GFD-R.069

Configuration description, Deployment and Lifecycle Management

2006-03-27

http://forge.gridforum.org/projects/cddlm-wg

**Editors:**

S. Loughran

# Configuration Description, Deployment, and Lifecycle Management

# CDDLM Deployment API

*Status of this Memo*

This document provides information to the community regarding the specification of the Configuration Description, Deployment, and Lifecycle Management (CDDLM) Language. Distribution of this document is unlimited.



## Abstract

Successful realization of the Grid vision of a broadly applicable and adopted framework for distributed system integration, virtualization, and management requires the support for configuring Grid services, their deployment, and managing their lifecycle. A major part of this framework is deployment APIs in which to express requests for the deployment. This document, produced by the CDDLM working group within the Global Grid Forum (GGF), provides the WSRF-based SOAP API for deploying applications to one or more target computers. The caller can upload files to it, then submit a deployment descriptor for deployment.

## CDDLM Specifications

There are five documents created by the CDDLM group. They are described in the text below.

The CDDLM Foundation document sets the stage for the remaining documents by introducing the area, by describing functional requirements, use cases, and high-level architecture. It also compares this group with other working groups as well as with other related efforts in the industry.

The SmartFrog Language spec [SF-CDL]describes a language primarily intended for configuration description and deployment. It is declarative, i.e. it supports attribute value pairs. Furthermore, it supports inheritance, references (including lazy), parameterization, predicates and schemas. It has the same functionality as CDL (see next paragraph), except that it is not XML-based. SmartFrog language predates CDL and it was used as a model when creating CDL. The two languages will be compatible. CDL is primarily intended for machines, SmartFrog for humans.

cddlm-wg@ggf.org

The CDDLM Configuration Description Language (CDL) [XML-CDL] is an XML-based language for declarative description of system configuration that consists of components (deployment objects) defined in the CDDLM Component Model. The Deployment API uses a deployment descriptor in CDL in order to manage deployment lifecycle of systems. The language provides ways to describe properties (names, values, and types) of components including value references so that data can be assigned dynamically with preserving specified data dependencies. A system is described as a hierarchical structure of components. The language also provides prototype-based template functionality (i.e., prototype references) so that the user can describe a system by referring to component descriptions given by component providers.

The CDDLM Component Model [CDDLM-CMP] outlines the requirements for creating a deployment object responsible for the lifecycle of a deployed resource. Each deployment object is defined using the CDL language and mapped to its implementation The deployment object provides a WS-ResourceFramework (WSRF) compliant "Component Endpoint" for lifecycle operations on the managed resource. The model also defines the rules for managing the interaction of objects with the CDDLM Deployment API in order to provide an aggregate, controllable lifecycle and the operations which enable this process.

The deployment API is the WSRF-based SOAP API for deploying applications to one or more target computers. Every set of computers to which systems can be deployed hosts one or more "Portal Endpoints", WSRF resources which provide a means to create new "System Endpoints". A System Endpoint represents a deployed system. The caller can upload files to it, then submit a deployment descriptor for deployment. A System Endpoint is effectively a component in terms of the Component Model specification -it implements the properties and operations defined in that document. It also adds the ability to resolve references within the deployed system, enabling remote callers to examine the state of components with it.

# 1 Table of Contents

# 2 Introduction

The CDDLM framework needs a deployment API for callers to deploy services described in the component model languages. This API must support remote access for deployment, terminating existing deployed systems, and for probing their state

This document defines the WS-Resource Framework-based deployment API for performing such tasks. It is targeted at those who implement either end of the API.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 2.1.1 Configuration Description Language

The CDDLM Configuration Description Language (CDL) [XML-CDL] is an XML-based language for declarative description of system configuration that consists of components (deployment objects) defined in the CDDLM Component Model. The Deployment API uses a deployment descriptor in CDL in order to manage deployment lifecycle of systems. The language provides ways to describe properties (names, values, and types) of components including value references so that data can be assigned dynamically with preserving specified data dependencies. A system is described as a hierarchical structure of components. The language also provides prototype-based template functionality (i.e., prototype references) so that the user can describe a system by referring to component descriptions given by component providers.

### 2.1.2 Component Model

The CDDLM Component Model [CDDLM-CMP] outlines the requirements for creating a deployment object responsible for the lifecycle of a deployed resource. Each deployment object is defined using the CDL language and mapped to its implementation. The deployment object provides a WS-ResourceFramework (WSRF) compliant "Component Endpoint" for lifecycle operations on the managed resource. The model also defines the rules for managing the interaction of objects with the CDDLM Deployment API in order to provide an aggregate, controllable lifecycle and the operations which enable this process.

### 2.1.3 Deployment API

The Deployment API is the WSRF-based SOAP API for deploying applications to one or more target computers. Every set of computers to which systems can be deployed hosts one or more "Portal Endpoints", WSRF resources which provide a means to create new "System Endpoints". A System Endpoint represents a deployed system. The caller can upload files to it, then submit a deployment descriptor for deployment. A System Endpoint is effectively a component in terms of the Component Model specification - it implements the properties and operations defined by the model. It also adds the ability to resolve references within the deployed system, enabling remote callers to examine the state of components with it.

### 2.2 XML Namespaces used in this document

Throughout the document, the following prefixes refer to the listed namespaces:

| prefix | URI | description |
|---|---|---|
| xsd | http://www.w3.org/2000/10/XMLSchema | XML Schema Types |
| wsa | http://schemas.xmlsoap.org/ws/2003/03/addressing | WS-Addressing types |
| api | http://www.gridforum.org/cddlm/deployapi/2005/02 | Deployment API types |
| apiw | http://www.gridforum.org/cddlm/deployapi/2005/02/wsdl | Deployment API WSDL |
| cdl | http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0 | XML CDL |
| cmp | http://www.gridforum.org/cddlm/components/2005/02 | Component Model |
| wsrf-bf | http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.xsd | WS-BaseFaults |
| wsrf-rl | http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd | WS-Resource Lifetime |
| wsrf-rlw | http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl | WS-Resource Lifetime WSDL |
| wsrf-rp | http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd | WS Resource Properties |
| wsrf-rpw | http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl | WS Resource Properties WSDL |
| wsnt | http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd | WS-BaseNotification |
| wsrf-top | http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd | WS-Topics |
| s12 | http://www.w3.org/2003/05/soap-envelope | SOAP1.2 Envelope |
| muws-p1-xs | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd | Management using Web Services part 1 |

### 2.3 Document Structure

Chapters 1-3, 10, 13 and Appendix B are informative. Chapters 4-9, 11, 12 and appendices A and C are normative. Chapter 14, the bibliography, is divided into normative and informative references.

## 3 Purpose of the Deployment API

The deployment API is the WS-ResourceFramework (WSRF) API for deploying applications to one or more target computers, physical or virtual.

The API is written assuming that the end user is deploying through a console program, a portal UI or some automated process. When deploying onto an OGSA™

grid fabric, that is a set of machines supporting the resource management and scheduling facilities of the OGSA architecture, the deployment API performs the task of instantiating user-specified applications onto hosts selected by the resource manager. It thus takes on a back-end role, rather than something client applications will be expected to interact with directly.

### 3.1 Use Cases

There are three different core use cases of the deployment API:

1 The deployment target is an OGSA Grid Fabric. Resource allocation and Job submission (using the JSDL language [JSDL] or equivalent) is part of the deployment process. In this use case, the deployment API integrates with the other OGSA services to deploy a CDDLM-language described system over the machines allocated by the resource manager.

2 The deployment target is a pre-allocated set of machines. The resource allocation process is bypassed —it can be presumed to have happened out of band. The user needs to upload data files to the cluster as part of the deployment.

3 One instance of a CDDLM runtime is delegating part of a deployment to another host. There is no guarantee that the two runtimes are the same implementation of CDDLM, or, if they are, that they are the same version.

### 3.2 File upload

Part of remote deployment often consists of providing files to the remote systems, both code and data. The preferred solution to this is a remote asset store of some form, with an efficient transport and secure, version-based access to assets. This is not something that falls within the scope of this working group, and has not been addressed here.

As an interim solution, pending the availability of such systems, the deployment API provides a means to submit files to the remote system. These files are submitted in the request, and a URL of type (`file:`, `http:` or `https:`) is returned. The URL can be used within the deployment descriptor, and passed to the applications.

Uploaded files remain present for the lifespan of the deployed application. There is no sharing between deployed applications, no way to update a file, and no way to delete a file. Clearly, therefore, it is a pale substitute for a full asset store —and that is its deliberate intent. When deploying to an infrastructure that hosts a full asset store, that store and its remote upload API SHOULD be used instead of the file upload mechanism described in this document.

## 4 Architecture

### 4.1 Core Architecture

The API comprises a model for deployment, and a WS-ResourceFramework [WSRF] based means of interacting with this model.

A *deployment client* is an application that wishes to use the deployment API to deploy to one or more hosts that have been pre-allocated using a resource allocation system. A *deployment portal* is a WSRF service endpoint that the deployment client communicates to, in order to deploy applications, an endpoint addressed via a WS-Addressing Endpoint Reference (EPR) [WS-A]. This specific EPR is referred to as the

*Portal EPR*. The actual process for obtaining a Portal EPR is not in the scope of this document; it is assumed to have been obtained by some means.

To deploy, the client first issues a request to the *Portal Endpoint* to create a system. This request includes a deployment descriptor in one of the CDDLM supported languages and potentially other information that describes and configures the application. This creation request returns a new EPR, which provides access to the state and operations of the system, the *System EPR.*

The *System EPR* can be bound to a *System Endpoint* hosted on any node that the Portal Endpoint chooses; there is no requirement that it is bound to the same node as the portal. For maximum availability, hosting the system endpoint on the same node that hosts the system may be the best approach. Figure 1 shows an example of this.

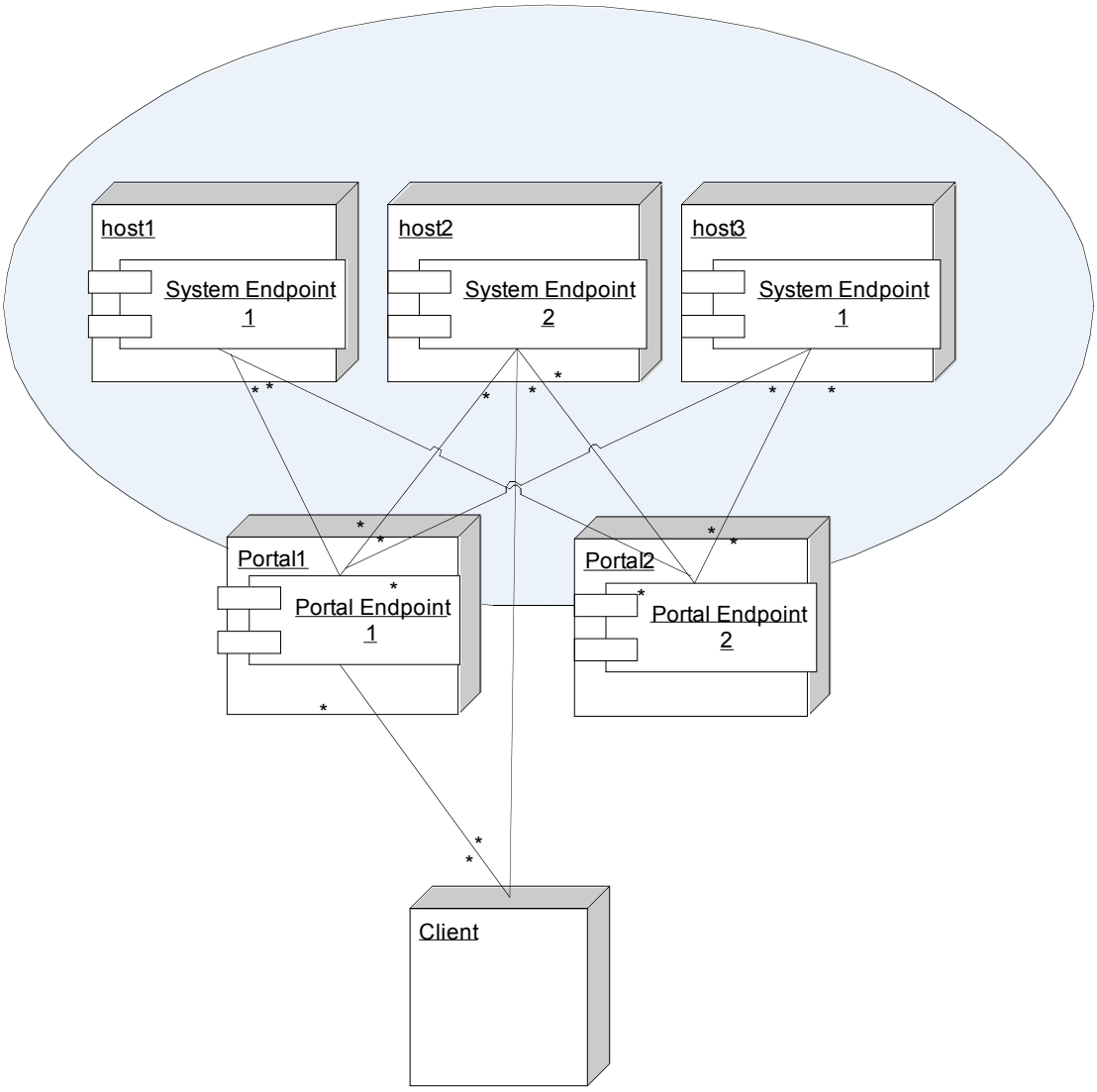


**Figure 1 . Model of how Portal and System endpoints may be distributed. Multiple Portals can manage the same set of deployment nodes. The client has references (EPRs) to the portals and systems that it is working with.**

The caller can then make a request to the *System Endpoint* to initialize the system. If successful, the application asynchronously enters the next state in its lifecycle,

*initialized*. Once a system has been initialized, it can be requested to enter through other stages of its lifecycle.

As a deployed system changes state, it sends lifecycle event notification messages to registered listeners, using a mechanism such as WS-BaseNotification [WS-BaseNotification]. The state of the system can also be determined by querying the appropriate resource property of the system, using the mechanism defined in the WS-Resource Properties [WS-ResourceProperties] specification. There is also a synchronous, blocking call to probe the health of a system; this MUST be routed to the system itself, so that it can determine its own health. This will return its current state, and any custom status information the system chooses to return. If the system has failed, or terminated after a failure, the status information will include the fault information.

The Portal Endpoint supports other properties and operations. The list of currently deployed systems can be determined, along with their system EPRs. There are also static information and dynamic information documents which can be retrieved from the server; again these are represented as properties following the WS-Resource Properties specification.

The Portal Endpoint can raise events when new systems are created, using the WS-BaseNotification protocol.

### *4.2 System State*

CDDLM components have a uniform lifecycle, which is normatively described in the component model specification [Schaeffer05]. The state of a system mirrors that of the lifecycle of the components within. This is essential to permit aggregation of systems.

The key difference is that state changing operations are asynchronous and non-blocking. A request to initialize, start or terminate a deployed system is received by the System Endpoint, validated, and, if valid, queued for execution.

The states of a system are as follows:

| | |
|---|---|
| *instantiated* | The system has just been instantiated. |
| *initialized* | The system has been initialized. |
| *running* | The system is running |
| *failed* | The system has failed |
| *terminated* | The system has terminated |

Instantiation and initialization represent the creation and configuration of a system. When it is moved into the *running* state, then it is actually functional. The state *failed* is entered automatically when a failure is detected. Termination is the only exit condition. *Terminated* is the end state of a component and can be entered through a termination request.

The Portal's *Create* operation creates and instantiate a system, a system which can then be directly manipulated via requests to its System Endpoint. The *Initialize* operation will then initialize a system in accordance with the provided deployment descriptor. A subsequent *Run* operation will move the system to the running state, and *Terminate* will move it to the terminated state.

If the initialization process encounters an error, or, while running, a system detects that it or its underlying components have failed, it will move to the failed state. From here, the sole state-changing operation is again *Terminate*.

The endpoint of a terminated system should still respond to until the endpoint is destroyed through the `<wsrf:Destroy/>` operation, or until the portal purges its set of terminated systems. Once a System Endpoint is destroyed, the system and any associated resources are no longer accessible.

## 4.3 Fault Tolerance

The architecture is intended to enable fault-tolerant implementations, to the extent that a failure of the deployment endpoint may not terminate the application, and may not render the application unreachable. As stated, the architecture must enable fault-tolerant implementations. Here is how the design permits, but does not require, a fault-tolerant implementation:

- Multiple Portal Endpoints can provide access to the same set of nodes.
- The failure of a portal does not imply the failure of a system.
- The failure of a node hosting a System Endpoint MUST result in the destruction of that system.
- Issuing a `<wsrf-rl:Destroy>` request to a System Endpoint MUST terminate and destroy the system.
- Every system instance MUST have a WSRF property `muws-p1-xs:ResourceId` of type `xsd:anyURI` property that MUST be unique; this enables a new System EPR for a known system to be regenerated by querying a different portal.
- Implementations MAY implement fault tolerant EPRs through the use of a dynamic DNS service, one in which the DNS entries for the hostname(s) of the portal are updated as portal instances appear and disappear. It is RECOMMENDED that Client systems are written with the knowledge that the IP addresses of an EPR MAY change, and not to cache resolved IP addresses indefinitely.

## 4.4 Other Architectural Features

### 4.4.1 Deployment Language Agnostic

The deployment API is agnostic as to which particular language, or version thereof, is used for a deployment descriptor. When a remote deployment is created, the language and version of the descriptor MUST be supplied. The sole requirement of a language is that it can either be nested inside an XML document, or that a URL to the descriptor is remotely accessible to the destination. In the case of the latter, the URL to the descriptor MUST be provided when initializing the system.

Every language is identified by a unique URI. This language URI MUST be supplied with the deployment descriptor or URI. A list of supported languages can be obtained from a Portal Endpoint.

The URIs for the CDDLM languages are listed in Appendix B.

### 4.4.2 Job Language Agnostic

Just as the API allows implementations to support deployment languages/versions, the API also permits multiple job description languages. The processing of information

provided in these languages is implementation-specific, and not covered in this document.

### 4.4.3 Extensibility

The deployment API is designed to support extensible implementations, and future enhancements to the API over time.

#### 4.4.3.1 Extra Operations

A service implementation may offer extra operations at any EPR.

1 Private extensions MUST NOT add new declarations to the XML namespaces used in this document: they MUST be in their own, private, namespace.
2 Implementations SHOULD document these operations and provide updated WSDL descriptions.
3 There is no requirement for the extra operations supported by an EPR to remain constant over any period of time. They may even change during the period in which an EPR remains valid.

#### 4.4.3.2 Extra WSRF operations and WS-Resource Properties

This specification and the accompanying XSD/WSDL documents define the minimum set of WSRF operations and properties that an endpoint MUST implement. There are other operations that the WSRF specification family and WSDM list as optional.

An implementation may extend the WSDL to support extra functionality. Such extensions MUST NOT be in the namespaces of the deployment API.

#### 4.4.3.3 Extra deployment options

Extra deployment options MAY be offered on different implementations. The core of such customization should be in deployment descriptors themselves, yet there MAY be a need to provide extra deployment metadata. As an example, there may be late-binding information mapping hostname aliases in the deployment descriptor to the names of hosts used in the actual deployment.

Customization and late-binding information is supported through an `<options>` element in the `<initialize>` message. This (optional) element contains a list of zero or more deployment options. These are extra parameters to the deployment request. Every option is named with a URI, and can have a string or integer attribute value, or contain nested XML. The `mustUnderstand` attribute is used to indicate whether or not an option must be understood.

Implementations are REQUIRED to interpret these options as follows:

- Every option MUST be named by a URI.

All URIs that begin with `http://gridforum.org/cddlm/` or `http://ggf.org/schemas/cddlm` are reserved for options defined by the CDDLM working group.

- Options MUST contain either string, integer, Boolean or arbitrary XML values. String and integer values are supported via attributes; XML is supported as nested data.
- An option MUST contain only one value type. Implementations MUST raise a fault if multiple nested or attribute values are declared on the same option.

- All options that an implementation supports MUST be enumerated in the server information property of the Portal Endpoint.
- It is an error to include multiple options of the same URI in a descriptor. Implementations MUST raise a fault when this occurs.
- Options MUST NOT require a specific order of processing. Options may be processed in any order.
- Service implementations MUST ignore any options that they do not recognize, if `mustUnderstand="false"` for that option.
- Service implementations MUST understand all options which are supplied with `mustUnderstand="true"` for that option. If any such option is not understood, a fault MUST be raised.

This leads to the following processing rules:

1. Option processing MUST take place before the system is moved to the running state.
2. Any option that is marked `mustUnderstand="true"` MUST be understood. If not, the Fault `"not-understood"` MUST be raised, identifying the particular option by its URI in the `extraData` field of the fault.
3. Implementations MUST NOT raise this fault when they do not understand any options that are marked `mustUnderstand="false"`, or for which there is no `mustUnderstand` attribute. These MUST be ignored.
4. Duplicate options MUST cause the operation to be rejected with a `bad-argument` fault, identifying the particular option by its URI in the `extraData` field of the fault.

### *4.5 WS-DM Integration*

The deployment infrastructure is designed to integrate with a WS-DM management framework. Both Portal and System endpoints MUST support the MUWS `ResourceId` and `ManageabilityCapability` attributes to uniquely identify each endpoint and to enumerate their management capabilities. All supported events are derived from the MUWS event type, and the state type of a System Endpoint is derived from the MUWS state types.

Implementations MAY add more management capabilities, as they see fit. For example, a Portal endpoint could export the MUWS capability `OperationalState` that describes the portal's operational state, then provide an event that notifies listeners of changes in that state.

## 5 Deployment API Overview

The Deployment API consists of two endpoint types, Portal Endpoints, addressed by Portal EPRs, and System Endpoints, addressed by System EPRs. Portal Endpoints are the mechanism by which new System EPRs are created, or existing Systems and their corresponding EPRs located.

The two endpoint types are *Resources* within the terminology of the WS-Resource Framework specifications.

## 5.1 Conceptual Model

The following UML diagram depicts the relationships of the Deployment API endpoints to the WSRF and WSDM concepts

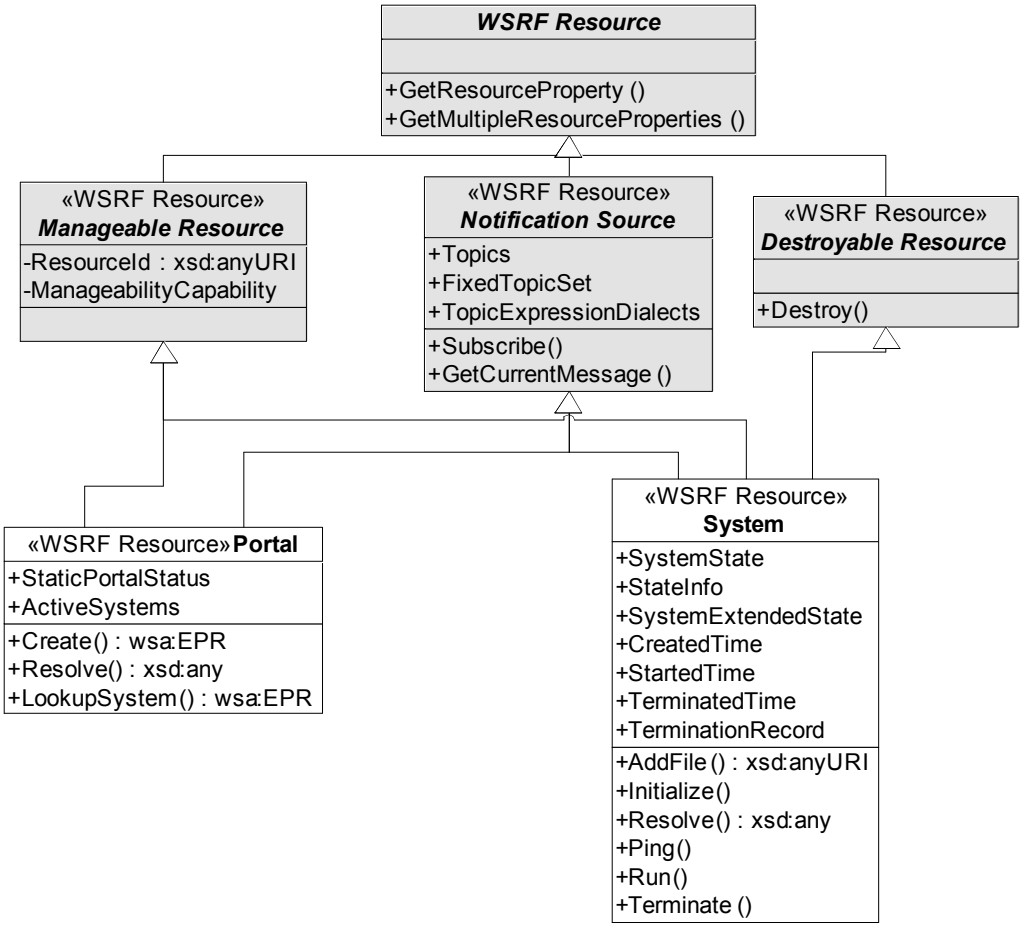


**Figure 2 . Conceptual model of the deployment system. The items in grey are specified externally.**

## 5.2 Portal Endpoint

The *portal endpoint* is the endpoint that the caller initially locates and communicates with. It can be used to create a new system within the set of nodes that it has coverage of, it can be used to locate an existing system, and it can be used as a source of system creation events.

### 5.2.1 Portal Endpoint Properties

| *Name* | *Type* | *Meaning* |
|---|---|---|
| `muws-p1-xs:ResourceId` | `xsd:anyURI` | Unique identifier of the portal. |
| `muws-p1-xs: ManageabilityCapability` | `xsd:anyURI list` | List of supported manageability capabilities. |

| | | |
|---|---|---|
| api:StaticPortalStatus | api:StaticPortalStatusType | Static portal information; constant for the lifetime of the portal itself. |
| api:ActiveSystems | api:SystemReferenceListType | List of System EPRs. |
| wsnt:Topics | wsnt:TopicExpressionType | List of supported notification topics. |
| wsnt:FixedTopicSet | xsd:boolean | A Flag to indicate whether the topic set is fixed. |
| wsnt: TopicExpressionDialects | xsd:anyURI | Dialect of the topicset. |

### 5.2.2 Portal Endpoint Operations

| *Name* | *In* | *Out* |
|---|---|---|
| Create | hostname | wsa:EPR |
| | Create a system; hostname is optional. | |
| LookupSystem | resourceId | wsa:EPR |
| | Look up a single system returning its System EPR. | |
| Resolve | resourceId, path | xsd:any |
| | Lookup a system and resolve a path against it. | |
| GetResourceProperty | wsrf-rp: GetResourcePropertyRequest | wsrf-rp: GetResourcePropertyResponse |
| | Get the value of a resource. | |
| GetMultipleResourceProperties | wsrf-rp: GetMultipleResource↵ PropertiesRequest | wsrf-rp: GetMultipleResource↵ PropertiesResponse |
| | Read multiple resources. | |
| Subscribe | wsnt:Subscribe | wsnt:SubscribeResponse |
| | Subscribe to events. | |
| GetCurrentMessage | wsnt: GetCurrentMessageRequest | wsnt: GetCurrentMessageResponse |
| | Get the current message for a topic. | |

## *5.3 System Endpoint*

This represents a deployed system. System EPRs are obtainable by creating a system at the Portal Endpoint, or through a `LookupSystem` operation offered by the Portal.

### 5.3.1 System Endpoint Properties

| *Name* | *Type* | *Meaning* |
|---|---|---|
| muws-p1-xs:ResourceId | xsd:anyURI | unique identifier |
| muws-p1-xs: ManageabilityCapability | xsd:anyURI | List of supported manageability capabilities. |
| api:SystemState | cmp:componentStatusType | current system state. |
| api:CreatedTime | xsd:dateTime | Time system was created. |
| api:StartedTime | xsd:dateTime | Time system moved into the running state. |
| api:TerminatedTime | xsd:dateTime | end time (not present until system is terminated). |
| api:TerminationRecord | cmp:TerminationRecordType | termination record (present after termination). |
| wsnt::Topics | wsnt:TopicExpressionType | List of supported notification topics. |
| wsnt:FixedTopicSet | xsd:boolean | A Flag to indicate whether the topic set is fixed. |
| wsnt:TopicExpressionDialects | xsd:anyURI | Dialect of the topicset. |

### 5.3.2 System Endpoint Operations

| *Name* | *In* | *Out* |
|---|---|---|
| Initialize | job, descriptor | void |
| | Initialize a system; pass in the job and component descriptors and build up the component graph. | |
| AddFile | mimetype, data | xsd:anyURI |
| | Add a file to this document so that it is accessible by a URI from within the deployment descriptor. | |
| Run | void | void |
| | Start running an initialized system. | |
| Ping | void | StatusType |

| *Name* | *In* | *Out* |
|---|---|---|
| | Probe a system's health. | |
| `Resolve` | `path` | `xsd:any` |
| | Resolve a reference relative to this system. Can return EPRs to components; string or other data. | |
| `Terminate` | `Message` | `void` |
| | Terminate a system; pass in a message. | |
| `wsrf-rl:Destroy` | | |
| | Destroy the System Endpoint, terminating the System if it is not yet terminated. | |
| `GetResourceProperty` | `wsrf-rp: GetResourcePropertyRequest` | `wsrf-rp: GetResourcePropertyResponse` |
| | Get the value of a resource | |
| `GetMultipleResourceProperties` | `wsrf-rp: GetMultipleResourcePropertiesRequest` | `wsrf-rp: GetMultipleResourcePropertiesResponse` |
| | Read multiple resources. | |
| `wsnt:Subscribe` | `wsnt:Subscribe` | `wsnt:SubscribeResponse` |
| | Subscribe to events. | |
| `GetCurrentMessage` | `wsnt: GetCurrentMessageRequest` | `wsnt: GetCurrentMessageResponse` |
| | Get the current message for a topic. | |

# 6 Portal

## *6.1 Portal Properties*

### 6.1.1 muws-p1-xs:ResourceId

This is a MUWS-defined property. It contains a URI that MUST uniquely identify this instance of a Portal Endpoint.

### 6.1.2 muws-p1-xs:ManageabilityCapability

This is a MUWS-defined (multiple) property that lists all MUWS-related management features implemented in this endpoint.

The minimum set of properties that an endpoint MUST report are the `ResourceId` and `ManageabilityCapability` facilities themselves, as normatively described in the MUWS specification(s), and a capability that declares the endpoint as a CDDLM Portal Endpoint, the latter with the URI
`http://www.gridforum.org/cddlm/deployapi/2005/02/capabilities/portal`

The non-normative listing of the capabilities is:

```
<muws-p1-xs:ManageabilityCapability>
```

```
http://docs.oasis-
open.org/wsdm/2004/12/mows/capabilities/ManageabilityReferences
</muws-p1-xs:ManageabilityCapability>
<muws-p1-xs:ManageabilityCapability>
http://docs.oasis-
open.org/wsdm/2004/12/muws/capabilities/ManageabilityCharacteristics
</muws-p1-xs:ManageabilityCapability>
<muws-p1-xs:ManageabilityCapability>
http://www.gridforum.org/cddlm/deployapi/2005/02/capabilities/portal
</muws-p1-xs:ManageabilityCapability>
```

### 6.1.3 api:StaticPortalStatus

This property contains static portal information; information constant for the lifetime of the portal instance. The elements contain static diagnostics information, specific to the implementation, and enumerations of features that supported by that implementation

A non-normative example of the status is:

```
<api:StaticPortalStatus>
  <api:portal>
    <api:name>
      Example Portal implementation
    </api:name>
    <api:build>Version 4.3 built 2005-06-21</api:build>
  </api:portal>
  <api:languages>
    <api:item>
      <api:name>CDL</api:name>
      <api:uri>
http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0</api:uri>
    </api:item>
  </api:languages>
  <api:joblanguages/>
  <api:notifications>
    <api:item>
http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-
01.xsd
    </api:item>
  </api:notifications>
  <api:options />
</api:StaticPortalStatus>
```

### 6.1.4 api:ActiveSystems

This is a list of deployed systems that the portal is aware of. This MAY include systems which the portal did not deploy, but which a peer portal has deployed. It MAY also be restricted to those systems to which the caller has access rights. Network partitioning and other events MAY cause systems to be temporarily invisible to this list, and return later. Callers SHOULD view the list not as complete and accurate, but as a snapshot enumeration of all deployed systems that were visible at the time the request was processed.

A non-normative example of the message is:

```
<api:ActiveSystems>
  <api:system>
      <wsa:Address>http://example.org/resources/2</wsa:Address>
  </api:system>
  <api:system>
      <wsa:Address>http://example.org/resources/3</wsa:Address>
  </api:system>
</api:ActiveSystems>
```

#### 6.1.5 wsnt:Topic, wsnt:FixedTopicSet, and wsnt:TopicExpressionDialects

These three properties are published in adherence with the WS-BaseNotification specification.

### *6.2 Operations*

#### 6.2.1 Create([hostname])

This requests the creation of a new system instance, ready for configuration.

The `hostname` element specifies an optional hostname hint. If set, it nominates a host onto which the port MAY instantiate the system, and MAY also instantiate the System Endpoint. The portal MAY instantiate the system on any host of its choosing. Thus, `hostname` is merely a hint, a hint to improve availability and performance. A non-normative example of a request is as follows:

```
    <api:createRequest>
      <api:hostname>localhost</api:hostname>
    </api:createRequest>
```

This request expresses a preference for the system to be created on the same host as the portal.

The successful response is a System EPR to the newly-created System, an EPR which can be immediately used for direct communications. Creation of a System Endpoint is therefore a synchronous operation.

If any entity is registered with the portal for creation events, then the portal MUST send notification to that entity that new system has been created. The notification MUST NOT be sent until the system is ready for direct communication. There is no specification of the ordering of returning from the `create` operation and the sending of any notification mechanism. If there are multiple portals supporting deployment to a cluster of nodes, notification events MAY be sent to listeners on one portal, even if the deployment was requested on the other.

#### 6.2.2 LookupSystem(ResourceId:uri)

This operation resolves a `ResourceId` to a system, and returns a System EPR.A non-normative example request is:

```
  <api:lookupSystemRequest>
    <api:ResourceId>http://example.org/systems/456</api:ResourceId>
  </api:lookupSystemRequest>
```

This permits a caller to obtain an EPR to a system whose `ResourceId` is known, and thus acquire endpoints from multiple portal endpoints providing uniform access to systems deployed over a set of hosts.

A non-normative example response is

```
<api:lookupSystemResponse>
  <wsa:Address>http://example.org/resources/2</wsa:Address>
</api:lookupSystemResponse>
```

#### 6.2.3 Resolve(ResourceId:uri, path:string)

This operation resolves a `ResourceId` to a system, and then resolves a path against it. It has the same semantics as using `LookupSystem` to obtain an EPR, then invoking `Resolve` on that EPR. A non normative example of the message of a resolve request is:

```
    <api:portalResolveRequest>
      <api:ResourceId>http;//example.org/resource/7</api:ResourceId>
      <api:path>/components/server/proxy/port</api:path>
    </api:portalResolveRequest>
```

A non-normative example response is the following:

```
    <api:portalResolveResponse>
      <other:a xmlns:other="http://example.org/other">
        XML in another namespace
      </other:a>
    </api:portalResolveResponse>
```

Note that the type of the messages are `portalResolveRequest` and `portalResolveResponse` respectively; this is to avoid confusion with the messages used in the resolve operation of individual System endpoints.

#### 6.2.4 GetResourceProperty/GetMultipleResourceProperties

These two operations are defined by the WS-ResourceProperties specification.

#### 6.2.5 Subscribe/GetCurrentMessage

These two operations are defined by the WS-BaseNotification specification.

# 7 System

A System Endpoint represents a deployed system.

## *7.1 System Properties*

#### 7.1.1 muws-p1-xs:ResourceId

This is a MUWS-defined property. It contains a URI that is unique to a particular instance of a system..

#### 7.1.2 muws-p1-xs:ManageabilityCapability

This is a MUWS-defined (multiple) property that lists all MUWS-related management features implemented in this endpoint.

The minimum set of properties that an endpoint MUST report are the `ResourceId` and `ManageabilityCapability` facilities themselves, as normatively described in the MUWS specification(s), and a capability that declares the endpoint as a CDDLM System Endpoint, the latter with the URI
`http://www.gridforum.org/cddlm/deployapi/2005/02/capabilities/system`

The non-normative listing of the capabilities is:

```
<muws-p1-xs:ManageabilityCapability>
http://docs.oasis-
open.org/wsdm/2004/12/mows/capabilities/ManageabilityReferences
</muws-p1-xs:ManageabilityCapability>
<muws-p1-xs:ManageabilityCapability>
http://docs.oasis-
open.org/wsdm/2004/12/muws/capabilities/ManageabilityCharacteristics
</muws-p1-xs:ManageabilityCapability>
<muws-p1-xs:ManageabilityCapability>
http://www.gridforum.org/cddlm/deployapi/2005/02/capabilities/system
</muws-p1-xs:ManageabilityCapability>
```

#### 7.1.3 api:SystemState

This is the state of the system, as represented by the `componentStatusType` of the CDDLM component model.

#### 7.1.4 api:CreatedTime, api:StartedTime and api:TerminatedTime

These are all `xsd:dateTime` timestamps of when a system entered a particular state.

#### 7.1.5 api:TerminationRecord

This is a `cmp:terminationRecordType` element. It contains information about the reason for the system's termination. It is only present after a system is terminated.

#### 7.1.6 wsnt:Topic, wsnt:FixedTopicSet and wsnt:TopicExpressionDialects

These three properties are published in adherence with the WS-BaseNotification specification.

### *7.2 System Operations*

#### 7.2.1 AddFile

This request uploads a file to the infrastructure, such that it is visible by deployed programs, and by the System Endpoint itself.

The request MAY include the file as base-64 encoded data in the `data` element. Unless both ends of the communication are specially written to stream large base-64 elements, the `addFileRequest/data` contents SHOULD be sent using the MTOM transmission mechanism. If DIME or Soap with Attachments is used then the attachment MUST be assigned a URI, a URI that MUST then be declared in the `addFileRequest/uri` element. Any attachment support mechanism MAY be implemented.

A URI can also be a URL to a remote file, a file which MUST be retrieved by the deployment and made available to deployed systems. Implementations SHOULD resolve remote URL references in the `addFileRequest/uri` element, using the delegated identity of the created job to grant access rights.

The `schema` element in the message is a declaration of the required type of the returned URL. All implementations MUST support the `http` scheme, and MAY support `https` or `file`. Implementations MAY also support other schema types. If a caller requests a schema type that is not supported, the implementation MUST reject the request.

The request supports a metadata element that contains arbitrary XML. This could be RDF metadata, file hash values for efficient re-use, or even file lifetime hints. All such metadata is implementation-specific, and is not defined here.

A non-normative example of a request is as follows:

```
<api:addFileRequest>
  <api:name>urn:numbers://46</api:name>
  <api:mimetype>application/x-ssh-key</api:mimetype>
  <api:scheme>http</api:scheme>
  <api:data>
  AAAAB3NzaC1yc2EAAAABIwAAAIEAwVmUkPzXdWEyJZ8nCR8GvdrDtO00RI4Z
  Bg3Gyviuz5IrWj2C6b2BdcKn+S/swDV1fiEFY4+ewYHUfmg+UKm2T8Lfksjn
  Hinks0GoVvkwy3bF48U5yVk1akAzR5YbSLJa6Naj8XS9681xVzWpbjxrV3KR
  QNWvEqI0MqRE34MzT4M=
  </api:data>
  <api:metadata>
    <x:expires date="2005-07-18" xmlns:x="http://example.org/expiry" />
  </api:metadata>
</api:addFileRequest>
```

Note that the WSDL accompanying this document does not declare how binary attachments are to be sent with the document. If the MTOM transmission mechanism is used [MTOM], then the `addFileRequest/data` element MUST contain the binary-data.

The response MUST be a list of one or more URIs. Each URI MUST be a URL in the requested schema that MUST resolve to the uploaded file. Each URL MUST be resolvable to all components and programs deployed as part of this system. It MAY be visible to other programs running with the same credentials. If exposed as a file:

URL, it MUST be visible, read-only to all processes of the system, on any node in the network onto which it has been deployed.

The lifespan of the uploaded file is bound to that of the created system; when the System Endpoint is destroyed, all uploaded files SHOULD be destroyed.

A non-normative example of a response message is as follows:

```
    <api:addFileResponse>
      <item>http://server/job5/files/1</item>
      <item>http://server2/job5/files/1</item>
    </api:addFileResponse>
```

In this response, two separate URLs to the uploaded file have been supplied, both using the http scheme and hence the HTTP protocol.

### 7.2.2 Initialize

This is a complex request, as it configures the system and moves it into the *initialized* state.

A non-normative example request message is as follows:

```
<api:initializeRequest>
  <api:descriptor
    language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
    <api:reference>http://server/descriptor.cdl</api:reference>
  </api:descriptor>
  <api:options>
    <api:option name="http://example.org/options/1" >
      <api:string>string value</api:string></api:option>
    <api:option name="http://example.org/options/2">
      <api:integer>-8</api:integer>
    </api:option>
    <api:option name="http://example.org/options/3"
       mustUnderstand="true">
       <api:boolean>true</api:boolean>
    </api:option>
    <api:option name=
"http://gridforum.org/cddlm/serviceAPI/options/propertyMap/2004/07/30"
       mustUnderstand="false">
      <api:propertyMapOption>
        <api:property>
          <api:name>jvm.maxmemory</api:name>
          <api:value>64M</api:value>
        </api:property>
        <api:property>
          <api:name>jvm.minmemory</api:name>
          <api:value>16M</api:value>
        </api:property>
        <api:property>
          <api:name>max.users</api:name>
          <api:value>128</api:value>
        </api:property>
      </api:propertyMapOption>
    </api:option>
  </api:options>
</api:initializeRequest>
```

A deployment descriptor MUST be supplied; it consists of a language URI, and either an inline deployment descriptor or a URL to a location where the descriptor can be retrieved.

An OPTIONAL `<job>` element contains the job description that was used when submitting the job to the front-end portal. As with the `<descriptor>`, it is of type `descriptorType`; it MUST have a language URI and either an inline body or a URL to

the descriptor. The interpretation of this data by the service implementation is undefined.

The OPTIONAL `<options>` element contains a list of zero or more configuration options. These are late-binding parameters to the deployment request, or to the deployment runtime.

When the request message is received, the System Endpoint MUST validate it synchronously and initialize the system. For CDDLM implementations, initialization implies that the deployment descriptor and job descriptor will be parsed. The application is then configured, and resolution begins. If successful, the system enters the *initialized* state. This is an asynchronous operation.

The response to a successful request is an empty response, `<initializeReponse/>`. Its presence implies that the initial validation was successful, and that initialization has begun, or has at least been scheduled.

If an initialize request is received and the application is in any state other than created, an error MUST be raised.

If the request is received while the application is already initializing itself, an error MUST be raised.

#### 7.2.2.1 The propertyMap schema type

To aid those options that take a map of name/value pairs, there is a predefined XML Schema type, `<api:propertyMap>`, that can describe the construct:

The `propertyMap` elements can be placed into the `<data>` child element of an option. Both the name and value of a `propertyTuple` within a `propertyMap` element are of type `xsd:string`; individual options are free to declare extra restrictions on the value of properties, restrictions that can be validated when processing the option.

There is no requirement that the name/value pairs are unique within a `propertyMap` element; that is also a restriction that can be declared in a specification of a particular option.

### 7.2.3 Run

This request runs a system. This triggers an asynchronous action, as it may take some time to enter the running state. It is only valid from a state in which the lifecycle permits running to be reached; *initialized* and, implicitly, *running*. In the case of the latter, the operation is a no-op. If the system is initializing itself, as a result of an `Initialize` request, the request SHOULD be queued for processing after the state transition is completed.

The response on success is an empty `RunResponse` message. Its receipt means that the system has been queued to enter the running state asynchronously.

### 7.2.4 Ping

This is a synchronous request to the system, to query its health.

If the system is not running, the System Endpoint MUST return with the current state, as defined in the component model. In these states, the System Endpoint is free to provide whatever extended state information that it chooses.

If the system is running, the request MUST be forwarded to the deployed system, which can return any extended state information, alongside the state “running”. This

acts as a liveness test upon the system. A successful response to the call implies that the system considers itself healthy.

### 7.2.5 Resolve

This operation resolves a path of type `cdl:pathType` and returns its value or an error. It MUST be a valid operation when a system is initialized or running. It SHOULD be valid in a failed or terminated system.

A non-normative example of a request is as follows:

```
<api:resolveRequest>/components/server/proxy/port</api:resolveRequest>
```

The response is arbitrary XML data, the contents of which depend upon what the path resolved to. For example, it could be an EPR to a resolved component, or it could be data in the response.

A non-normative example of a response is:

```
<api:resolveResponse>
  <wsa:Address>http://example.org/resources/2</wsa:Address>
</api:resolveResponse>
```

### 7.2.6 Terminate

This request terminates the system. This call MUST NOT raise a fault when the system is already terminated, or when termination is in progress.

A non-normative example of a termination request is:

```
    <api:terminateRequest>
      <api:reason>
        Scheduled weekly upgrade
      </api:reason>
    </api:terminateRequest>
```

Upon receipt, the system MUST be terminated. Termination is asynchronous. The response is an empty element, `<api:terminateResponse/>`.

### 7.2.7 Destroy

The WS-ResourceLifetime `<wsrf-rl:Destroy/>` operation destroys the System endpoint itself. All files uploaded are destroyed, and the system is terminated if it is not already terminated.

After sending this message and receiving a response, service consumers SHOULD NOT make calls of the endpoint, as it may be invalid.

### 7.2.8 GetResourceProperty/GetMultipleResourceProperties

These two operations are defined by the WS-ResourceProperties specification.

### 7.2.9 Subscribe/GetCurrentMessage

These two operations are defined by the WS-BaseNotification specification.

## 8 Notification

Notification enables front-end applications to receive notification when a system changes state. It also enables management tools to track the number of running systems.

All implementations of the deployment API MUST support the WS-BaseNotification notification mechanism. Implementations MAY implement additional mechanisms;

Every implementation is REQUIRED to enumerate all supported mechanisms in a list included in its static portal information property.

### 8.1 Notification Policy

- Implementations MUST support WS-Notification version 1.2-draft-01.
- Implementations MAY support alternate notification mechanisms.
- Implementations MUST list the URIs all supported notification mechanisms in the `StaticPortalStatus/notification` list.
- The URI of the supported version WS-Notification is `http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd`
- Implementations MUST support the list of basic topics defined for each EPR type.
- Implementations MAY support any other notification topics.
- Implementations MAY also support Terminate notification events of WS-ResourceLifetime, which are raised after an EPR is destroyed.
- There is no requirement for fault-tolerant subscriptions. Implementations MAY include policy metadata that informs callers how to renew subscriptions in the event of system failure.

### 8.2 WS-Notification Support

As stated above, implementations MUST support WS-Notification. There are specific topic spaces [WS-Topics] defined:

- Portal Endpoints MUST support a WS-TopicSpace that contains one topic: system creation events. This notifies callers that a new system has been created.
- System Endpoints MUST support the WS-TopicSpace and notifications defined in the Component Model specification. This includes a notification of changes in a system's lifecycle state.

### 8.3 Portal Notifications

The notification to be sent to listeners of a new System Endpoint MUST be a WSDM management event. This event must contain the an element of the following type, `SystemCreatedEventDataType`, under the path

```
muws-p1-xs:ManagementEvent/api:SystemCreatedEventData:
```

The declaration of the element and its type are described in the deployment API schema and are, no-normatively, as follows:

```
<xsd:complexType name="SystemCreatedEventDataType">
  </xsd:documentation>
  </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="muws-p1-xs:ResourceId"/>
      <xsd:element name="Reference" type="wsa:EndpointReferenceType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="SystemCreatedEventData"
type="api:SystemCreatedEventDataType"/>
```

The newly created system is identified by a unique WSDM `ResourceId`, and is addressable at the address supplied in the `Reference` element.

Implementations MAY extend the message with extra data.

A non-normative example of a system creation event is

```
<muws-p1-xs:ManagementEvent>
  <muws-p1-xs:EventId>http://example.org/events/1</muws-p1-xs:EventId>
  <muws-p1-xs:SourceComponent>
    <muws-p1-xs:ResourceId>http://example.org/resources/1</muws-p1-
xs:ResourceId>
  </muws-p1-xs:SourceComponent>
  <api:SystemCreatedEventData>
    <muws-p1-xs:ResourceId>http://example.org/resources/2</muws-p1-
xs:ResourceId>
    <api:Reference>
      <wsa:Address>http://example.org/resources/2</wsa:Address>
    </api:Reference>
  </api:SystemCreatedEventData>
</muws-p1-xs:ManagementEvent>
```

The normative declaration of the portal notification topic space is in Appendix A.

### *8.4 System Notifications*

System endpoints MUST support the lifecycle event notifications of the component model.. The normative declaration of the system notification topic space is in Appendix A.

### *8.5 Fault-Tolerant Notification*

Implementations are NOT REQUIRED to provide fault-tolerant notification. The failure of portal MAY result in the loss of portal event subscriptions, and the failure of a system MAY result in the loss of system event subscriptions.

## 9 Fault Policy

Faults are based upon the WS-BaseFault model [WS-BF], taking on some of the lessons of [Loughran02], namely that extra information such as hostname and process is essential for locating which process among many has failed on a clustered system.

Faults are raised in response to errors either at the remote endpoint, in the local framework, or between the remote endpoint and other parts of the distributed system. They can be returned to callers in response to an operation on an endpoint, or sent as part of a notification event.

All faults that will be explicitly sent are derived from WS-BaseFault faults. Service implementations MAY implicitly raise SOAPFault faults, as that is inherent in most implementations.

### *9.1 Fault Categories*

#### 9.1.1 Service Faults

These faults are raised by the service. They are grouped into a hierarchy of WS-BaseFault faults. There is a base fault class `DeploymentFault`, from which all others are derived.

All Service interfaces MUST declare that they raise these `DeploymentFault` instances, rather than list the specific faults. This is to provide forward extensibility.

The API lists specific extended faults of `DeploymentFault` that MAY be generated by a service or received by a client. These faults represent some of the faults that a service implementation MAY send.

If an implementation has a fault state whose meaning matches that of the predefined fault, the predefined fault MUST be thrown. If this predefined fault has standard elements for embedded fault information, they MUST be initialized with all the appropriate information, unless that information is unavailable. The implementation MAY add implementation-specific data within the `extra-data` element of the fault, to supplement this information. This extra data MUST NOT declare new types within the XML namespaces of the CDDLM specifications. The XML schema and semantics of this extra data SHOULD be documented.

If an existing fault type is not suitable, implementations MAY create new fault types. New fault types MUST extend the existing fault types which operations are declared as throwing. This effectively means that they MUST extend `DeploymentFault`. These new faults MUST NOT change the XML schemas of the deployment API, and they MUST be in a new namespace. The new faults and XML content SHOULD be publicly documented.

If an implementation adds new operations or properties at the existing endpoints, these new operations MAY raise whatever faults they see fit, within the constraints of the WS-BaseFault specification. Again, the implementation MUST NOT add new types to the deployment API namespace.

### 9.1.2 Transport faults

Transport faults will inevitably be raised as the appropriate fault for the system. For example, the Apache Axis SOAP client raises `AxisFault` faults for all SOAP events, wrapping stack trace and even HTTP Fault data within the fault as DOM elements. Microsoft .NET WSE has a similar fault class.

### 9.1.3 Relayed Faults

Relayed faults are those received by the far end and passed on. They may be WS-BaseFault Faults; HTTP error codes, SOAP faults, native language faults wrapped as SOAPFaults, or predefined deployment faults.

WS-BaseFault uses fault nesting for relaying faults; however, all faults MUST be a derivative of WS-BaseFault. This is addressed by defining a new WS-BaseFault derivative, a `WrappedSOAPFaultType`. This type is actually an extension of `CddlmFaultType`. This fault can nest any received SOAPFault, with an element containing the received XML data. Well-known elements in this fault data (such as the Apache Axis stack trace and HTTP fault code) SHOULD be copied into any fields in the main fault that fill the same role.

### 9.1.4 Fault Hierarchy

The UML representation of the fault hierarchy is shown in Figure 3 .

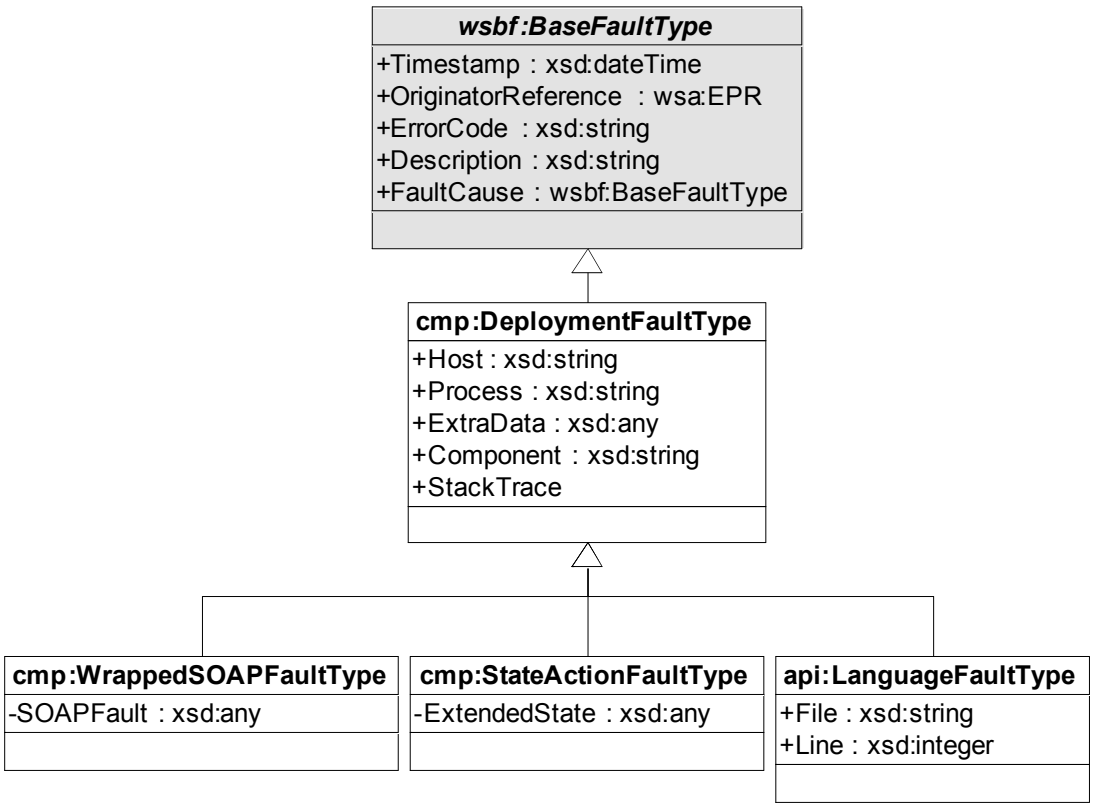


**Figure 3 . Hierarchy of XSD datatypes used to describe faults**

### *9.2 Fault Security*

Sites offering deployment services, MAY, for security reasons, wish to strip out some information, such as stack trace data. Implementations SHOULD provide a means to enable such an action prior to transmitting faults to callers.

Host name and process information may be viewed as sensitive, yet again, this is exceedingly useful to operations. Implementations MAY provide a means to disguise this information, so that it does not describe the real hostname or process ID of a process, but instead pseudonyms that can still be used in communications with any operations team.

### *9.3 Internationalization*

The WS-BaseFault specification makes no statement upon which language error descriptions are described.

If an implementation can return descriptions in one language, it MUST use `xml:lang` attributes to indicate the language of a description. Multiple descriptions, in different languages MAY be included. The client application SHOULD extract the description(s) whose language is the nearest match to that of the client.

### *9.4 Faults*

#### 9.4.1 cmp:DeploymentFaultType

This type represents any fault thrown by the deployment infrastructure. All endpoint operations MUST declare that they throw this fault, and MUST NOT explicitly declare any derivative faults that they might throw.

There is an element `api:DeploymentFault` that is of this type; it is this element that is used in defining the WSDL fault messages of all operations in the deployment API WSDL.

| *Element* | *Type* | *Meaning* |
|---|---|---|
| `Host` | `xsd:string` | Hostname or pseudonym |
| `Process` | `xsd:string` | Any process identifier suitable for diagnostics |
| `ExtraData` | `cmp:unboundedXMLAnyNamespace` | Extra fault data |
| `Component` | `xsd:string` | Path to component raising the fault |
| `Stack` | `stringListType` | Optional stack trace |

Implementations MUST include a component reference if it is known. Implementations SHOULD include hostname and process information. Process information MAY be a low-level identifier (such as an operating system process ID), or it MAY be some application specific identifier. Its role is merely to distinguish which process amongst many in a load-balanced implementation raised the fault.

### 9.4.2 api:LanguageFaultType

A language fault represents any fault in language processing for which a file and line number are relevant.

| *Element* | *Type* | *Meaning* |
|---|---|---|
| `File` | `xsd:string` | Filename/URI of file at fault |
| `Line` | `xsd:integer` | Line number within the file |

If the error is in the inline deployment descriptor, the `File` element MUST be empty, that is, "", or omitted. Furthermore, the `Line` element MUST be relative not to the deployment request, but to the inline descriptor. Recipients of faults can then infer from the empty/absent file element that the fault was in the inline request.

Note that a consequence of this design is that implementations SHOULD preserve white space in the deployment descriptor when saving them to file.

A non-normative example of a language fault is

```
<api:LanguageFault>
  <wsbf:Timestamp>2005-06-06T18:22:00Z</wsbf:Timestamp>
  <wsbf:Description>Missing closing bracket</wsbf:Description>
  <cmp:Host>34.example.org</cmp:Host>
  <cmp:Process>12</cmp:Process>
  <cmp:Stack>
    <cmp:item>org.example.cdl.Parser:456</cmp:item>
    <cmp:item>org.example.cdl.Dispatch:210</cmp:item>
    <cmp:item>org.example.cdl.PortalEndpoint:233</cmp:item>
    <cmp:item>org.example.cdl.Main:156</cmp:item>
    <cmp:item>org.alpine.dispatch:668</cmp:item>
    <cmp:item>org.alpine.http.servlet:1045</cmp:item>
    <cmp:item>org.apache.tomcat.servlet</cmp:item>
  </cmp:Stack>
  <api:File>http://files.example.org/cdl/parents/cdl1.cdl</api:File>
  <api:Line>45</api:Line>
</api:LanguageFault>
```

### 9.4.3 cmp:WrappedSOAPFaultType

This type represents a mapping of a classic W3C `SOAPFault` [SOAP1.2] to a WS-BaseFault, as an extension of `DeploymentFault`. It caches data unique to SOAP Faults.

| *Element* | *Type* | *Meaning* |
|---|---|---|
| `SoapFault` | `s12:Fault` | Fault code information |

The normative mapping of `SOAPFault` elements to `WrappedSOAPFault` elements is as follows:

| *SOAP1.2* | *WrappedSOAPFault* |
|---|---|
| `/s12:Fault` | `WrappedSOAPFault/api:SoapFault` |
| `SOAP endpoint` | `WrappedSOAPFault/wsrf-bf:originator` |

The SOAP endpoint MUST be translated into a `wsa:EndpointReference` if it is a simple URL/SOAPAction tuple.

Detail from SOAP stacks with well-known fault fields, such as the Apache Axis stack trace, MAY be imported into appropriate fields in the `DeploymentFault`.

## *9.5 Fault Error Codes*

Specific fault error codes, and their meaning, will be covered in a separate informative document.

# 10 Example Interactions

## *10.1 A simple deployment*

Figure 4 is a UML sequence diagram of an interaction with the deployment infrastructure. A client application connects a portal that it has knowledge of, and creates a system resource. It can then add files to the system, before moving the system into the functional state.

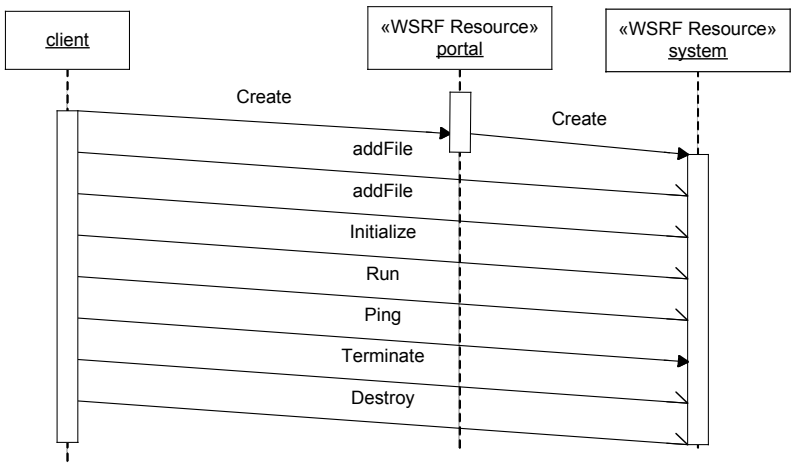


**Figure 4 . Sequence diagram of a simple deployment**

During the life of the system, it will respond to `Ping` and `Resolve` operations. Eventually it is terminated, and then finally the resource itself is destroyed.

### 10.2 Subscribing to events from an existing system

Figure 5 shows a client connecting to a portal, and making a `LookupSystem` call to get the EPR of a system.

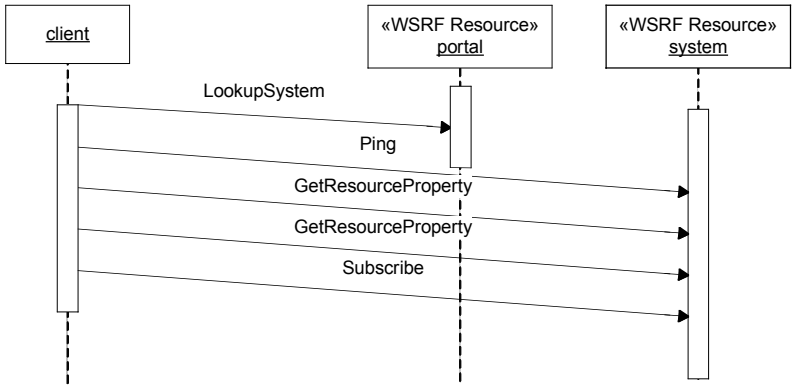


**Figure 5 . Subscription sequence diagram**

The client issues a `Ping` call to verify the health of the remote system, then two `GetResourceProperty` messages to access resource properties, first its state (`api:SystemState`) and then its supported message topics (`wsnt:Topic`). Finally, it sends a `Subscribe` request to subscribe to the lifecycle event topic of the system.

Later, when lifecycle events take place, the system will send notifications to the EPR provided in the subscription message.

## 11 Implementation Requirements

Implementations MAY validate incoming requests against the XML Schema used to describe this service. If this is not done, the implementation SHOULD validate messages by other means. It is an error if a required element is not included in a request, or it occurs more often than the schema permitted.

Instances of system and portal endpoints MUST be re-entrant.

Implementations MAY provide the XSD/WSDL of the endpoints using the de-facto standard of `GET endpoint+"?wsdl"`, or by using some other mechanism.

The generation and processing of SOAP messages and HTTP error codes (if using SOAP over HTTP), MUST be in accordance with the WS-I Basic Profile 1.1 specification.

There are number of places in the specification in which the contents of remote URLs are to be retrieved. This retrieval MUST also be re-entrant, that is, any caching mechanism MUST be thread-safe. Furthermore, all such requests SHOULD use HTTP/1.1, implement a time-out mechanism on downloads, verify the length of retrieved data against the content-length header (which is REQUIRED by HTTP1.1), and fail if an error occurs.

## 12 Security Considerations

The deployment requests MUST only be granted by suitably authorized individuals, or their suitably authorized agents. For deployment to a Grid infrastructure, that means that the standardized security model of the infrastructure SHOULD be used to

authenticate callers. Only callers with the relevant rights MUST be allowed to deploy systems.

When delegating deployments across nodes, the node issuing the deployments needs to have the rights to do so, and the deployment itself still needs to be authenticated as a legitimate request of the sender.

Along with deployment, the ability of a caller to list and manipulate running systems, introduces another security issue: that of who has access to the set of deployed systems.

Files uploaded via `addFile` SHOULD be secured against unauthorized access. There SHOULD also be a limit to the total size of files uploaded by a single user, if a denial of service attack on the file system is to be prevented. Quota-enabled filesystems are one possible solution.

There are a number of places in the system in which remote URLs to data may be supplied, as an alternative to sending the information inline. In these situations, the service implementation MUST NOT retrieve this content with greater rights than that of the caller. Furthermore, to ensure that the content is that which the caller has chosen to publish, the HTTPS/TLS protocols SHOULD be preferred over HTTP, unless the downloaded content is itself authenticated by some form of signing mechanism.

# 13 Editor Information

Steve Loughran, HP Laboratories

steve_loughran@hpl.hp.com

# 14 References

## *14.1 Normative References*

[MOWS] Sedukhin I. et al, *Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0,* OASIS, 2004.

[MTOM] Gudgin, M., *SOAP Message Transmission Optimization Mechanism*, W3C, 2005.

[RFC2119] S. Bradner, *RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels*, Internet Engineering Task Force, 1997.

[CDDLM-CMP] Schaeffer S., *CDDLM Component Model Specification*, 2005

[SOAP1.2] Gudgin, M. et al, *SOAP Version 1.2*, W3C, 2003.

[JSDL] Savva A., et al., *Job Submission Description Language (JSDL) Specification, Version 1.0*, GGF, 2005.

[XML-CDL] Tatemura, J. *CDDLM XML Configuration Description Language Specification version 1.0*, GGF, 2005.

[WS-A] Gudgin, M. and Hadley S., *Web Services Addressing 1.0 - Core*, W3C, 2005.

[WS-BF] Tuecke et al., *Web Services Base Faults (WS-BaseFaults)*, OASIS, 2004.

[WS-BaseNotification] Graham S. et al., *Web Services Base Notification 1.0 (WS-BaseNotification)*, OASIS, 2004.

[WS-BrokeredNotification] Graham S. et al., *Web Services Brokered Notification 1.0 (WS-BrokeredNotification)*, OASIS, 2004.

[WS-ResourceLifetime] Frey J. et al., *Web Services ResourceLifetime 1.1 (WS-ResourceLifetime)*, OASIS, 2004.

[WSRF] Tuecke et al., *Web Services Resource Framework (WSRF)*, 2004.

[WS-ResourceProperties] Graham S. et al., *Web Services Resource Properties 1.1 (WS-ResourceProperties)*, OASIS, 2004.
[WS-ServiceGroups] Graham S. et al., *Web Services Service Group Specfication 1.0 (WS-ServiceGroups)*, OASIS, 2004.
[WS-Topics] Graham S. et al., *Web Services Topics (WS-Topics)* , OASIS, 2004.

### *14.2 Informative References*

[Loughran02] Loughran, Making Web Services that Work, HP Laboratories, TR-HPL-2002-274, 2002.

## Appendix A: Event Topics

The normative listing of the Portal event topics is

```
<wstop:TopicSpace name="PortalNotificationTopics"
  targetNamespace=
  "http://www.gridforum.org/cddlm/deployapi/2005/02/events/portal"
  xmlns:api="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns:muws-p1-xs="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part1.xsd"
  xmlns:wstop="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-
draft-01.xsd"
  >
  <wstop:Topic name="SystemCreatedEvent"
    messageTypes="muws-p1-xs:ManagementEvent">
  </wstop:Topic>
</wstop:TopicSpace>
```

The normative listing of the System event topics is

```
<wstop:TopicSpace name="SystemNotificationTopics"
  targetNamespace="
  http://www.gridforum.org/cddlm/deployapi/2005/02/events/system"
  xmlns:muws-p1-xs="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part1.xsd"
  xmlns:wstop="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-
draft-01.xsd"
  >
  <wstop:Topic name="LifecycleEvent"
    messageTypes="muws-p1-xs:ManagementEvent"/>
</wstop:TopicSpace>
```

## Appendix B: Language URIs

The current set of well-known Language URIs are as follows:

| | |
|---|---|
| http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0 | CDL |
| http://www.gridforum.org/namespaces/2005/02/cddlm/smartfrog | SmartFrog |

## Appendix C: Deployment API XML Schema

This section contains the full normative XML Schema definition for CDDLM deployment API 1.0.

If, however, there are any discrepancies between the definitions in this section and those portions described in other sections in the specification then the definitions in this section MUST be considered normative.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

Copyright (C) Global Grid Forum (2004-2006). All Rights Reserved.

-->
<xsd:schema targetNamespace="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns:api="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
  xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-
01.xsd"
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.xsd"
  xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.xsd"
  xmlns:wsnt="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.xsd"
  xmlns:wstop="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd"
  xmlns:muws-p1-xs="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd"
  xmlns:muws-p2-xs="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2.xsd"
  elementFormDefault="qualified">
  <!-- ============================================================= -->
  <xsd:annotation>
    <xsd:documentation>

    This is the XSD describing the types of the CDDLM service API


    Version: 2005-05-13

    This draft uses WS-RF as the underpinnings of interaction

    The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
    NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED",  "MAY", and
    "OPTIONAL" in this document are to be interpreted as described in
    RFC 2119.
    http://www.ietf.org/rfc/rfc2119.txt

    </xsd:documentation>
  </xsd:annotation>
  <!-- ============================================================= -->
  <!-- BEGIN IMPORTS -->
  <!-- ============================================================= -->

  <xsd:import namespace="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0"
    schemaLocation="xml-cdl.xsd"/>
  <xsd:import namespace="http://www.gridforum.org/cddlm/components/2005/02"
    schemaLocation="component-model.xsd"/>
  <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
    schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>
  <xsd:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
    schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/XML/1998/namespace"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
```

```
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-
1.2-draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-
draft-01.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-
1.2-draft-01.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-
draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-
01.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part1.xsd"
    schemaLocation="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part1.xsd"/>
  <xsd:import namespace="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part2.xsd"
    schemaLocation="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part2.xsd"/>



  <!-- ============================================================= -->
  <!-- END IMPORTS -->
  <!-- ============================================================= -->
  <!-- low level type: XML with no validations, namespace = ##other -->
  <!-- ============================================================= -->
  <!-- naming rules for a system -->
  <!-- ============================================================= -->
  <xsd:simpleType name="systemNameType">
    <xsd:annotation>
      <xsd:documentation>
        This is the policy for the naming of systems
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:NCName">
      <xsd:pattern value="[a-zA-Z_\-\.][a-zA-Z_\-\.\P{Nd}]*"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- ============================================================= -->
  <!-- allows the caller to pass a URL to a deploy descriptor, instead
     of the body itself. This allows signed descriptors inside
     files to be used as a source of data -->
  <!-- ============================================================= -->
  <xsd:simpleType name="remoteDescriptorType">
    <xsd:annotation>
      <xsd:documentation>
        Descriptors can also be URLs.
        We extend the URI type in case of future needs to add attributes such as
        authentication information.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:anyURI"/>
  </xsd:simpleType>
  <!-- ============================================================= -->
  <!-- BEGIN COMPLEXTYPES -->
  <!-- ============================================================= -->
  <xsd:complexType name="descriptorType">
    <xsd:annotation>
      <xsd:documentation>
      A descriptor is either arbitrary XML, or a URL to arbitrary XML,
      And a URI identifying the language/version
      </xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
      <xsd:element name="reference" type="xsd:anyURI">
        <xsd:annotation>
          <xsd:documentation>
            A remote reference to a descriptor.
          </xsd:documentation>
```

```
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="body">
      <xsd:annotation>
        <xsd:documentation>
          The deployment descriptor, inline
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:any namespace="##other" processContents="lax"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
  <xsd:attribute name="language" type="xsd:anyURI" use="required">
    <xsd:annotation>
      <xsd:documentation>
      The URI identifying the language of the descriptor.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>
<!-- ============================================================ -->
<!-- options -->
<!-- ============================================================ -->
<xsd:complexType name="optionType">
  <xsd:annotation>
    <xsd:documentation>
      Options are @uri plus (string|integer|boolean|XML)
      really, all could be XML, but that forces too much parse complexity
      having string and integer attributes makes it simpler.

      A processor of the options MUST ignore options that it does not recognise
      and that are marked mustUnderstand==false.

      It MUST fail if it receives an option marked mustUnderstand==true that it
      does not recognise or can not process in accordance with the
      normative/descriptive definition of the option.

      It is also illegal to have more than one of the option types defined.
      If the SOAP runtime does not validate the choice statement, callers
      must instead.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:element name="string" type="xsd:string" xml:space="preserve"/>
    <xsd:element name="integer" type="xsd:integer"/>
    <xsd:element name="boolean" type="xsd:boolean"/>
    <xsd:element name="propertyMapOption" type="api:propertyMapType"/>
    <xsd:element name="data" type="cmp:unboundedXMLAnyNamespace"/>
  </xsd:choice>
  <xsd:attribute name="name" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="mustUnderstand" type="xsd:boolean" use="optional"
    default="false"/>
</xsd:complexType>
<!-- ============================================================ -->
<!-- a set of options -->
<!-- ============================================================ -->
<xsd:complexType name="optionMapType">
  <xsd:annotation>
    <xsd:documentation>
      A set of options. Only one of each URI is allowed.
      An option map with duplicate URIs MUST BE rejected.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="option" type="api:optionType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- property tuples-->
<!-- ============================================================ -->
<xsd:complexType name="propertyTupleType">
  <xsd:annotation>
    <xsd:documentation>
```

```
      A name/value pair of properties. Names are restricted to systemNameType;
      values are any string. Why restrict name type? For easier incorporation
      into languages.

    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="name" type="api:systemNameType"/>
    <xsd:element name="value" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- a map of property tuples -->
<!-- to use this in a deployment, add it under the option -->
<!-- http://gridforum.org/cddlm/serviceAPI/options/properties/2004/07/30 -->
<!-- ============================================================ -->
<xsd:complexType name="propertyMapType">
  <xsd:annotation>
    <xsd:documentation>
      A set of name/value pair of properties.
     There is no explicit requirement for (name,value) tuples to be unique.
     Users of the propertyMapType may impose this restriction
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="property" type="api:propertyTupleType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- this section includes various lists of server information -->
<!-- which are returned in the server status response -->
<!-- ============================================================ -->
<!-- ============================================================ -->
<!-- list of [(name,uri)] -->
<!-- ============================================================ -->
<xsd:complexType name="nameUriListType">
  <xsd:annotation>
    <xsd:documentation>
      This is a list of (name,uri) values.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:annotation>
          <xsd:documentation>
            A language is defined by a name and a namespace.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="uri" type="xsd:anyURI"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- list of [uri] -->
<!-- ============================================================ -->
<xsd:complexType name="uriListType">
  <xsd:annotation>
    <xsd:documentation>
      This is a type representing a list of URIs
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="item" type="xsd:anyURI" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- Portal status -->
<!-- this section includes various lists of server information -->
<!-- which are returned in the server status response -->
<!-- ============================================================ -->
```

```
<!-- static server info -->
<xsd:complexType name="portalInformationType">
  <xsd:annotation>
    <xsd:documentation>
      This is static information about the server itself.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string">
      <xsd:annotation>
        <xsd:documentation>
        Name of the implementation of the server
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="build" type="xsd:string" xml:space="preserve" minOccurs="0" >
      <xsd:annotation>
        <xsd:documentation>
        Human readable build information: version, build number, build date,
        etc.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="home" type="xsd:anyURI"  minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
      URL to a page providing support and docs for the server
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="location" type="xsd:string"  minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
        Location string describing server location
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="timezoneUTCOffset" type="xsd:integer"  minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
          Offset from UTC of the local clock.
          Provides a location hint and good diagnostics
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="diagnostics" type="cmp:unboundedXMLAnyNamespace"
      minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
          Arbitrary static diagnostic information.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!-- portal static information -->
<xsd:complexType name="staticPortalStatusType">
  <xsd:annotation>
    <xsd:documentation>
      This Static information MAY be expected to be constant for the life of
      this instance of the service. This does not mean MUST; a dynamic server
      MAY change attributes of its service dynamically.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="portal" type="api:portalInformationType">
      <xsd:annotation>
        <xsd:documentation>Portal Details
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="languages" type="api:nameUriListType">
      <xsd:annotation>
        <xsd:documentation>
          List of languages
        </xsd:documentation>
      </xsd:annotation>
```

```
    </xsd:element>
    <xsd:element name="joblanguages" type="api:nameUriListType">
      <xsd:annotation>
        <xsd:documentation>
          List of supported Job languages; each by their own URI.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="notifications" type="api:uriListType">
      <xsd:annotation>
        <xsd:documentation>
          List of supported notification mechanisms
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="options" type="api:uriListType">
      <xsd:annotation>
        <xsd:documentation>
          List of options that are understood
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- list of system is used for enumerating running systems -->
<!-- ============================================================ -->
<xsd:complexType name="systemReferenceListType">
  <xsd:annotation>
    <xsd:documentation>
      This is a (potentially empty) list of  systems
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="system" type="wsa:EndpointReferenceType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ============================================================ -->
<!-- empty elements get used in a few places -->
<!-- ============================================================ -->
<xsd:complexType name="emptyElementType">
  <xsd:annotation>
    <xsd:documentation>
      This is an empty message
    </xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
<!-- ============================================================ -->
<!-- option elements -->
<!-- ============================================================ -->
<xsd:element name="propertyMapOption" type="propertyMapType">
  <xsd:annotation>
    <xsd:appinfo>
      http://gridforum.org/cddlm/serviceAPI/options/propertyMap/2004/07/30
    </xsd:appinfo>
    <xsd:documentation>
      this is the contents of a property map option
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
<!-- ============================================================ -->
<!-- BEGIN PORTAL EPR PROPERTIES AND TYPES -->
<!-- ============================================================ -->
<!-- ============================================================ -->
<!-- declare the WS-ResourceProperties properties for the portal -->
<!-- ============================================================ -->
<xsd:element name="StaticPortalStatus" type="api:staticPortalStatusType"/>

<xsd:element name="ActiveSystems" type="api:systemReferenceListType"/>

<!--Resource property document declaration-->
<!-- you MUST use namespace prefixes here, for Apache Muse to parse it -->
<xsd:element name="portalEndpointProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="muws-p1-xs:ResourceId"/>
```

```
        <xsd:element ref="muws-p1-xs:ManageabilityCapability"/>
        <xsd:element ref="api:StaticPortalStatus"/>
        <xsd:element ref="api:ActiveSystems"/>
        <xsd:element ref="wsnt:Topic" maxOccurs="unbounded"/>
        <xsd:element ref="wsnt:FixedTopicSet"/>
        <xsd:element ref="wsnt:TopicExpressionDialects" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- Requests and responses. These are the complex types that
get sent/received as messages-->
  <!-- ============================================================ -->
  <xsd:element name="createRequest">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>
          This is our request to create a system
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="hostname" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- ============================================================ -->
  <xsd:element name="createResponse">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>
          This is our deployment response.
          It consists of a reference to the system,
          which can then be used directly
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="ResourceId" type="xsd:anyURI"/>
        <xsd:element name="systemReference" type="wsa:EndpointReferenceType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- look up a system  -->
  <!-- ============================================================ -->
  <xsd:element name="lookupSystemRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ResourceId" type="xsd:anyURI">
          <xsd:annotation>
            <xsd:documentation>Application to resolve</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- the endpoint that we resolved  -->
  <!-- ============================================================ -->
  <xsd:element name="lookupSystemResponse" type="wsa:EndpointReferenceType"/>
  <!-- ============================================================ -->
  <!-- resolve something relative to a system  -->
  <!-- ============================================================ -->
  <xsd:element name="portalResolveRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ResourceId" type="xsd:anyURI">
          <xsd:annotation>
            <xsd:documentation>Application to resolve</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="path" type="cdl:pathType">
          <xsd:annotation>
            <xsd:documentation>Path to resolve</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
```

```
  </xsd:complexType>
</xsd:element>
<!-- ================================================================= -->
<!-- contents of the resolution -->
<!-- ================================================================= -->
<xsd:element name="portalResolveResponse" type="cmp:unboundedXMLAnyNamespace"/>

<!-- ================================================================= -->
<!-- BEGIN SYSTEM EPR PROPERTIES AND TYPES -->
<!-- ================================================================= -->
<!-- ================================================================= -->
<!-- declare the WS-ResourceProperties properties for the systems -->
<!-- ================================================================= -->
<xsd:element name="SystemState" type="cmp:componentStatusType"/>
<!-- time the system was created -->
<xsd:element name="CreatedTime" type="xsd:dateTime"/>
<!-- time the system was started -->
<xsd:element name="StartedTime" type="xsd:dateTime"/>
<!-- time the system was terminated -->
<xsd:element name="TerminatedTime" type="xsd:dateTime"/>
<xsd:element name="TerminationRecord" type="cmp:terminationRecordType"/>
<!--Resource property document declaration-->
<!-- you MUST use namespace prefixes here, for Apache Muse to parse it -->
<xsd:element name="SystemEndpointProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="muws-p1-xs:ResourceId"/>
      <xsd:element ref="muws-p1-xs:ManageabilityCapability"/>
      <xsd:element ref="api:CreatedTime"/>
      <xsd:element ref="api:StartedTime" minOccurs="0"/>
      <xsd:element ref="api:TerminatedTime" minOccurs="0"/>
      <xsd:element ref="api:SystemState"/>
      <xsd:element ref="api:TerminationRecord" minOccurs="0"/>
      <xsd:element ref="wsnt:Topic" maxOccurs="unbounded"/>
      <xsd:element ref="wsnt:FixedTopicSet"/>
      <xsd:element ref="wsnt:TopicExpressionDialects" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!-- ================================================================= -->
<!-- <init/> request -->
<!-- ================================================================= -->
<xsd:element name="initializeRequest">
  <xsd:complexType>
    <xsd:annotation>
      <xsd:documentation>
        Move a created system to the initialized state.
        This will trigger whatever validation/parsing of the
        descriptor is desired, and may result in the creation and
        initialisation of components in the system.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="job" type="api:descriptorType" minOccurs="0">
        <xsd:annotation>
          <xsd:documentation/>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="descriptor" type="api:descriptorType">
        <xsd:annotation>
          <xsd:documentation>
          deployment descriptor
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="options" type="api:optionMapType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!-- ================================================================= -->
<!-- this response means that the request has been submitted, -->
<!-- some initial validation may take place (and so be failed immediately) -->
<!-- but the core deployment operation is asynchronous -->
<!-- ================================================================= -->
<xsd:element name="initializeResponse" type="cmp:void"/>
<!-- ================================================================= -->
<!-- request to add a file-->
```

```
  <!-- ============================================================ -->
  <xsd:element name="addFileRequest">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>
          Attach a file to the application.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:anyURI">
          <xsd:annotation>
            <xsd:documentation/>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="mimetype" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>Mime type of the data</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="scheme" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>scheme of the returned URI</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:choice>
          <xsd:element name="data" type="xsd:base64Binary">
            <xsd:annotation>
              <xsd:documentation>Data. </xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="uri" type="xsd:anyURI">
            <xsd:annotation>
              <xsd:documentation>URI of attachment or URL to
source</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
        </xsd:choice>
        <xsd:element name="metadata" type="cmp:unboundedXMLAnyNamespace" minOccurs="0"
>
          <xsd:annotation>
            <xsd:documentation>optional metadata for any implementations to
use.</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- return  A URL of the file uploaded -->
  <!-- ============================================================ -->
  <xsd:element name="addFileResponse" type="api:uriListType">
    <xsd:annotation>
      <xsd:documentation>list of URLs to of the uploaded file</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- request to run the engine -->
  <!-- ============================================================ -->
  <xsd:element name="runRequest" type="xsd:string">
    <xsd:annotation>
      <xsd:documentation>
Request to run a system.
Valid only if the system is in a state from which
it can enter the running state, or it is already running.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- this response means that the request has been submitted, -->
  <!-- some initial validation may take place (and so be failed immediately) -->
  <!-- but the core deployment operation is asynchronous -->
  <!-- ============================================================ -->
  <xsd:element name="runResponse" type="cmp:void"/>
  <!-- ============================================================ -->
  <!-- system::TerminateRequest -->
  <!-- ============================================================ -->
  <xsd:element name="terminateRequest">
```

```
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>
Request to terminate a system.
It is not a fault to terminate a system in the terminated
state, but it is a fault (no-such-system) to terminate an
unknown system.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="reason" type="xsd:string" minOccurs="0" >
          <xsd:annotation>
            <xsd:documentation>
Reason for termination. This message may be included in the system logs or otherwise
recorded.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- terminate response -->
  <!-- ============================================================ -->
  <xsd:element name="terminateResponse" type="cmp:void"/>
  <!-- ============================================================ -->
  <!-- resolve something relative to a system  -->
  <!-- ============================================================ -->
  <xsd:element name="resolveRequest" type="cdl:pathType">
    <xsd:annotation>
      <xsd:documentation>Path to resolve</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <!-- ============================================================ -->
  <!-- contents of the resolution -->
  <!-- ============================================================ -->
  <xsd:element name="resolveResponse" type="cmp:unboundedXMLAnyNamespace"/>
  <!-- ============================================================ -->
  <!-- probe a system  -->
  <!-- ============================================================ -->
  <xsd:element name="pingRequest" type="cmp:void"/>
  <!-- ============================================================ -->
  <!--  response from a ping is explicitly a status -->
  <!-- implicitly, any response is a sign of health. -->
  <!-- ============================================================ -->
  <xsd:element name="pingResponse" type="cmp:componentStatusType"/>
  <!-- ============================================================ -->
  <!-- FAULTS -->
  <!-- ============================================================ -->

  <xsd:element name="DeploymentFault" type="cmp:DeploymentFaultType"/>

  <!-- Language Fault -->
  <xsd:complexType name="LanguageFaultType">
    <xsd:annotation>
      <xsd:documentation>Language Fault</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="cmp:DeploymentFaultType">
        <xsd:sequence>
          <xsd:element name="File" type="xsd:string" minOccurs="0"/>
          <xsd:element name="Line" type="xsd:integer" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="LanguageFault" type="api:LanguageFaultType"/>

  <!-- ============================================================ -->
  <!-- This is the data that is supplied when a system is created    -->
  <!-- It is included in the payload of a WSDM event -->
  <!-- ============================================================ -->

  <xsd:complexType name="SystemCreatedEventDataType">
    <xsd:annotation>
      <xsd:documentation>
Information about a new System Endpoint that has been created.
```

```
The message includes the WSRF resource identifier, and a reference
to the new endpoint itself.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="muws-p1-xs:ResourceId"/>
      <xsd:element name="Reference" type="wsa:EndpointReferenceType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="SystemCreatedEventData" type="api:SystemCreatedEventDataType"/>

  <!-- ============================================================ -->
  <!-- end schema-->
  <!-- ============================================================ -->
</xsd:schema>
```

## Appendix D. Deployment API WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://www.gridforum.org/cddlm/deployapi/2005/02/wsdl"
  xmlns:tns="http://www.gridforum.org/cddlm/deployapi/2005/02/wsdl"
  xmlns:api="http://www.gridforum.org/cddlm/deployapi/2005/02"

  xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-
01.xsd"
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.xsd"
  xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.wsdl"
  xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.xsd"
  xmlns:wsrf-rlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.wsdl"
  xmlns:wsnt="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.xsd"
  xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.wsdl"
  xmlns:muws-p1-xs="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd"
  xmlns:mows-wsdl="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.wsdl"

  >
  <!-- ================================================================= -->
  <wsdl:documentation>

    This is the WSDL Describing the service API for the public deployment
    services of a CDDLM Basic Services runtime.

    It binds the XSD types described in the deployment API types document
    to a service endpoint.

    2005-02-23 WSDL for a WS-RF endpoint

    The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
    NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED",  "MAY", and
    "OPTIONAL" in this document are to be interpreted as described in
    RFC 2119.
    http://www.ietf.org/rfc/rfc2119.txt

  </wsdl:documentation>
  <!-- ================================================================= -->
  <!-- BEGIN IMPORTS -->
  <!-- ================================================================= -->


  <wsdl:import namespace="http://www.gridforum.org/cddlm/components/2005/02"
    location="component-model.xsd"/>

  <wsdl:import
    namespace="http://www.gridforum.org/cddlm/deployapi/2005/02"
    location="deployment-api.xsd"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.xsd"
    location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.xsd"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl"
    location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl"/>


  <wsdl:import
```

```
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.xsd"
    location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.xsd"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.wsdl"
    location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.wsdl"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.wsdl"
    location="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.wsdl"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.xsd"
    location="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.xsd"/>

  <wsdl:import
    namespace="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.wsdl"
    location="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.wsdl"/>




  <!-- ================================================================= -->
  <!-- END IMPORTS -->
  <!-- ================================================================= -->

  <!-- ================================================================= -->
  <!-- BEGIN TYPES -->
  <!-- ================================================================= -->


  <wsdl:types>
      <xsd:schema elementFormDefault="qualified"
          targetNamespace="http://www.gridforum.org/cddlm/deployapi/2005/02/wsdl">


  <xsd:import
    namespace="http://www.gridforum.org/cddlm/deployapi/2005/02"
    schemaLocation="deployment-api.xsd"/>
  <xsd:import
    namespace="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0"
    schemaLocation="xml-cdl.xsd"/>
  <xsd:import
    namespace="http://www.gridforum.org/cddlm/components/2005/02"
    schemaLocation="component-model.xsd"/>

  <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
    schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>
  <xsd:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
    schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/XML/1998/namespace"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-
draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-
01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-
draft-01.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.xsd"
```

```
    schemaLocation="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-
1.2-draft-01.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd"
    schemaLocation="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-
01.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd"
    schemaLocation="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part1.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2.xsd"
    schemaLocation="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-
part2.xsd"/>



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

  <!-- ================================================================= -->
  <!-- END TYPES -->
  <!-- ================================================================= -->

  <!-- ================================================================= -->
  <!-- begin WSDL messages-->
  <!-- ================================================================= -->

  <!-- ================================== -->
  <!-- For the Portal EPR -->
  <!-- ================================== -->
  <wsdl:message name="createRequest">
    <wsdl:part element="api:createRequest" name="createRequest"/>
  </wsdl:message>

  <wsdl:message name="createResponse">
    <wsdl:part element="api:createResponse" name="createResponse"/>
  </wsdl:message>

  <wsdl:message name="lookupSystemRequest">
    <wsdl:part element="api:lookupSystemRequest" name="lookupSystemRequest"/>
  </wsdl:message>
  <wsdl:message name="lookupSystemResponse">
    <wsdl:part element="api:lookupSystemResponse" name="lookupSystemResponse"/>
  </wsdl:message>

  <!-- ================================== -->
  <!-- For the System EPR -->
  <!-- ================================== -->


  <wsdl:message name="initializeRequest">
    <wsdl:part element="api:initializeRequest" name="initializeRequest"/>
  </wsdl:message>
  <wsdl:message name="initializeResponse">
    <wsdl:part element="api:initializeResponse" name="initializeResponse"/>
  </wsdl:message>

  <wsdl:message name="addFileRequest">
    <wsdl:part element="api:addFileRequest" name="addFileRequest"/>
  </wsdl:message>
  <wsdl:message name="addFileResponse">
    <wsdl:part element="api:addFileResponse" name="addFileResponse"/>
  </wsdl:message>

  <wsdl:message name="runRequest">
    <wsdl:part element="api:runRequest" name="runRequest"/>
  </wsdl:message>
  <wsdl:message name="runResponse">
    <wsdl:part element="api:runResponse" name="runResponse"/>
  </wsdl:message>

  <wsdl:message name="terminateRequest">
    <wsdl:part element="api:terminateRequest" name="terminateRequest"/>
  </wsdl:message>
  <wsdl:message name="terminateResponse">
    <wsdl:part element="api:terminateResponse" name="terminateResponse"/>
```

```
</wsdl:message>

<wsdl:message name="resolveRequest">
  <wsdl:part element="api:resolveRequest" name="resolveRequest"/>
</wsdl:message>
<wsdl:message name="resolveResponse">
  <wsdl:part element="api:resolveResponse" name="resolveResponse"/>
</wsdl:message>

<wsdl:message name="portalResolveRequest">
  <wsdl:part element="api:portalResolveRequest" name="portalResolveRequest"/>
</wsdl:message>
<wsdl:message name="portalResolveResponse">
  <wsdl:part element="api:portalResolveResponse"
    name="portalResolveResponse"/>
</wsdl:message>

<wsdl:message name="pingRequest">
  <wsdl:part element="api:pingRequest" name="pingRequest"/>
</wsdl:message>

<wsdl:message name="pingResponse">
  <wsdl:part element="api:pingResponse" name="pingResponse"/>
</wsdl:message>
<!-- ============================================================ -->
<!-- Fault messages -->
<!-- ============================================================ -->


<!-- This is from the API schema; it is of type cmp:DeploymentFaultType -->
<wsdl:message name="DeploymentFault">
  <wsdl:part name="fault" element="api:DeploymentFault"/>
</wsdl:message>

<!-- ============================================================ -->
<!-- begin WSDL operations -->
<!-- ============================================================ -->

<!-- ============================================================ -->
<!-- Portal EPR -->
<!-- ============================================================ -->


<wsdl:portType name="PortalEPR"
  wsrf-rp:ResourceProperties="api:portalEndpointProperties">

  <!-- create -->
  <wsdl:operation name="Create">
    <wsdl:input message="tns:createRequest" name="CreateRequest"/>
    <wsdl:output message="tns:createResponse" name="CreateResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>

  <!-- resolve a path -->
  <wsdl:operation name="Resolve">
    <wsdl:input message="tns:portalResolveRequest"
      name="PortalResolveRequest"/>
    <wsdl:output message="tns:portalResolveResponse"
      name="PortalResolveResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>


  <!-- look up an app -->
  <wsdl:operation name="LookupSystem">
    <wsdl:input message="tns:lookupSystemRequest" name="LookupSystemRequest"/>
    <wsdl:output message="tns:lookupSystemResponse"
      name="LookupSystemResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>


  <!-- ========== WSRF-RP Specific ============== -->
  <wsdl:operation name="GetResourceProperty">
```

```
      <wsdl:input name="GetResourcePropertyRequest"
        message="wsrf-rpw:GetResourcePropertyRequest"/>
      <wsdl:output name="GetResourcePropertyResponse"
        message="wsrf-rpw:GetResourcePropertyResponse"/>
      <wsdl:fault name="ResourceUnknownFault"
        message="wsrf-rpw:ResourceUnknownFault"/>
      <wsdl:fault name="InvalidResourcePropertyQNameFault"
        message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
    </wsdl:operation>

    <wsdl:operation name="GetMultipleResourceProperties">
      <wsdl:input name="GetMultipleResourcePropertiesRequest"
        message="wsrf-rpw:GetMultipleResourcePropertiesRequest"/>
      <wsdl:output name="GetMultipleResourcePropertiesResponse"
        message="wsrf-rpw:GetMultipleResourcePropertiesResponse"/>
      <wsdl:fault name="ResourceUnknownFault"
        message="wsrf-rpw:ResourceUnknownFault"/>
      <wsdl:fault name="InvalidResourcePropertyQNameFault"
        message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
    </wsdl:operation>

    <!-- =========== NotificationProducer Specific ============== -->
    <wsdl:operation name="Subscribe">
      <wsdl:input name="SubscribeRequest"
        message="wsntw:SubscribeRequest"/>
      <wsdl:output name="SubscribeResponse"
        message="wsntw:SubscribeResponse"/>
      <wsdl:fault name="ResourceUnknownFault"
        message="wsntw:ResourceUnknownFault"/>
      <wsdl:fault name="SubscribeCreationFailedFault"
        message="wsntw:SubscribeCreationFailedFault"/>
      <wsdl:fault name="TopicPathDialectUnknownFault"
        message="wsntw:TopicPathDialectUnknownFault"/>
    </wsdl:operation>

    <wsdl:operation name="GetCurrentMessage">
      <wsdl:input name="GetCurrentMessageRequest"
        message="wsntw:GetCurrentMessageRequest"/>
      <wsdl:output name="GetCurrentMessageResponse"
        message="wsntw:GetCurrentMessageResponse"/>
      <wsdl:fault name="ResourceUnknownFault"
        message="wsntw:ResourceUnknownFault"/>
      <wsdl:fault name="InvalidTopicExpressionFault"
        message="wsntw:InvalidTopicExpressionFault"/>
      <wsdl:fault name="TopicNotSupportedFault"
        message="wsntw:TopicNotSupportedFault"/>
      <wsdl:fault name="NoCurrentMessageOnTopicFault"
        message="wsntw:NoCurrentMessageOnTopicFault"/>
    </wsdl:operation>

  </wsdl:portType>

  <!-- ============================================================ -->
  <!-- System EPR -->
  <!-- ============================================================ -->

  <wsdl:portType name="SystemEPR"
    wsrf-rp:ResourceProperties="api:SystemEndpointProperties">
    <!-- add a file -->
    <wsdl:operation name="AddFile">
      <wsdl:input message="tns:addFileRequest" name="AddFileRequest"/>
      <wsdl:output message="tns:addFileResponse" name="AddFileResponse"/>
      <wsdl:fault name="DeploymentFault"
        message="tns:DeploymentFault"/>
    </wsdl:operation>

    <!-- initialize an application -->
    <wsdl:operation name="Initialize">
      <wsdl:input message="tns:initializeRequest" name="InitializeRequest"/>
      <wsdl:output message="tns:initializeResponse" name="InitializeResponse"/>
      <wsdl:fault name="DeploymentFault"
        message="tns:DeploymentFault"/>
    </wsdl:operation>

    <!-- resolve a path -->
    <wsdl:operation name="Resolve">
      <wsdl:input message="tns:resolveRequest" name="ResolveRequest"/>
```

```
    <wsdl:output message="tns:resolveResponse" name="ResolveResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>

  <!-- start an application -->
  <wsdl:operation name="Run">
    <wsdl:input message="tns:runRequest" name="RunRequest"/>
    <wsdl:output message="tns:runResponse" name="RunResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>

  <!-- terminate -->
  <wsdl:operation name="Terminate">
    <wsdl:input message="tns:terminateRequest" name="TerminateRequest"/>
    <wsdl:output message="tns:terminateResponse" name="TerminateResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>

  <!--  server status -->
  <wsdl:operation name="Ping">
    <wsdl:input message="tns:pingRequest" name="PingRequest"/>
    <wsdl:output message="tns:pingResponse" name="PingResponse"/>
    <wsdl:fault name="DeploymentFault"
      message="tns:DeploymentFault"/>
  </wsdl:operation>

  <!-- WSRF-RP -->
  <wsdl:operation name="GetResourceProperty">
    <wsdl:input name="GetResourcePropertyRequest"
      message="wsrf-rpw:GetResourcePropertyRequest"/>
    <wsdl:output name="GetResourcePropertyResponse"
      message="wsrf-rpw:GetResourcePropertyResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  </wsdl:operation>

  <wsdl:operation name="GetMultipleResourceProperties">
    <wsdl:input name="GetMultipleResourcePropertiesRequest"
      message="wsrf-rpw:GetMultipleResourcePropertiesRequest"/>
    <wsdl:output name="GetMultipleResourcePropertiesResponse"
      message="wsrf-rpw:GetMultipleResourcePropertiesResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rpw:ResourceUnknownFault"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
  </wsdl:operation>


  <!-- WS-RF Resource Lifetime: ImmediateResourceTermination -->

  <wsdl:operation name="Destroy">
    <wsdl:input
      name="DestroyRequest"
      message="wsrf-rlw:DestroyRequest"/>
    <wsdl:output name="DestroyResponse"
      message="wsrf-rlw:DestroyResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsrf-rlw:ResourceUnknownFault"/>
    <wsdl:fault name="ResourceNotDestroyedFault"
      message="wsrf-rlw:ResourceNotDestroyedFault"/>
  </wsdl:operation>

  <!-- ========== NotificationProducer Specific ============== -->
  <wsdl:operation name="Subscribe">
    <wsdl:input name="SubscribeRequest"
      message="wsntw:SubscribeRequest"/>
    <wsdl:output name="SubscribeResponse"
      message="wsntw:SubscribeResponse"/>
    <wsdl:fault name="ResourceUnknownFault"
      message="wsntw:ResourceUnknownFault"/>
    <wsdl:fault name="SubscribeCreationFailedFault"
      message="wsntw:SubscribeCreationFailedFault"/>
```

```
      <wsdl:fault name="TopicPathDialectUnknownFault"
        message="wsntw:TopicPathDialectUnknownFault"/>
    </wsdl:operation>

    <wsdl:operation name="GetCurrentMessage">
      <wsdl:input name="GetCurrentMessageRequest"
        message="wsntw:GetCurrentMessageRequest"/>
      <wsdl:output name="GetCurrentMessageResponse"
        message="wsntw:GetCurrentMessageResponse"/>
      <wsdl:fault name="ResourceUnknownFault"
        message="wsntw:ResourceUnknownFault"/>
      <wsdl:fault name="InvalidTopicExpressionFault"
        message="wsntw:InvalidTopicExpressionFault"/>
      <wsdl:fault name="TopicNotSupportedFault"
        message="wsntw:TopicNotSupportedFault"/>
      <wsdl:fault name="NoCurrentMessageOnTopicFault"
        message="wsntw:NoCurrentMessageOnTopicFault"/>
    </wsdl:operation>

  </wsdl:portType>

  <!-- ================================================================= -->
  <!-- Portal EPR binding to doc/lit SOAP1.2 -->
  <!-- ================================================================= -->
  <wsdl:binding name="PortalEPRBinding" type="tns:PortalEPR">
    <wsdlsoap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="Create">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="CreateRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CreateResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="Resolve">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="PortalResolveRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="PortalResolveResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="LookupSystem">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="LookupSystemRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="LookupSystemResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <!-- =========== WSRF-RP Specific ============== -->
    <wsdl:operation name="GetResourceProperty">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="GetResourcePropertyRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetResourcePropertyResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
```

```
      <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidResourcePropertyQNameFault">
      <wsdlsoap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="GetMultipleResourceProperties">
    <wsdlsoap:operation soapAction="" style="document"/>
    <wsdl:input name="GetMultipleResourcePropertiesRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="GetMultipleResourcePropertiesResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidResourcePropertyQNameFault">
      <wsdlsoap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <!-- =========== NotificationProducer Specific ============== -->
  <wsdl:operation name="Subscribe">
    <wsdlsoap:operation soapAction="" style="document"/>
    <wsdl:input name="SubscribeRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="SubscribeResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SubscribeCreationFailedFault">
      <wsdlsoap:fault name="SubscribeCreationFailedFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="TopicPathDialectUnknownFault">
      <wsdlsoap:fault name="TopicPathDialectUnknownFault" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="GetCurrentMessage">
    <wsdlsoap:operation soapAction="" style="document"/>
    <wsdl:input name="GetCurrentMessageRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="GetCurrentMessageResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidTopicExpressionFault">
      <wsdlsoap:fault name="InvalidTopicExpressionFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="TopicNotSupportedFault">
      <wsdlsoap:fault name="TopicNotSupportedFault" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="NoCurrentMessageOnTopicFault">
      <wsdlsoap:fault name="NoCurrentMessageOnTopicFault" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

</wsdl:binding>

<!-- ============================================================ -->
<!-- System binding to doc/lit SOAP1.2-->
<!-- ============================================================ -->

<wsdl:binding name="SystemEPRBinding" type="tns:SystemEPR">
  <wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

  <!-- AddFile also expects binary attachments, but that is not
  specified, as it creates too many interop problems if declared -->
```

```
    <wsdl:operation name="AddFile">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="AddFileRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="AddFileResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>


    <wsdl:operation name="Initialize">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="InitializeRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="InitializeResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>


    <wsdl:operation name="Resolve">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="ResolveRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="ResolveResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="Run">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="RunRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="RunResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="Terminate">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="TerminateRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="TerminateResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="Ping">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="PingRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="PingResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="DeploymentFault">
        <wsdlsoap:fault name="DeploymentFault" use="literal"/>
```

```
      </wsdl:fault>
    </wsdl:operation>

    <!-- ========== WSRF-RP Specific ============== -->
    <wsdl:operation name="GetResourceProperty">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="GetResourcePropertyRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetResourcePropertyResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="InvalidResourcePropertyQNameFault">
        <wsdlsoap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="GetMultipleResourceProperties">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="GetMultipleResourcePropertiesRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetMultipleResourcePropertiesResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="InvalidResourcePropertyQNameFault">
        <wsdlsoap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <!-- ========== NotificationProducer Specific ============== -->
    <wsdl:operation name="Subscribe">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="SubscribeRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="SubscribeResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="SubscribeCreationFailedFault">
        <wsdlsoap:fault name="SubscribeCreationFailedFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="TopicPathDialectUnknownFault">
        <wsdlsoap:fault name="TopicPathDialectUnknownFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="GetCurrentMessage">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="GetCurrentMessageRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetCurrentMessageResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="InvalidTopicExpressionFault">
        <wsdlsoap:fault name="InvalidTopicExpressionFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="TopicNotSupportedFault">
        <wsdlsoap:fault name="TopicNotSupportedFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="NoCurrentMessageOnTopicFault">
        <wsdlsoap:fault name="NoCurrentMessageOnTopicFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
```

```
    <!-- WS Resource Lifetime -->
    <wsdl:operation name="Destroy">
      <wsdlsoap:operation soapAction="" style="document"/>
      <wsdl:input name="DestroyRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="DestroyResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdlsoap:fault name="ResourceUnknownFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ResourceNotDestroyedFault">
        <wsdlsoap:fault name="ResourceNotDestroyedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>


  <!-- ========================================================== -->
  <!-- end service definitions-->
  <!-- ========================================================== -->


  <!-- ========================================================== -->
  <!-- end WSDL definitions-->
  <!-- ========================================================== -->
</wsdl:definitions>
```