregate together the individual port types that make up the management interface, as well as adding additional service data elements and operations that are specific to the resource type. This is analogous to defining the interface to a Java^{tm} object by making it implement a number of interfaces that define the individual parts of the overall interface.

A basic manageable resource type must extend the CRM BaseManageableResource port which is itself an extension of the OGSA GridService port type. Some container resource types will also extend the CRM ServiceGroup port type to make the resources that it contains locatable. This is illustrated in Figure 3, using the earlier database example.

The resource type identifier is simply the QName of the port type that defines the management interface for the manageable resource.

A management system may use type information to reason about the management interface of different resource types, for example to understand which types support what operations. If required, the resource type information can be gathered into a dictionary for efficiency.

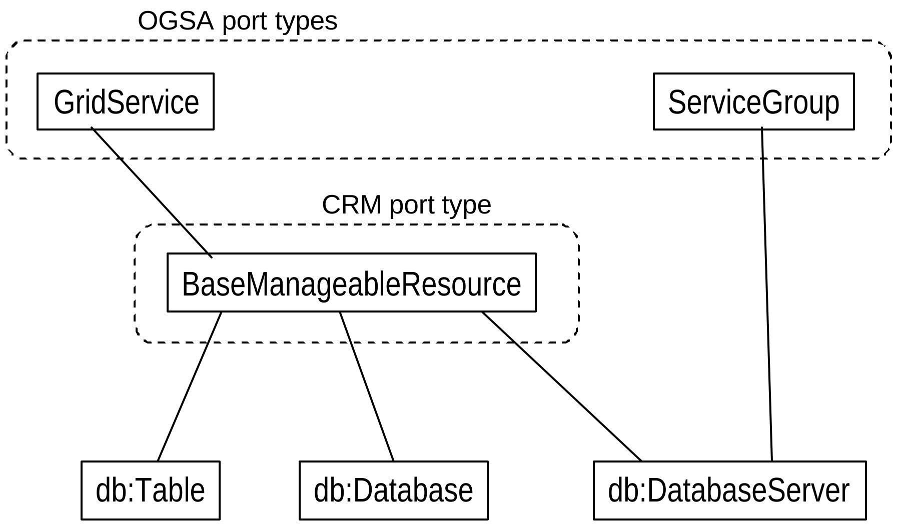

Figure 3: Example manageable resource port types

7.2.1 Resource type properties

Some of the SDEs within the manageable resource port type are declared with mutability attribute of 'static', meaning that their values are set in the WSDL and are shared by all instances of the resource type. These could thus be considered to be properties of the resource type. These properties can be used by management systems to reason about the behavior of a manageable resource type, or to find types that satisfy certain criteria.

What is a basic set of properties that every resource type should set?

- Vendor?
- Name?
- Version?
- Category – e.g. database – should be identified by extending a "database" port type in the resource port type, rather than defining a separate SDE.

7.3 Manageable resource identity

Every manageable resource must be uniquely identifiable within a system. The implementation of a grid service that provides management function for an underlying resource will frequently be based upon existing instrumentation, using CIM or JMX, for example.

It is up to the implementers of each resource type to decide what URI scheme to use and what information the GSH contains. For example, in the case of an implementation based on JMX, the writer may choose to include a JMX MBean name within an HTTP URI. Any mapping that is needed between the GSH and the identity in those other management systems is the responsibility of the implementer.

It is possible that management applications or repositories of information that they use will also need to perform some of these mappings. It may be useful to define a further port type that resource groups could implement to map GSHs that they are aware of to other management systems. This is a topic for further discussion.

7.4 Searching for resources

The criteria used to find manageable resources are likely to be some or all of the following:

- The type of the manageable resource, in the form of the QName of a port type.
  - This may in turn have been found by looking at a number of port type WSDLs to find those that match required characteristics – for example find all port type that extend a standard "database" port type (i.e. find all database resource types)
- Particular values of some of the properties of that resource – these may perhaps be expressed as XPath expressions involving the SDEs of the resource.
- Relationships to other manageable resources.

These criteria are probably not unique to locating manageable resources, and are likely to be generally used for finding other grid services. When combined together these conditions could produce very complex queries, and it may be necessary to define a new query type for FindServiceData that gives additional structure to the query to allow it to be implemented practically. As this query applies at the wider levels than the resource group, it will be necessary to work closely with those working on the wider scale problem, such as Service Domains to ensure that the CRM needs are included.

7.4.1 Search properties

In a large system it would be impractical to have to check against the actual values on each resource, and it would also be impractical to cache the value of all properties for each resource in the registry. For this reason, a service data element, searchProperty, is used to list the important properties that can be used for finding each resource type.

The values of the identified SDEs are held in the Content portion of entry SDEs within the ServiceGroup and are thus available for clients to use to find specific resource instances.

SearchProperty is a static SDE so that management applications can introspect the WSDL to understand what properties are available for finding instances of the resource type.

Ideally there will be a few common basic properties, plus a relatively small set of domain specific properties, e.g. for databases. Individual resource types can also identify additional resource specific properties that will be useful in locating instances of that resource type. The values of the searchProperty service data element are set in the port type definition and cannot change at runtime. As port types are extended to define individual resource types, additional resource type specific properties may be added to the list.

Is there a set of basic search properties that should be implemented by all manageable resources? Name, version, ip address, …? IP address is a strong candidate: many searches are likely to be looking for collocated resources – it would greatly improve these searches if the IP address was held in the resource group.

8. Relationships and Dependencies

Relationships exist between instances of manageable resources.[3] For example, a disk can be contained in a computer system, and a computer system can be part of a cluster. *Relationships* describe which resources are connected and what type of connection exists, but they do not describe the details of how one resource depends on another.

Dependencies add additional information to the relationship to describe exactly how one resource depends on another – for example, a database resource might indicate that it uses a storage resource, and gives details of how much free space is required on that resource.

Relationships and dependencies are described in the following sections.

8.1 Relationships between resource instances

Relationships between manageable resources are named by the verb that describes the relationship, for example "hosts". Relationships have an associated direction, for example the computer system contains the disk. The resources at the two ends of a directional relationship are termed the source and the sink of the relationship. These correspond to the subject and the object of the relationship verb. In the above example, the computer system is the source and the disk is the sink.

The relationship "A hosts B" could also be described as "B is hosted by A", depending on the perspective of the person describing the relationship. The canonical form of the relationship is "A hosts B" – where the transitive form of the verb is used: the relationship type is "hosts", the source is A, and the sink is B. However, this can be hard to understand and it is anticipated that the alternative form (where the inverse of the verb is used) will also be used in descriptive text and user interfaces.

Relationships between resource instances are discovered through the relatedInstance service data element of the BaseManageableResource port type. Note that this service data element only provides information about resource instances and their relationships: it does not provide any modeling information to describe relationships between resource types. This service data element can not have a value assigned in the interface description – the information is only valid for real instances of the resource.

The relatedInstance service data element allows a view of relationships as they are known by the resources at each end of the relationship. In practice, management system may store additional

[3] Relationships are not really specific to CRM, but are really applicable to grid or web services in general. At some point, these constructs should be moved to either the grid services specification or the WSDL specification. Also, we should discuss whether the proposed relationship types are what are needed at the WS-metamodel /OGSI level, or whether they are specific to CRM.

information about the relationship between two resources, and this information is not known to either resource involved. In this case, a separate, independent service is provided that gives access to this information. A management system that wishes to use this additional information must be aware that such a service exists and make use of it as required. This additional service is not covered by this specification.

8.2 Relationships between resource types

Relationships can also be described between manageable resource types, allowing management applications to understand what relationships may exist between instances of those resource types. Relationships between a type and other types are defined using the relatedType service data element in the BaseManageableResource port type.

8.3 Predefined relationship types

<We need to take a look at existing metamodels like UML to determine if whether we can use them to achieve the types of needed relationships>

There is a canonical set of relationship types that may be used:

- Hosts
- Contains
- Federates
- Aggregates
- Uses
- Implements

These are described in the following sections.

Note that this is not an exhaustive list and further relationship types will be added over time. It is important, though, that the canonical relationships are used wherever possible and new types are not added arbitrarily.

Two of the predefined relationships, *hosts* and *contains*, are used to describe the physical containment of resources in the system, and have particular characteristics and implications. The *hosts* relationship is about one resource providing the environment within which another resource lives and runs; the *contains* relationship is about how one resources is built from a set of contained resources.

Both of these relationships imply characteristics on the lifecycle of the hosted or contained resource with respect to its host or container, including when the resource is created or destroyed, and when the resource can be running.

Both of these relationships have a cardinality of one-to-many, i.e. a resource can only be hosted or contained by a single resource. In addition a resource can either be hosted or contained, but not both.

One key difference between these two relationships is the lifecycle of one resource with respect to another – hosts implies a relatively independent lifecycle, where the hosted resource is created after the host and can be deleted before the host: contains implies the same lifecycle as the container, where the resource is created when the container is created and deleted when the container is deleted.

The other relationships describe various other aspects of the system. *Federates* describes the logical structure of an application or solution. *Aggregates* describes how resources are grouped together. *Uses* describes where one resource makes use of the functions of another in order to perform its job. *Implements* describes the way that one resource is actually implemented, and is useful to bridge between a logical, functional view of the system and its physical implementation.

8.3.1 Hosts

A resource A hosts another resource B if resource A provides an environment in which resource B is created and runs. The life cycle of resource B is a subset of the life cycle of resource A: resource B can be created in resource A and can be later removed from resource A. Resource B cannot exist without resource A: if resource A is removed from the system, then resource B must/will also be removed. The hosts relationship also implies that the host resource, A, must be running in order for the hosted resource, B, to be running.

For example, a database hosts the tables within it. Tables can be created at any time within the database, but they cannot exist without the database: if the database is deleted then the tables must also be deleted.

A resource can be the source of any number of *hosts* relationships: that is it can host any number of resource instances.

A resource may be the sink of zero or one *hosts* relationships: that is it cannot be hosted by more than one resource. In addition a resource may not be the sink of both a *hosts* relationship and a *contains* relationship.

The grid services for manageable resources that can be the source of a *hosts* relationship (i.e. that hosts other resources) MUST be able to enumerate the manageable resources that they host, through the relatedResources service data element of the relationships port type.

The informal name of the inverse perspective of this relationship is “isHostedBy”.

8.3.2 Contains

A resource may actually consist of a number of other resources and, therefore, is said to contain them. The contained resource has the same lifetime as the resource that contains it. Therefore, if resource A contains resource B, then when resource A in installed, so is resource B, and when resource A is removed, so is resource B. The *contains* relationship also implies that if resource A is stopped, resource B will also be stopped. It is likely that if resource B is stopped, the operation of resource A will be degraded.

An example is a deployed J2EE application which contains various modules. When the J2EE application is deployed in an application server the modules are also deployed: when the application is removed, the modules are also removed. While the application is running it may be possible to start and stop a module independently, but if the application is stopped then the module will also stop.

A resource may be the source of any number of *contains* relationships: that is it can contain any number of other resource instances.

A resource may be the sink of zero or one *contains* relationships: that is it cannot be contained by more than one resource. In addition a resource may not be the sink of both a *hosts* relationship and a *contains* relationship.

The grid services for manageable resources that can be the source of a *contains* relationship (i.e. that can contain other resources) MUST be able to enumerate the manageable resources that they contain, through the relatedResources service data element of the relationships port type.

The informal name of the inverse perspective of this relationship is “isContainedBy”.

8.3.3 Federates

Where a number of resources in different hosting environments are used together to form another resource, then that resource is said to federate the other resources. The new resource introduces new management operations and possibly new function.

The federated resources do not know about each other, unless some other relationship exists between them for some other reason. For example, if an application includes a database and a message queue, then those two resources do not know about each other – only the application knows that they are working together as part of a larger unit.

Unlike *hosts* and *contains* this relationship does not imply anything about the lifecycle of the federated resource, except that the resource is thought to exist. The database in the above example may exist before the application is installed, and may continue to exist after the application is removed.

A resource may be the source or the sink of any number of *federates* relationships. For example the database that is federated by the above application, may also be a part of another application.

The informal name of the inverse perspective of this relationship is “isFederatedBy”.

8.3.4 Aggregates

Where a number of resources are grouped together for some purpose, then that resource is said to aggregate the other resources. The new resource does not introduce new management operations or function. These groups may be of heterogeneous or homogeneous resource types. They may be defined explicitly through a list of manageable resource identifiers or by a query that is evaluated against the manageable resource registry. An example is a resource that represents all the computers in a department.

The aggregated resources do not know about each other, unless some other relationship exists between them for some other reason.

This relationship does not imply anything about the lifecycle of the aggregated resource, except that the resource is thought to exist.

A resource may be the source or the sink of any number of *aggregates* relationships.

The informal name of the inverse perspective of this relationship is “isAggregatedBy”.

8.3.5 Uses

This relationship is used where one resource uses another resource in order to perform its functions. For example, a user management system may use an LDAP directory to hold user information. This relationship is distinct from *federates* in that *federates* brings together various resources that between them provide a useful function, but is not itself making functional calls to those resources. With the *uses* relationship, the user is actually using the function of the other resource.

A resource may be the source or sink of any number of *uses* relationships. In addition it is also possible for one resource to use another and also to be used, either directly or indirectly, by that other resource. For this reason it is necessary to be careful when following chains of *uses* relationships.

The informal name of the inverse perspective of this relationship is “isUsedBy”.

8.3.6 Implements

This relationship is used where one resource is used to implement the function of another. For example, a database server may be implemented as a Windows service. The implementing resource unit must be running for the implemented resource to be running. The implementing resource may be created dynamically when the implemented resource is run, for example an operating system process, or it may persist between invocations. The implementing resource is usually at a lower level of abstraction than the implemented resource: this relationship would typically be used during problem determination when looking for the source of an application problem.

This is an unusual relationship in that there are probably few cases where one would naturally think of the relationship as “implements”, but rather as the inverse, “is implemented by”. However the relationship is called “implements” in keeping with the practice of using transitive verbs for directional relationships.

A resource may be the source or sink of any number of *implements* relationships.

The informal name of the inverse perspective of this relationship is “isImplementedBy”.

8.4 Dependencies

Relationships describe which resources are connected and what type of connection exists, but they do not describe the details of how one resource depends on another. For example, a database resource might indicate that it uses a storage resource, but this does not indicate how much storage is required.

This dependency information is useful to management applications to allow them to monitor the current validity of the dependency. For example, in the database example, a monitor could be put in place that will detect when the amount of free disk space falls below the minimum value required. This could then be used to alert a person to the problem, or to trigger an automated response.

Specific dependencies on a relationship are expressed as requirements on properties of the related resource. Where multiple requirements apply to the same property of the same resource, those dependencies must be combined in some way. For example if multiple database tables each require a certain amount of free space in a storage device, then the storage device should have the cumulative total of all of the storage requirements. If multiple resources each express the maximum average processor utilization then the lowest one is the correct one to monitor against.

Requirements on a dependency are expressed within the relatedResources service data in the relationships port type. The relatedResource complex type contains an optional ‘requirement’ element, which may occur any number of times. In a future version of the specification, elements will be defined to describe the requirements in a standard way: in the meantime the requirement element just contains an extension point to allow any valid XML to be inserted.

<What is needed for dependencies is for relationships that have associated properties/attributes: perhaps this is not limited to expressing dependencies. The relationship types defined above ('uses',…) may need different property types for different pairs of services they associate: one way of doing this may be by associating another 'entity' to the relationship. This needs further discussion and thought.>

Note that these dependencies relate to the operation of resources once they have been created. They do not attempt to describe or model the requirements in order to be able to create the resource in the first place. For example, the storage requirements for a database table are those required for the table to be used during normal runtime - such as to create new rows - and not a specification of how much free space is required to create the table in the first place - such as to pre-populate it with rows of data.

9. Notifications and events

The Grid Service specification describes a notification mechanism which allows clients to subscribe to receive notifications when the values of specified service data elements changes. Some of the details are still under discussion, for example about support for events (i.e. not related to the change of a SDE). The CRM does not add anything to this notification model. But it probably does need to describe how the notification model is used in the context of manageable resources.

10. XML Attributes

10.1 Change control attributes

There is a well-defined set of XML attributes that can provide change control information about port types, operations, and service data[4].

10.1.1 Version

It is useful to define the concept and constructs for versioning because the definition of port types, operations, and service data will change over time, but it is desirable to leave names the same. It is also desirable to be able to add new port types into namespaces. The XML attribute 'version' is defined as a patterned value of the form majorNum.minorNum.patch

The majorNum and minorNum are monotonically increasing positive numbers. The patch can be numeric, alphabetic, or alpha-numeric. A patch is not required to be monotonically increasing because patches (in terms of implementation) may either be discrete or cumulative. The patch is optional, and if it is not present, it defaults to zero.

<define majorNum, minorNum, and patch; also define compatibility rules within and between the version parts>

<define how *version* is affected by inheritance>

The XML attribute 'version' is defined in the CRM namespace as

[4] The XML attributes version, deprecated, and experimental are not really specific to CRM, but are really applicable to grid or web services in general. At some point, these constructs need to be moved to either the grid services specification or the WSDL specification.

```
   <xsd:simpleType name="versionType">
      <xsd:restriction base="xsd:string">
         <xsd:pattern value="(([0-9])+).(([0-9])+).([(([a-z]|[A-Z]|[0-9])+])"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:attribute name="version" type="xsd:versionType" scope="global"/>
```

Example
In this example, the version of the port type is 1.3.1d.

```
   <wsdl:portType name="OperatingSystem"
                        crm:version=”1.3.1d”>
      …
   </wsdl:portType>
```

10.1.2 Deprecated

The XML attribute ‘deprecated’ indicates that the construct (port type, operation, service data element, service data description, binding) to which it is applied is tolerated but not recommended and may be superceded by another. The qname value of this attribute indicates the name of the replacement definition that should be used. If there is no replacement, then the value must be set to the string ‘null’. The deprecated attribute is scoped according to the following rules:

- When applied to portType, all operations and service data for that portType are also deprecated.
- When applied to a portType that extends another portType, only the operations and service data of extension portType are deprecated.
- When applied to an operation, only the operation is deprecated.
- When applied to a service data element, then that service data element is deprecated.
- When applied to binding, only the binding is deprecated.

<define how *deprecated* is affected by inheritance>

The XML attribute ‘deprecated’ is defined in the CRM namespace as

```
   <xsd:attribute name="deprecated" scope="global">
      <xsd:simpleType>
         <xsd:union>
            <xsd:simpleType>
               <xsd:restriction base="xsd:QName">
               </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
               <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="null"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:union>
      </xsd:simpleType>
   </xsd:attribute>
```

Example
In this example, the portType ‘OperatingSystemPortType’ is deprecated and its replacement is ‘newOSPortType’ in the IBM XML namespace.

```
   <wsdl:portType name="OperatingSystemPortType"
                        deprecated=”ibm:newOSPortType”>
      <wsdl:documentation>Operating System port type</wsdl:documentation>
```

```
     …
     (operations and service data for this port type)
     …
  </wsdl:portType>
```

10.1.3 Experimental

The XML attribute ‘experimental’ indicates that the construct (port type, operation, service data element., service data description, binding) to which it is applied is available for experimentation and at some point will either become part of a formal release or removed entirely. (Removal of anything marked experimental would not be re-labeled deprecated). The default value is ‘false’. The ‘experimental’ attribute is scoped according to the following rules:

- When applied to portType, all operations and service data for that portType are also experimental.
- When applied to a portType that extends another portType, only the operations and service data of extended portType are experimental.
- When applied to an operation, only the operation is experimental.
- When applied to a service data element, only that service data element is experimental.
- When applied to binding, only the binding is experimental.

<define how *experimental* is affected by inheritance>

The XML attribute ‘experimental’ is defined in the CRM namespace as

```
  <xsd:attribute name="experimental" type="xsd:boolean" scope="global"/>
```

Example
In this example, the portType ‘newOperatingSystemPortType’ is experimental.

```
  <wsdl:portType name="newOperatingSystemPortType"
                 crm:experimental=”true”>
    <wsdl:documentation>Operating System port type</wsdl:documentation>
     …
     (operations and service data of the port type)
     …
  </wsdl:portType>
```

10.2 Units attribute

The XML attribute ‘units’ defines the unit of measure for the value of a XSD schema element used in the service data element. Its type is the union of an enumerated list of well-known units and a string to allow the value of ‘units’ to be extensible.

The XML attribute ‘units’ is defined as

```
  <xsd:simpleType name=”crm:unitsType”>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="unit1"/>
          <xsd:enumeration value="unit2"/>
          …
          <xsd:enumeration value="unitn"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base="xsd:string" />
      </xsd:simpleType>
```

```
        </xsd:union>
    </xsd:simpleType>
```

where ***unit1***, ***unit2***, to ***unitn*** are well-known units of measure. An appendix will be added later with the complete enumeration – based on units used in CIM and common metrics based on product experience.

```
    <xsd:attribute name="units" type="crm:unitsType" scope="global"/>
```

Example
In this example, the value of the service data element totalVirtualMemorySize is in units of kilobytes.

```
    <sd:serviceData name="totalVirtualMemorySize" type=" xsd:nonNegativeInteger"
            minOccurs="0" maxOccurs="1" mutability="mutable"
            crm:units=kilobytes
        <wsdl:documentation>
            Number of Kbytes of virtual memory. For example,
            this may be calculated by adding the amount of total RAM to
            the amount of paging space (i.e., adding the amount of
            memory in/aggregated by the ComputerSystem to the property,
            SizeStoredInPagingFiles.
        </wsdl:documentation>
    </sd:serviceData>
```

10.3 Lifecycle attributes

There are XML attributes that further describe the lifecycle characteristics of the resource for use by an application or management tool, specifically the changeability, validity, volatility, and latency of the pieces comprising the service.

Taking a message queue example, not all of the service data elements of the queue have the same behavior over time. For example, the queue will have a service data element queueName which is set set when the queue is created. Another service data element, averageTimeInQueue is only valid while the queue is running, and the queue may require a restart for changes to the service data element maximumNumberOfMessages to take effect.

The value of the lifecycle XML attributes changeable and valid can be one or more lifecycleValueTypes representing valid combinations of lifecycleState and subState defined in the resources lifecycleModel SDE, or the values ‘any’ or ‘unknown’. The ‘any’ value indicates that resource can be in any state for the service data / operation to be valid or changeable. The ‘unknown’ value indicates that the state a resource’s service data element or operation must be in to be valid or changeable is not known or not defined. lifecycleValueType is a simple XML type that represents a lifecycleState and optional substate defined as:

```
    <xsd:simpleType name="lifecycleValueType">
        <xsd:annotation>
            <xsd:documentation>A lifecycleValue contains two parts - the
             lifecycle state and the lifecycle sub-state. Each of these
             is an NCName, and the sub-state part is optional. The
             format is defined as follows:

               lifecycleValue ::= lifecycleState (':' lifecycleSubState)?
               lifecycleState ::= NCName
            lifecycleSubState ::= NCName</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[\i-[:]][\c-[:]]*(:[\i-[:]][\c-[:]]*)?"></xsd:pattern>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="lifecycleValueListType">
    <xsd:list itemType="crm:lifecycleValueType"/>
  </xsd:simpleType>
```

The following table summarizes what/when the XML attribute can be specified (indicated by Y) and not specified (indicated by N).

- The “Type” column indicates whether the attribute is used in the WSDL/GSDL that defines the port type – i.e. the value of the attribute does not change from one instance of the resource to the next.
- The “Instance” column indicates whether this attribute can be used within the service data of the service, so that the value of the attribute can vary from one service data value to another or from one service instance to another.
- The “Service Data” and “Operation” columns indicate whether the attribute applies to service data (either its definition or content) and operations respectively.

	Type	**Instance**	**Service Data**	**Operation**
modifiable	Y	N	Y	N
valid	Y	N	Y	Y
changeable	Y	N	Y	N
volatile	Y	Y	Y	N
latency	Y	N	Y	Y

10.3.1 valid

The XML attribute ‘valid’ specifies when in the lifecycle of a resource an element defined in the port type is valid. The semantic meaning depends on the type of element:

- For service data, validity indicates the lifecycle(s) in which the service data can be read and is meaningful.
- For operations, validity indicates the lifecycle in which it is meaningful to invoke the operation.

This attribute is defined as follows:

```
  <attribute name="valid" type="crm:lifecycleValueListType" scope="global"/>
```

Example
In this example (from the OperatingSystem port type in Appendix C), the lastBootUpTime service data is meaningful and valid to be read when the OperatingSystem port type is in the down, starting, up, stopping, or failed state.

```
    <sd:serviceData name="lastBootUpTime" type="xsd:datetime"
      minOccurs="0" maxOccurs="1"  mutability="mutable"
      crm:valid="down starting up stopping failed">
      <wsdl:documentation>
        Time when the OperatingSystem was last booted.
      </wsdl:documentation>
    </sd:serviceData>
```

10.3.2 changeable

The XML attribute ‘changeable’ is used with service data to indicate when in the lifecycle of a resource the value(s) of its service data can be changed by an application or management tool.

It differs from the XML attribute ‘valid’ in that validity indicates when the service data can be read. The distinction is important because service data may be modifiable only at certain times. For example, the persistence state of a message queue (indicating whether messages are persisted or not) may only be changed while the queue is down, but can be viewed at any time. In this example, the changeable value would be “down” whereas the valid value would be “any”.

This attribute is defined as follows:

```
<attribute name="changeable" type="crm:lifecycleValueListType" scope="global"/>
```

Example
In this example (from the OperatingSystem port type in Appendix C), the localDateTime service data can be changed by an application or management tool when the OperatingSystem port type is in the starting, up, stopping, or failed state.

```
<sd:serviceData name="localDateTime" type="xsd:datetime" minOccurs="0"
   maxOccurs="1" mutability="mutable" modifiable="true"
   crm:valid="down starting up stopping failed"
   crm:changeable="starting up stopping failed"
   crm:latency="immediate”>
   <wsdl:documentation>
      OperatingSystem's notion of the local date and time of day.
   </wsdl:documentation>
</sd:serviceData>
```

10.3.3 latency

The XML attribute ‘latency’ is valid for modifiable service data and for operations. Latency indicates when the write action or the result of the operation takes effect. For example, latency=”whenStarted” indicates that the resource needs to be restarted before changes to configuration or an operation action takes effect.

The following simpleType is defined for use with the XML attribute latency.

```
<simpleType name=”crm:latencyValueType”>
   <restriction base=”string”>
      <enumeration value=”whenStarted“/>
      <enumeration value=“immediate“/>
      <enumeration value=”afterResync“/>
      <enumeration value=”delayed“/>
   </restriction>
</simpleType>
```

The enumerated values mean:

- whenStarted – when the resource is restarted
- immediate – now; during or upon return from executing the action
- afterResync – resource requires a specific command to execute before new value takes effect, e.g. re-read of the resource’s configuration file
- delayed – after a reasonable amount of time; volatility can be used to indicate the period until the change takes effect

How do the latencyValueTypes relate to the lifecycleModel? Perhaps the possible latency values should be specified as part of the model.

This attribute is defined as follows:

```
   <attribute name="latency" type="crm:latencyValueType" scope="global"/>
```

Example
In this example (from the OperatingSystem port type in Appendix C),when an application or management tool changes the localDateTime service data, the change takes effect immediately.

```
     <sd:serviceData name="localDateTime" type="xsd:datetime" minOccurs="0"
        maxOccurs="1" mutability="mutable" modifiable="true"
        crm:valid="down starting up stopping failed"
        crm:changeable="starting up stopping failed"
        crm:latency="immediate”>
        <wsdl:documentation>
           OperatingSystem's notion of the local date and time of day.
        </wsdl:documentation>
     </sd:serviceData>
```

10.3.4 volatile

The XML attribute ‘volatile’ indicates how frequently a service data element may be changed by the resource as part of its normal operation. It provides a rough guideline or hint to the application as to the frequency that changes may occur. For example, a monitoring application that reads service data for graphing purposes would use the volatile value to determine roughly how often to read that service data. Note that this attribute does not give any indication that changes will actually occur at this frequency – in some periods multiple changes may occur, or it may be much longer then this period between changes.

The unit of measure of the volatility attribute is seconds. (Note: a service data element may change more frequently than every second, but from a monitoring viewpoint ‘seconds’ is a reasonable unit of measure). The ‘units’ attribute cannot be used to indicate other units for the volatile attribute, because its use here would apply to the element, for example a counter or gauge, rather than the volatile attribute.

Is it necessary to allow units to be specified for the volatile attribute? If so, then what is the best approach to add this? Perhaps make the attribute type a string (or some restriction of string) and allow the units to be added within the value of the volatile attribute. Or add a second attribute that indicates the units e.g. volatileUnits.

The volatility attribute is different from the grid service lifecycle attributes goodFrom, goodUntil, and availableUntil; the grid service lifecycle attributes state the validity of cached values of service data whereas the volatility attribute indicates the frequency of change at the resource.

This attribute is defined as follows:

```
   <attribute name="volatile" type="xsd:nonNegativeInteger" scope="global"/>
```

Example
In this example (from the OperatingSystem port type in Appendix C), the number of processes changes very frequently within the operating systems; volatility is set to 1 second – the minimum volatility change time.

```
     <sd:serviceData name="numberOfProcesses" type="crm:gauge"
        minOccurs="0" maxOccurs="1" mutability="mutable"
        crm:valid="starting up stopping failed"
        crm:volatile=”1”>
        <wsdl:documentation>
            Number of process contexts currently loaded or running on
            the OperatingSystem.
            Although CIM defines a property for maxNumberOfProcesses,
            it is not necessary here;  maxNumberOfProcesses is expressed
            as the maximum value for this numberOfProcessses gauge
        </wsdl:documentation>
    </sd:serviceData>
```

11. Use of XML data types for modeling

The data types defined in XML [xmldatatypes] are the basis for the data types used to model a resource as a service. There are many data types from existing resource models that are low level and reflect implementation. To make these useful in XML, these are modeled as an abstraction that can be mapped or implemented in low level constructs. Examples are arrays and bits. This chapter describes how some common data types used in existing management models may be mapped into existing XML data types. This chapter also describes two additional data types, counter and gauge, that are useful for modeling management interfaces. These new data types are derived from the basic XML set of data types.

11.1 Mapping common data types to XML data types

Resource models define data types. In the OGSA Common Resource Model, XML schema is used, and there is a set of predefined data types. In existing resource models, there are data types that one might find are not obvious how to express in XML. And in most cases, it is not fruitful to just add those same data types from existing resource models in the OGSA Common Resource Model since those data types can be expressed using the current XML schema. This section show how some of the common data types from other resource models can (and should) be expressed in XML.

11.1.1 Array

An array is used to hold multiple values in a structured manner. In systems management, arrays have typically been 1-dimensional. Multi-dimensional arrays may be useful, depending on what is being expressed.

A 1-dimensional array can be expressed on the element construct by specifying the constraints minOccurs and maxOccurs. If maxOccurs=unbounded, then the array is unbounded. All of an array's values are homogeneous, that is, they are of the same data type.

Since an array is really a set of elements, another way to define a 1-dimensional array is to define a sequence of elements, each of the same data type. Multi-dimensional arrays are then sequences or sequences. Each value of each element is single or multi-valued, again controlled by use of minOccurs and maxOccurs.

The author defining the array decides what significance is attached to the document order of the array.

An array can be indexed. You can add an attribute to the elements which allows you to find whatever entry you want, using the appropriate index.

XML modeling experience will surface best practices for modeling arrays.

11.1.2 Bit or binary

Bit or binary is a low level implementation. Bit or binary could be defined using hexBinary, but in practice they are used to combine a number of independent fields within a single string of bits. These fields may a single bit, giving a binary choice, or may have multiple bits, giving either a small selection of enumerated values, or a small number. When modeling this in XML it is appropriate to describe each of the fields explicitly within a sequence using appropriate XML data types, for example 'boolean', 'integer' or enumerations. In addition, it is usually appropriate to define enumerations in the form of text strings, rather than numeric values.

For example, suppose a resource natively supplies a set of data, resourceInfo, in the form of an 8 bit field, specifically an unsignedByte, (labeled 0-7 right to left).

- Bits 0 and 1 display 4 states of the resource: 00=deployed, 01=installed, 10=executable, 11=running
- Bit 2 is unused
- Bit 3 tells whether a particular facet of the resource is available, a boolean
- Bit 4 is unused
- Bits 5-7 give the 3 speeds of a resource, one bit for each speed where only one of the bits may be set at any given time: bit 5 when set is speed=1x, bit 6 when set is speed 2x, bit 7 when set is speed 4x

In XML, this bit or binary implementation of an unsignedByte could be defined as:

```
<complexType name=”resourceInfo”>
   <sequence>
      <element name=”resourceState”>
         <restriction base=”string”>
            <enumeration value=”deployed”/>
            <enumeration value=”installed”/>
            <enumeration value=”executable”/>
            <enumeration value=”running”/>
         </restriction>
      </element>
      <element name=”resourceFacetEnabled” type=”boolean”/>
      <element name=”resourceSpeed”>
         <restriction base=”string”>
            <enumeration value=”1x”/>
            <enumeration value=”2x/>
            <enumeration value=”4x/>
         </restriction>
      </element>
   </sequence>
</complexType>
```

11.1.3 Octet

The hexBinary data type is used to express data specified in the octet data type from other resource models.

11.2 New XML data types

The data types used in the Common Resource Model are those defined in [xmldatatypes] plus the derived data types defined in this section. These derived data types are associated with the CRM namespace.

11.2.1 Counter

A ‘counter’ represents a non-negative integer that monotonically increases until it reaches a maximum value, when it wraps around and starts increasing again from zero. A counter may also be reset to zero as needed. Counter is a complex concrete data type. Its minimum value is implicitly zero; its maximum value is specified as an XML attribute of the gauge, allowing it to take different values for individual instances of a resource, and even to allow the maximum value to change during the lifetime of the resource instance.
If a counter element for a particular resource has an absolute maximum value that constrains the possible range of the counter, then this can be specified using the maxValue attribute on the definition of the counter in the schema for that resource. The minimum width of the data type (byte, short, int, long) is an implementation detail and may be derived from the maximum value of the counter.

The counter data type is only applied to input and output messages (of operations) and service data elements. The counter data type has the implied semantic (stated in previous paragraph) that is implemented by the resource and useful knowledge for (management) applications when displaying the information represented by those data types. Applications that need to discover whether a service data element represents a counter can read the XML for that service data element and parse for a type set to ‘counter’, or base type set to ‘counter’ for new data types derived from data type counter.

A counter can be reset to zero by an application or management tool if it is used in a context where writing the value is possible. In the Common Resource Model writing is possible if, for example, the service data element that contains the counter is writeable, i.e. has the ‘modifiable’ attribute set.

11.2.1.1 Lexical representation

A counter has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678967543233, 100000, 9999, 126.

11.2.1.2 Canonical representation

The canonical representation for a counter is defined by prohibiting certain options from the Lexical representation. Specifically, leading zeroes are prohibited.

11.2.1.3 XML definition

```
<complexType name="counter">
   <extension base=”nonNegativeInteger”>
      <attribute name=maxValue type=”nonNegativeInteger” use=”required”/>
      <attribute ref=”units”/>
   </extension>
</complexType>
```

11.2.1.4 Example

A disk resource may keep statistics on the number of disk read errors. This is a monotonically increasing number that can wrap to zero when the maximum specified value is reached. This data can be modeled as a counter. The width of the data type (for implementation) is chosen based on the maximum value. In this example, the element
diskReadErrors is defined in the schema to be a counter, and in the diskReadErrors element instance it has a current value of 453 and a maximum value of 999.

```
…
<element name=”diskReadErrors” type=”counter”/>
```

```
…
```

```
…
<diskReadErrors maxValue=”999”>453</diskReadErrors>
…
```

11.2.2 Gauge

A ‘gauge’ represents an integer that may increase or decrease, but can never exceed a minimum or maximum value. Gauge is a complex concrete data type. The maximum and minimum values for the gauge are specified as attributes of the gauge, allowing them to take different values for individual instances of a resource, and even to allow the maximum and minimum values to change during the lifetime of the resource instance. An example of the latter case is a gauge that represents freeVirtualMemory, where the maximum value would change if additional paging space was added to the resource.
If a gauge element for a particular resource has absolute maximum and minimum values that constrain the possible range of the gauge, then these can be specified using the minValue and maxValue attributes on the definition of the gauge in the schema for the resource. The minimum width of the data type (byte, short, int, long) is an implementation detail and may be derived from the maximum value of the gauge.

The gauge data type is only applied to input and output messages (of operations) and service data elements. The gauge data type has the implied semantic (stated in previous paragraph) that is implemented by the resource and useful knowledge for (management) applications when displaying the information represented by those data types. Applications that need to discover whether a service data element represents a gauge can read the XML for that service data element and parse for a type set to ‘gauge’, or a base type set to ‘gauge’ for new data types derived from data type gauge.

The gauge defined here allows integer values only. Is another gauge type needed that allows float or double values?

11.2.2.1 Lexical representation

A gauge has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678967543233, -100000, 9999, 126, -12.

11.2.2.2 Canonical representation

The canonical representation for a gauge is defined by prohibiting certain options from the Lexical representation. Specifically, leading zeroes are prohibited.

11.2.2.3 XML definition

```
<complexType name="gauge">
  <extension base=”integer”>
    <attribute name=minValue type=”integer” use=”required”/>
    <attribute name=maxValue type=”integer” use=”required”/>
    <attribute ref=”units”/>
  </extension>
<complexType>
```

11.2.2.4 Example

An operating system tracks the amount of free physical memory. For example, a machine may have 512 megabytes of physical memory. The amount of free physical memory, at a point in time, can range from zero to 512 megabytes. The amount free decreases when memory is

allocated and increases when it is freed. When no more memory can be allocated, the value stays pinned at zero. This behavior is typical of a gauge, and hence is modeled as a gauge. The width of the data type (for implementation) is chosen based on the maximum value. The units attribute of the values (in this example, it is megabytes) is discussed in Section 10.2.
In this example, the element freePhysicalMemory is defined in the schema as being a gauge. In the example freePhysicalMemory element instance, it has a current value of 213, with maximum value of 512 and minimum of 0; the units of the values are in kilobytes.

```
…
<element name=”freePhysicalMemory”  type=”gauge” />
…
```

```
…
<freePhysicalMemory minValue=”0” maxValue=”512”
     units=”kilobytes”>213</freePhysicalMemory>
…
```

Another example of a gauge is a volt meter which may have a reading range of min=-12 to max=+12 where the units are ‘volts’.

12. How to model manageable resources

This will consist of a set of guidelines or hints and tips for how to go about modeling manageable resources. This may be pulled out into a separate document and will no doubt grow as modeling progresses.

13. Security Considerations

This specification defines the abstract interaction between the grid service that represents a manageable resource and clients of that service. 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 defined how the abstract interactions are bound to specific communication protocols, and to specific programming environments.

Appendix A. A Common Lifecycle Model

A resource or service goes through a set of states throughout its life. For example, a queue is in the down state when it is created, in the up state when it can receive and process requests, and in the failed state if it stops unexpectedly and is no longer responding.

Note that different operations will be valid in different states. For example, ‘stop’ is not a valid operation if you are in a ‘down’ state.

There are five lifecycle states for a resource:

- Down – In this state the resource has been created, but cannot do useful work until it is Up. However, information about the resource is available in this state. (Note that the some of the operations or service data elements of the service are available in this state, although the resource the service represents is not).

- Starting – This is a transient state that indicates that the resource is in the process of starting up. The next state is either the Up or Failed state.
- Up – In this state the resource is running and available to process new work.
- Stopping – This is a transient state that indicates that the resource is in the process of stopping. During this state it is unlikely that new work will be accepted. The next state is either the Down or Failed state.
- Failed – In this state the resource is not available except for managing first-failure artifacts that are useful for problem determination. This state is typically entered after the resource crashes. The resource must be transitioned to a Down state before it can be started again.

These states, and the possible transitions between them, are illustrated in Figure 4.

If a manageable resource cannot define its lifecycle state, then the resource is said to be in the 'unknown' state. When a resource is in the 'unknown' state, the information provided by the lifecycle XML attributes is not useful to an application or management tool.

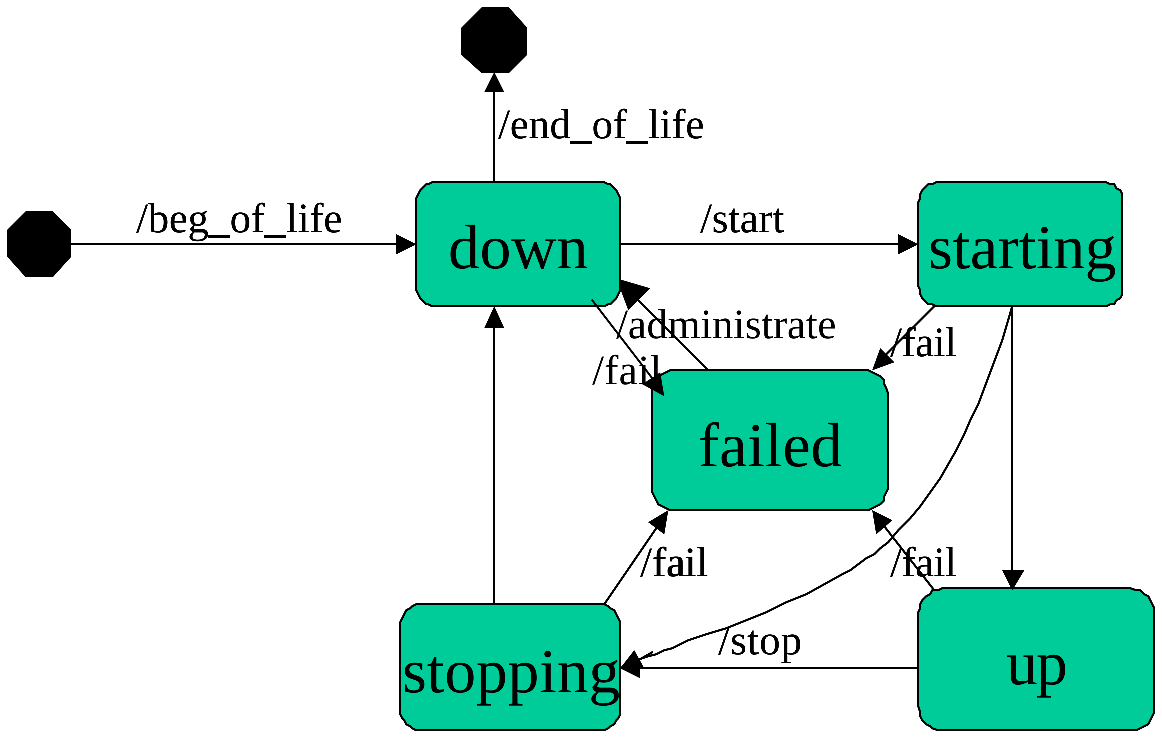

Figure 4: Lifecycle states

Each lifecycle state has additional information about its operational state.

A resource in the down state can specify these operational characteristics:
- restartable: resource is stopped but is restartable
- recovered: resource is down but is restartable. This can occur if the resource has abnormally ended and recovered an error or is in maintenance.

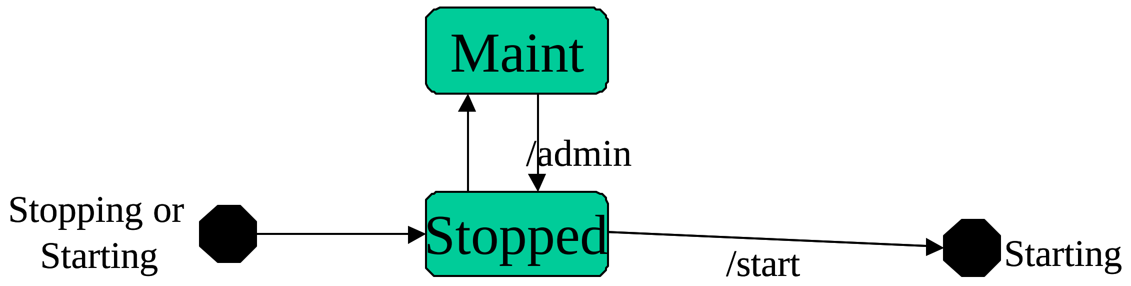

Figure 3. Down sub-states

A resource in the starting state can specify these operational characteristics:

- OK: resource expected to attain stated state soon
- error: resource expected to attain failed state soon

A resource in the up state can specify these operational characteristics:

- idle: resource provides the expected service, can accept new work, but is not currently processing work
- busy: resource provides the expected service, can accept new work, and is currently processing work
- degraded: resource runs but not optimally. It may not deliver 100% of expected service, e.g. performance bottleneck, or it may be consuming excess system resources, it may be starved for resources, it may be saturated with work, it may recognize that failure is imminent, or it may be running maintenance with limited availability

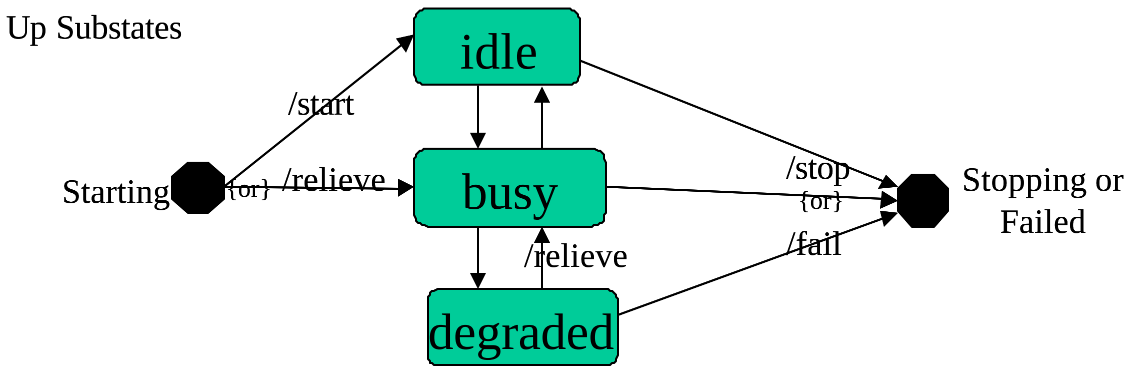

Figure 3. Up sub-states

A resource in the stopping state can specify these operational characteristics:

- OK: resource expected to attain stopped state soon
- error: resource expected to attain failed state soon

A resource in the failed state can specify these operational characteristics:

- dependencyFailure: resource not restartable because of a loss of a supporting/hosting resource
- nonRecoverableError: resource not restartable because of a non-recoverable error

So, the lifecycle model described above would be expressed in a resource's port type as:

```
<sd:staticServiceDataValues>
  <lifecycleModel>
```

```
    <crm:lifecycleState name="down">
      <subState name="restartable"/>
      <subState name="recovered"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="starting">
      <subState name="OK"/>
      <subState name="error"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="up">
      <subState name="idle"/>
      <subState name="busy"/>
      <subState name="degraded"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="stopping">
      <subState name="OK"/>
      <subState name="error"/>
    </crm:lifecycleState>
    <crm:lifecycleState name="failed">
      <subState name="dependencyFailure"/>
      <subState name="nonrecoverableError"/>
    </crm:lifecycleState>
  </lifecycleModel>
</sd:staticServiceDataValues>
```

Appendix B. BaseManageableResource Port Type

This is the WSDL for the BaseManageableResource port type. The CRM defined XML data types are assumed to be in a separate XML schema file.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="xxx"
   targetNamespace="http://www.gridforum.org/namespaces/2003/17/crm"
   xmlns:crm="http://www.gridforum.org/namespaces/2003/17/crm"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <gwsdl:portType name="BaseManageableResource" extends="gwsdl:OGSIGridService”>

      <sd:serviceData
            name="lifecycleModel"
            type="crm:lifecycleModelType"
            minOccurs=”1”
            maxOccurs=”1”
            nillable=”true”
            mutability=”static”/>

      <sd:serviceData
            name="currentLifecycleState"
            type=”lifecycleStateType"
            minOccurs=”1”
            maxOccurs=”1”
            mutability=”mutable”/>

      <sd:serviceData
```

```
            name="serviceGroupType"
            type="xsd:QName"
            minOccurs="0"
            maxOccurs="1"
            mutability="static" />

      <sd:serviceData
            name="searchProperty"
            type="xsd:QName"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability="static"/>

      <sd:serviceData
            name="relatedInstance"
            type="crm:relatedInstance"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability="mutable"/>

      <sd:serviceData
            name="relatedType"
            type="crm:relatedType"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability="static"/>

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

Appendix C. Operating System Port Type

This is the start of the port type for an operating system based on the current Grid Service specification and this version of the Common Resource Model specification. It is based on the CIM OperatingSystem class. It is still under construction.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="xxx" targetNamespace="someURI"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types>
      <xsd:schema targetNamespace="someURI">
         <xsd:simpleType name="OSTypeType">
            <xsd:union>
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:enumeration value="Unknown"/>
                     <xsd:enumeration value="MACOS"/>
                     <xsd:enumeration value="ATTUNIX"/>
                     <xsd:enumeration value="DGUX"/>
                     <xsd:enumeration value="DECNT"/>
                     <xsd:enumeration value="Tur64 UNIX"/>
                     <xsd:enumeration value="OpenVMS"/>
                     <xsd:enumeration value="HPUX"/>
                     <xsd:enumeration value="AIX"/>
                     <xsd:enumeration value="MVS"/>
```

```
<xsd:enumeration value="OS400"/>
<xsd:enumeration value="OS/2"/>
<xsd:enumeration value="JavaVM"/>
<xsd:enumeration value="MSDOS"/>
<xsd:enumeration value="WIN3x"/>
<xsd:enumeration value="WIN95"/>
<xsd:enumeration value="WIN98"/>
<xsd:enumeration value="WINNT"/>
<xsd:enumeration value="WINCE"/>
<xsd:enumeration value="NCR3000"/>
<xsd:enumeration value="NetWare"/>
<xsd:enumeration value="OSF"/>
<xsd:enumeration value="DC/OS"/>
<xsd:enumeration value="Reliant UNIX"/>
<xsd:enumeration value="SCO UnixWare"/>
<xsd:enumeration value="SCO OpenServer"/>
<xsd:enumeration value="Sequent"/>
<xsd:enumeration value="IRIX"/>
<xsd:enumeration value="Solaris"/>
<xsd:enumeration value="SunOS"/>
<xsd:enumeration value="U6000"/>
<xsd:enumeration value="ASERIES"/>
<xsd:enumeration value="TandenNSK"/>
<xsd:enumeration value="TandemNT"/>
<xsd:enumeration value="BS2000"/>
<xsd:enumeration value="LINUX"/>
<xsd:enumeration value="Lynx"/>
<xsd:enumeration value="XENIX"/>
<xsd:enumeration value="VM/ESA"/>
<xsd:enumeration value="Interactive UNIX"/>
<xsd:enumeration value="BSDUNIX"/>
<xsd:enumeration value="FreeBSD"/>
<xsd:enumeration value="NetBSD"/>
<xsd:enumeration value="GNU Hurd"/>
<xsd:enumeration value="OS9"/>
<xsd:enumeration value="MACH Kernel"/>
<xsd:enumeration value="Inferno"/>
<xsd:enumeration value="QNX"/>
<xsd:enumeration value="EPOC"/>
<xsd:enumeration value="IxWorks"/>
<xsd:enumeration value="VxWorks"/>
<xsd:enumeration value="MiNT"/>
<xsd:enumeration value="BeOS"/>
<xsd:enumeration value="HP MPE"/>
<xsd:enumeration value="NextStep"/>
<xsd:enumeration value="PalmPilot"/>
<xsd:enumeration value="Rhapsody"/>
<xsd:enumeration value="Windows 2000"/>
<xsd:enumeration value="Dedicated"/>
<xsd:enumeration value="OS/390"/>
<xsd:enumeration value="VSE"/>
<xsd:enumeration value="TPF"/>
<xsd:enumeration value="Windows (R) Me"/>
<xsd:enumeration value="Caldera Open UNIX"/>
<xsd:enumeration value="OpenBSD"/>
<xsd:enumeration value="Not Applicable"/>
```

```
                    <xsd:enumeration value="Windows XP"/>
                 </xsd:restriction>
              </xsd:simpleType>
              <xsd:simpleType>
                 <xsd:restriction base="xsd:string"/>
              </xsd:simpleType>
           </xsd:union>
        </xsd:simpleType>

        <xsd:simpleType name="versionType">
           <xsd:restriction base="xsd:string">
              <xsd:pattern
                 value="(([0-9])+).(([0-9])+).([([a-z]|[A-Z]|[0-9])+])"/>
           </xsd:restriction>
        </xsd:simpleType>

     </xsd:schema>
  </wsdl:types>

  <wsdl:message name="ResultResponse">
     <wsdl:part name="Return" type="xsd:unsignedInt"/>
  </wsdl:message>

  <wsdl:message name="shutdownRequest"></wsdl:message>

  <wsdl:message name="rebootRequest"></wsdl:message>

  <gwsdl:portType name="OperatingSystem" extends="crm:BaseManageableResource">
     <wsdl:documentation>
        An OperatingSystem is software/firmware that makes a
        ComputerSystem's hardware usable, and implements and/or manages the
        resources, file systems, processes, user interfaces, services, ...
        available on the ComputerSystem.
     </wsdl:documentation>

     <wsdl:operation name="reboot">
        <wsdl:input message="rebootRequest"/>
        <wsdl:output message="ResultResponse"/>
        <wsdl:documentation>
           Requests a reboot of the OperatingSystem. The return
           value should be 0 if the request was successfully executed, 1
           if the request is not supported and some other value if an
           error occurred. In a subclass, the set of possible return codes
           could be specified.
        </wsdl:documentation>
     </wsdl:operation>

     <wsdl:operation name="shutdown">
        <wsdl:input message="shutdownRequest"/>
        <wsdl:output message="ResultResponse"/>
        <wsdl:documentation>
           Requests a shutdown of the OperatingSystem. The return
           value should be 0 if the request was successfully executed, 1
           if the request is not supported and some other value if an
           error occurred. It is up to the implementation or subclass of
           OperatingSystem to establish dependencies between the Shutdown
        and Reboot methods, and for example, to provide more
        sophisticated capabilities such as scheduled shutdown, reboot,
        etc. In a subclass, the set of possible return codes could be
        specified.
      </wsdl:documentation>
    </wsdl:operation>

    <sd:staticServiceDataValues>
      <lifecycleModel>
        <crm:lifecycleState name="down">
          <subState name="restartable"/>
          <subState name="recovered"/>
        </crm:lifecycleState>
        <crm:lifecycleState name="starting">
          <subState name="OK"/>
          <subState name="error"/>
        </crm:lifecycleState>
        <crm:lifecycleState name="up">
          <subState name="idle"/>
          <subState name="busy"/>
          <subState name="degraded"/>
        </crm:lifecycleState>
        <crm:lifecycleState name="stopping">
          <subState name="OK"/>
          <subState name="error"/>
        </crm:lifecycleState>
        <crm:lifecycleState name="failed">
          <subState name="dependencyFailure"/>
          <subState name="nonrecoverableError"/>
        </crm:lifecycleState>
      </lifecycleModel>
    </sd:staticServiceDataValues>

    <sd:staticServiceDataValues>
      <wsdl:documentation>
        operating systems are contained by computer systems
      </wsdl:documentation>
      <crm:relatedType>
        <relationshipType>contains</relationshipType>
        <source>crm:ComputerSystem</source>
      </crm:relatedType>
    </sd:staticServiceDataValues>

    <sd:serviceData name="OSType" type="OSTypeType" minOccurs="0"
      maxOccurs="1" mutability="setOnce"
      crm:valid="down starting up stopping failed">
      <wsdl:documentation>
        A string describing the manufacturer and OperatingSystem type.
      </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="version" type="versionType" minOccurs="0"
      maxOccurs="1" mutability="setOnce"
      crm:valid="down starting up stopping failed">
      <wsdl:documentation>
        A string describing the Operating System's version number. The
```

```
            format of the version information is as follows:
            majorNumber.minorNumber.revision or
            majorNumber.minorNumber.revisionLetter
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="lastBootUpTime" type="xsd:datetime"
        minOccurs="0" maxOccurs="1"  mutability="mutable"
        crm:valid="down starting up stopping failed">
        <wsdl:documentation>
            Time when the OperatingSystem was last booted.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="localDateTime" type="xsd:datetime" minOccurs="0"
        maxOccurs="1" mutability="mutable" modifiable="true"
        crm:valid="down starting up stopping failed"
        crm:changeable="starting up stopping failed"
        crm:latency="immediate”>
        <wsdl:documentation>
            OperatingSystem's notion of the local date and time of day.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="curentTimeZone" type="xsd:short" minOccurs="0"
        maxOccurs="1" mutability="mutable" modifiable="true" crm:units="minutes"
        crm:valid="down starting up stopping failed"
        crm:changeable="starting up stopping failed"
        crm:latency="immediate">
        <wsdl:documentation>
            CurrentTimeZone indicates the number of minutes
            the OperatingSystem is offset from Greenwich Mean Time. Either
            the number is positive, negative or zero.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="numberOfLicensedUsers" type="unsignedInt"
        minOccurs="0" maxOccurs="1" mutability="mutable" modifiable="true"
        crm:valid="down starting up stopping failed"
        crm:changeable="up failed"
        crm:latency="immediate">
        <wsdl:documentation>
            Number of user licenses for the OperatingSystem. If
            unlimited, enter 0.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="numberOfUsers" type="crm:gauge" minOccurs="0"
        maxOccurs="1" mutability="mutable"
        crm:valid="down starting up stopping failed">
        <wsdl:documentation>
            Number of user sessions for which the OperatingSystem is
            currently storing state information.
        </wsdl:documentation>
    </sd:serviceData>
```

```
<sd:serviceData name="numberOfProcesses" type="crm:gauge"
    minOccurs="0" maxOccurs="1" mutability="mutable"
    crm:valid="starting up stopping failed"
    crm:volatile=”1”>
    <wsdl:documentation>
        Number of process contexts currently loaded or running on
        the OperatingSystem.
        Although CIM defines a property for maxNumberOfProcesses,
        it is not necessary here;  maxNumberOfProcesses is expressed
        as the maximum value for this numberOfProcessses gauge
    </wsdl:documentation>
</sd:serviceData>

<sd:serviceData name="totalSwapSpaceSize" type="unsignedLong"
    minOccurs="0" maxOccurs="1" mutability="mutable" nillable="true"
    modifiable="true" crm:units="kilobytes"
    crm:valid="down starting up stopping failed"
    crm:changeable="down starting up stopping failed"
    crm:latency="whenStarted">
    <wsdl:documentation>
        Total swap space in Kbytes. This value may be NULL
        (unspecified) if swap space is not distinguished from page
        files. However, some Operating Systems distinguish these
        concepts. For example, in UNIX, whole processes can be 'swapped
        out' when the free page list falls and remains below a
        specified amount.
    </wsdl:documentation>
</sd:serviceData>

<sd:serviceData name="totalVirtualMemorySize" type="unsignedLong"
    minOccurs="0" maxOccurs="1" mutability="mutable" crm:units="kilobytes"
    crm:valid="down starting up stopping failed">
    <wsdl:documentation>
        Number of Kbytes of virtual memory. For example, this may
        be calculated by adding the amount of total RAM to the amount
        of paging space (ie, adding the amount of memory in/aggregated
        by the ComputerSystem to the property, SizeStoredInPagingFiles.
    </wsdl:documentation>
</sd:serviceData>

<sd:serviceData name="freeVirtualMemory" type="crm:gauge"
    minOccurs="0" maxOccurs="1" mutability="mutable" crm:units="kilobytes"
    crm:valid="starting up stopping failed"
    crm:volatile="1">
    <wsdl:documentation>
        Number of Kbytes of virtual memory currently unused and
        available. For example, this may be calculated by adding the
        amount of free RAM to the amount of free paging space (ie,
        adding the properties, FreePhysicalMemory and FreeSpace
        InPagingFiles).
    </wsdl:documentation>
</sd:serviceData>

<sd:serviceData name="freePhysicalMemory" type="crm:gauge"
    minOccurs="0" maxOccurs="1" mutability="mutable" crm:units="kilobytes"
    crm:valid="starting up stopping failed"
```

```
        crm:volatile="1">
        <wsdl:documentation>
            Number of Kbytes of physical memory currently unused and
            available.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="totalVisibleMemorySize" type="unsignedLong"
        minOccurs="0" maxOccurs="1" mutability="mutable" crm:units="kilobytes"
        crm:valid="starting up stopping failed"
        crm:volatile="1">
        <wsdl:documentation>
            The total amount of physical memory (in Kbytes) available to
            the OperatingSystem. This value does not necessarily indicate
            the true amount of physical memory, but what is reported to the
            OperatingSystem as available to it.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="sizeStoredInPagingFiles" type="unsignedLong"
        minOccurs="0" maxOccurs="1" mutability="mutable" modifiable="true"
        crm:units="kilobytes"
        crm:valid="down starting up stopping failed"
        crm:changeable="down starting up stopping failed"
        crm:latency="whenStarted">
        <wsdl:documentation>
            The total number of KBytes that can be stored in the
            OperatingSystem's paging files. 0 indicates that there are no
            paging files.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="freeSpaceInPagingFiles" type="crm:gauge"
        minOccurs="0" maxOccurs="1" mutability="mutable" modifiable="true"
        crm:units="kilobytes"
        crm:valid="down starting up stopping failed"
        crm:changeable="down starting up stopping failed"
        crm:latency="whenStarted">
        <wsdl:documentation>
            The total number of KBytes that can be mapped into the
            OperatingSystem's paging files without causing any other pages
            to be swapped out. 0 indicates that there are no paging files.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="maxProcessMemorySize" type="unsignedLong"
        minOccurs="0" maxOccurs="1" mutability="mutable" modifiable="true"
        crm:units="kilobytes"
        crm:valid="down starting up stopping failed"
        crm:changeable="down starting up stopping failed"
        crm:latency="afterResync">
        <wsdl:documentation>
            Maximum number of Kbytes of memory that can be allocated
            to a Process. For Operating Systems with no virtual memory,
            this value is typically equal to the total amount of physical
            Memory minus memory used by the BIOS and OS. For some Operating
            Systems, this value may be infinity - in which case, 0 should
            be entered. In other cases, this value could be a constant -
            for example, 2G or 4G.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="distributed" type="boolean" minOccurs="0"
        maxOccurs="1" mutability="setOnce"
        crm:valid="down starting up stopping failed">
        <wsdl:documentation>
            Boolean indicating whether the OperatingSystem is
            distributed across several ComputerSystem nodes. If so, these
            nodes should be grouped as a Cluster.
        </wsdl:documentation>
    </sd:serviceData>

    <sd:serviceData name="maxProcessesPerUser" type="unsignedInt"
        minOccurs="0" maxOccurs="1" mutability="mutable" modifiable="true">
        crm:valid="down starting up stopping failed"
        crm:changeable="down starting up stopping failed"
        crm:latency="afterResync">
        <wsdl:documentation>
            A value that indicates the maximum processes that a user can
            have associate with it.
        </wsdl:documentation>
    </wsdl:serviceData>

  </gwsdl:portType>

</wsdl:definitions>
```

Author Information

Ellen Stokes
IBM
11400 Burnet Rd
Austin TX 78758
Email: stokese@us.ibm.com
Phone: +1 512 838 0552

Nick Butler
IBM
MP188, Hursley Park,
Winchester,
Hampshire, SO21 2JN
United Kingdom
Email: nickb@uk.ibm.com
Phone: +44 1962 818081

Glossary

OGSi

gsdl

wsdl

References

[cimschema]	http://www.dmtf.org/standards/cim_schema_v26.php
[cimspec]	http://www.dmtf.org/standards/standard_cim.php
[gsspec]	http://www.globalgridforum.org/ogsi-wg/drafts/GS_Spec_draft03-2002-07-17.pdf
[wsdl]	<insert ref for draft 1.2>
[xmldatatypes]	http://www.w3c.org/TR/xmlschema-1
[xmlstructures]	http://www/w3c.org/TR/xmlschema-2

Acknowledgements

This is a preliminary list, in no particular order. Apologies to anyone who has been left off – it is certainly not intentional!

Steve Graham
Tom Maguire
Heather Kreger
Revathi Subramanian
Atul Patel
Juergen Schneider
Christine Draper
Andreas Maier
Helen Chen