Testing the Deployment API

Steve Loughran
2005-11-14

The CDDLM deployment API is a SOAP/WS-RF API to deploy systems. This is the recommended test plan for validating an implementation, or a deployment.

1. These are mostly functional tests; the underlying SOAP stack and language implementation(s) have to be present before these tests can take place.
2. In this document, XML namespace prefixes are used. All these prefixes have the same URI mapping as in the deployment API document.
3. Tests marked as "status: experimental" are tests for which compliance is not required for compliance with the deployment API specification.

Test Documents

Many of the tests will submit documents for deployment. The CDL files used will be described in the appendix. Here are their goals

api-test-doc-1: valid-descriptor

An empty descriptor that deploys a valid application. It will deploy and stay deployed until undeployed. It contains some simple values that can be resolved, in order to test resolution.

api-test-doc-2: parse-time-error in CDL

This document is valid CDL, but contains a resolution-time error and cannot be instantiated.

api-test-doc-3: fail-to-run

This document is valid CDL, but contains a cdl:lazy reference which will not resolve at run time, so the deployment will fail then.

api-test-doc-4: fail-on-ping

This document is valid CDL, but contains a component designed to fail on a liveness test.

api-test-doc-5: fail-on-terminate

This document contains a special test component which will raise an exception during termination.

api-test-doc-6: slow-start

This document contains a special test component that takes a specified number of second to start. This can be used to verify that starting is an asynchronous operation.

Portal and Creation tests

The following tests mostly test the portal EPR

api-1: portal-wsdm-properties

Verify that a portal exports the following resources, and that they match their specifications in the WSRF, WS-N and WSDM specifications.

Resource	*expected*
`muws-p1-xs:ResourceId`	A URI
`muws-p1-xs:ManageabilityCapability`	The list of manageability capabilities. This must include those listed in section 6.1.2 of the deployment API.
`wsnt:Topics`	This must list notification topics, including that for resource properties changing, and for the `SystemCreatedEvent`
`wsnt:FixedTopicSet`	
`wsnt:TopicExpressionDialects`	

The list of manageability capabilities must include the following elements

```
<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>
```

api-2: get-portal-state

Examine the state of a portal; including languages and notification mechanisms

Actions:

1. `SystemEPR->GetProperty(api:StaticPortalStatus) => status`

Assertions:

- Response is valid by the XSD declarations of the `StaticPortalStatus` type.
- `/languages/item/name[.="CDL"]` resolves successfully.
- `/notifications/item[.="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd"]` resolves successfully.

Other languages and notification mechanisms may be listed: their presence is not an error.

api-3: Create a System, then destroy it

Create a system , then destroy it immediately.

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
```

```
3. SystemEPR->GetProperty(api:CreatedTime) => CreatedTime
4. SystemEPR->Destroy => SUCCESS.
```

Assertions:

- The system EPR returned is a valid URL.
- A `GetProperty(muws-p1-xs:ResourceId)` MUWS capability is valid.
- The `CreatedTime` property is a valid `xsd:dateTime` element.
- The system EPR responds successfully to a `<Destroy/>` operation.

api-4: Create a System, then destroy it

Status: experimental

In a newly created system, the `StartedTime` and `TerminatedTime` properties

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(api:StartedTime) =>  StartedTime
3. SystemEPR->GetProperty(api:TerminatedTime) =>   TerminatedTime
4. SystemEPR->Destroy => SUCCESS.
```

Assertions:

- The system EPR returned is a valid URL.
- The `StartedTime` and `TerminatedTime` properties return a value to indicate that these times have not been reached yet. There is no normative definition of what this may be.
- The system EPR responds successfully to a `<Destroy/>` operation.

api-5: After destruction, properties are not readable

Create a system , then destroy it immediately.

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
3. SystemEPR->GetProperty(api:CreatedTime) => CreatedTime
4. SystemEPR->Destroy => SUCCESS
5. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ERROR
```

Assertions:

- The system EPR returned is a valid URL.
- The EPR has a resource ID
- The system EPR responds successfully to a `<Destroy/>` operation.
- After destruction, the EPR no longer has a resource ID property.

api-6: After destruction, operations are not allowed

Create a system , then destroy it immediately.

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->Destroy => SUCCESS
3. SystemEPR->Ping() => ERROR
```

Assertions:

- The system EPR returned is a valid URL.
- The EPR has a resource ID
- The system EPR responds successfully to a `<Destroy/>` operation.
- After destruction, the EPR no longer responds to ping operations

api-7: Created Systems that are never instantiated get destroyed eventually.

Status: experimental

To prevent a DoS attack, orphan systems should eventually be destroyed. A test for this is to create a system, do nothing and check a few hours or days later. The EPR should no longer be valid.

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
3. Sleep(24 hours)
4. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ERROR
```

Assertions:

- After an extended period, the server-side structures supporting a created but undeployed system are purged and the endpoint no longer resolves to new requests

api-8: Creating too many Systems.

Status: experimental

This test explores what happens when 'too many' systems are created. There is nothing in the specification to cover this, but one would expect that there is an operational limit on how many systems can be created. The goal of this test is to see what this limit is and what happens when it is reached.

```
1. PortalEPR->Create => SystemEPR
2. Goto #1
```

api-9: Created systems

After creating a system, verify that it is in the list of created systems by looking up its ResourceID in the list of deployed systems.

Actions

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
3. PortalEPR->GetProperty(api:ActiveSystems) => ActiveSystems
4. SystemEPR->Destroy => SUCCESS
5. PortalEPR->GetProperty(api:ActiveSystems) => ActiveSystems2
```

Assertions

•`SystemEPR` is found in `ActiveSystems`.

•`SystemEPR` is not found in `ActiveSystems2`.

In this context, "found" means that an EPR with the same URI and reference parameters exists in the list, or more formally, an EPR whose resourceID is obtained equals that of the original EPR.

Create a system with invalid hostname hint.

Create a system with an invalid hostname hint. The hint should be ignored and a valid EPR returned as before.

```
1. PortalEPR->Create("no-name.example.org") => SystemEPR
2. SystemEPR->Destroy => SUCCESS
```

api-10: Create a system with a valid hostname hint.

Create a system as above, this time with an valid hostname hint. The EPR should contain that hostname.

```
1. PortalEPR->Create("localhost") => SystemEPR
2. SystemEPR->Destroy => SUCCESS
```

This test is hard to perform, as it needs a set of two or more (potentially virtual) hosts, all part of the deployment infrastructure.

api-11: Verify uniqueness of ResourceId values.

ResourceId values are required by WSDM to be strongly unique. This is hard to test. Simple checks can verify that during a single session, unique identifiers are returned. Long-term tests are more thorough, but harder to perform.

Action:

1. All tests that create a system should get its ResourceId and verify that it has not been returned by a previous creation operation.

This will verify that IDs are unique during a single session, but not that they are unique across machines or server restarts

api-12: Verify multiple systems can be created.

Verify that a number of systems can be created at the same time.

```
1. PortalEPR->Create => SystemEPR
2. PortalEPR->Create => SystemEPR2
3. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
4. SystemEPR2->GetProperty(muws-p1-xs:ResourceId) => ResourceID2
5. PortalEPR->GetProperty(api:ActiveSystems) => ActiveSystems
6. SystemEPR->Destroy => SUCCESS
7. SystemEPR2->Destroy => SUCCESS
```

Assertions

•`SystemEPR != SystemEPR2`

•`ResourceID != ResourceID2`

•`ActiveSystems` contains both System Endpoint References

api-13: Look up a system

Verify that a number of systems can be created at the same time.

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
3. PortalEPR->LookupSystem(ResourceId) => SystemEPR2
4. SystemEPR2->GetProperty(muws-p1-xs:ResourceId) => ResourceId2
```

Assertions

•`ResourceId == ResourceId2`

Paired-portal tests

The deployment API is designed to support multiple portals providing access to the same set of

systems. If the implementation enables this, then it must be tested

api-14: Create a system on one portal, view in another

After creating a system, look in the "deployed systems" property of the peer portal for a system with the same ResourceID. Once destroyed in one portal, all other EPRs must be invalid, even if they are mapped to a different address.

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ResourceId
3. PortalEPR2->LookupSystem(ResourceId) => SystemEPR2
4. SystemEPR2->GetProperty(muws-p1-xs:ResourceId) => ResourceId2
5. PortalEPR2->GetProperty(api:ActiveSystems) => ActiveSystems
6. SystemEPR2->Destroy => SUCCESS
7. SystemEPR->GetProperty(muws-p1-xs:ResourceId) => ERROR
```

Assertions

•`ResourceId == ResourceId2`

•`SystemEPR` or `SystemEPR2` is found in `ActiveSystems`

System Creation Notification tests

The following tests require WS-Notification support in the test harness.

Test systems may wish to provide a private property listing all eprs subscribing to a system or portal EPR. This will enable the outcome of these tests to be easily verified.

api-15: Subscribe to System creation events

The test system can subscribe via the portal for system creation events.

api-16: Unsubscribe

After subscribing to events, An unsubscribe should unsubscribe them. After unsubscribing, creating a new system must not result in an event being sent to the (now unsubscribed) endpoint.

api-17: Create a System while subscribed.

While subscribed to a portal for creation events, create a system.

Assertions:

- A system creation event is received.

api-18: Create a System while subscribed with an invalid endpoint

Subscribe to a portal for creation events with a notification EPR that is invalid. There are the following types of invalid portal

- Nonexistent URL (e.g. "http://notifications.example.org/epr1").
- endpoint with no open connection (e.g. "http://localhost:8081/epr1"), assuming port 8081 is closed
- 404 URL (e.g. "http://deployment.sourceforge.net/example/epr1").

Assertions:

- In all cases, the notification delivery failure is noted, the endpoint is no longer subscribed

api-19: Create a System while subscribed to a broken endpoint

Subscribe to a portal for creation events with a notification EPR that is broken. There are the following types of broken endpoint

1. EPR always responds with a SOAPFault of some kind
2. EPR keeps the connection open, and does not terminate the connection.

Test #1 should fail as in the previous test. Test#2 should eventually time out, and be recorded by the portal as a failure.

api-20: Multiple system event subscribers

Multiple EPRs can be subscribed to system events; events will be delivered to all. When one endpoint unsubscribes, events will still be delivered to the remainder.

api-21: Invalid XML posts

Each SOAP Operation must be tested with

1. XML documents that do not match the schema. These must be rejected.
2. XML documents that contain extra data where extra data is permitted. These must be accepted.
3. The portal must also reject SOAP actions of the systems themselves, such as an initialize request.

System Initialization

These tests that are performed against a created System endpoint. These probe the endpoints' language abilities, and the support for options.

In order to test valid deployments, we will need test components whose behaviour are well defined, and which are implemented on all test systems.

api-22: deploy a valid configuration inline

A CDL descriptor is submitted inline

Actions:

```
1. PortalEPR->Create => SystemEPR
2. SystemEPR->Initalize(valid-cdl) => SUCCESS

 <api:initializeRequest>
   <api:descriptor
     language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
     <api:body>
       //insert api-test-doc-1 document here
     </api:body>
   </api:descriptor>
 </api:initializeRequest>

3. SystemEPR->Terminate() => SUCCESS
4. SystemEPR->Destroy() => SUCCESS
```

Assertions:

- That the application initializes.

api-23: deploy a valid configuration via AddFile

A CDL descriptor is uploaded via `AddFile`, the returned URL being used for the deployment.

Actions:

```
   1. PortalEPR->Create => SystemEPR
   2. SystemEPR->AddFile => [URL11,...]

<api:addFileRequest>
  <api:name>valid-cdl.cdl</api:name>
  <api:mimetype>text/xml</api:mimetype>
  <api:schema>file</api:schema>   //or http
  <api:data>
      //base 64 encoding of api-test-doc-1 document
  </api:data>
</api:addFileRequest>

   3. SystemEPR->Initalize(url1) => SUCCESS
    <api:initializeRequest>
      <api:descriptor
        language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
        <api:reference>${URL1}</api:reference>
      </api:descriptor>
    </api:initializeRequest>
   4. SystemEPR->Terminate() => SUCCESS
   5. SystemEPR->Destroy() => SUCCESS
```

Assertions:

- That `AddFile` uploads a file and returns at least one URL pointing to where it was uploaded.
- If an http: URL, that URL *may* be visible to the testing application.
- That the application initializes with the URL as a parameter in the request
- If an http: URL, that URL is not visible to the testing application after the System is destroyed.

api-24: Unknown options are ignored if mustUnderstand="false"

Using the test above, the initialize request is extended with a set of options, all of which are marked `mustUnderstand="false"` or with no mustUnderstand value at all:

```
      <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://example.org/map"
            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:option name="http://example.org/bool"
      <api:option name="http://example.org/xml"
          mustUnderstand="false">
        <api:data>
          <cdl:import location=".."/>
        </api:data>
      </api:option>
    </api:options>
```

Assertions

- The unknown options are ignored

api-25: Unknown options are rejected

The previous test is repeated, this time with an option marked `mustUnderstand="true"`.

```
    <api:options>
      <api:option name="http://example.org/options/1" mustUnderstand="true">
        <api:string>string value</api:string>
      </api:option>
      <api:option name="http://example.org/options/2">
        <api:integer>-8</api:integer>
      </api:option>
    </api:options>
```

Assertions

- The `<Initialize>` operation fails with an error that is a derivative of `api:DeploymentFaultType`.
- Ideally, the faultcode QName should be (`"http://gridforge.org/cddlm/serviceAPI/faults/2004/10/11/"`, `"not-understood"`), though this is not (currently) required.

api-26: Unknown languages are rejected

A system is sent the following Initialize document:

```
  <api:initializeRequest>
    <api:descriptor
      language="http://example.org/language/unknown"
        xmlns:odd="http://example.org/language/unknown"
        >
      <api:body>
        <odd:lisp version="1.0">
          ((language 3 4) (unknown))
        </odd:lisp>
      </api:body>
    </api:descriptor>
  </api:initializeRequest>
```

Assertions

- The `<Initialize>` operation fails with an error that is a derivative of `api:DeploymentFaultType`.
- Ideally, the faultcode QName should be (`"http://gridforge.org/cddlm/serviceAPI/faults/2004/10/11/"`,

```
    "unsupported-language")
```

api-27: Initialize with an invalid URL.

Status: experimental.

Initialize systems using the following template for a request, with the values of ${URL} specified below:

```
  <api:initializeRequest>
    <api:descriptor
      language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
      <api:reference>${URL}</api:reference>
    </api:descriptor>
  </api:initializeRequest>
```

These tests are only valid when a system is online, and the server has proxy settings configured to allow Internet access.

Url	*test*
`http://noname.example.org`	Hostname lookup fails
`http://www.ggf.org/`	Not valid XML. Parsing should fail
`file:/etc/passwd`	On a Unix system, this should be rejected for security reasons

api-28: Initialize with descriptor as an attachment

Status: experimental.

If attachments (DIME, MTOM or SwA) are supported, then deployment using the supported attachment mechanisms must be tested. The tests in such as case are

Succeeding

- Send an <Initialize> an request with the inline descriptor sent as an MTOM attachment.
- Send an <Initialize> an request with the URL-referenced descriptor sent as an SwA or DIME attachment.

Failing:

- Deploy request with both an inline document and attachment ->Unknown action. Ignore the attachment?
- Deploy request with very large (128+MB) attachment (stress test). This may be rejected as too large, or it may be accepted. Most of the request can be generated white space.
- Deploy request that has an include statement whose URI is resolved against another attachment.

Deploy Invalid Configurations

The implementation must parse and reject those deployment configurations that can be assessed as invalid during the initialization phase. That includes

- configurations with looped dependencies
- configurations with unresolved references
- configurations whose values does not match the (enforced) type system.

- Inclusion of a non-resolvable URI

api-29: Initialization fails for invalid CDL content.

A system is sent the following Initialize document:

```
<api:initializeRequest>
  <api:descriptor
    language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
    <api:body>
      <cdl:cdl >
        <cdl:documentation>
          This system extends a reference that is unknown.
          Resolution MUST fail
        </cdl:documentation>
        <cdl:system>
          <Component cdl:extends="unknown">
          </Component>
        </cdl:system>
      </cdl:cdl>
    </api:body>
  </api:descriptor>
</api:initializeRequest>
```

Assertions

- The `<Initialize>` operation fails with an error that is a derivative of `api:DeploymentFaultType`.

DeployAPI/Component Model Combined Tests

api-30: Deploy a system that fails during the transition to running state.

api-31: Deploy a system that fails during a <Ping /> operation.

api-32: Deploy a system that successfully terminates (workflow-style)

api-33: Deploy a system that does not respond in a timely fashion to a <Ping/> operation.

Status: experimental.

This test needs a component that can be configured to pause during a <Ping/> operation, so that the client's behaviour during a long delay can be explored.

System notifications

api-34: Subscribe to lifecycle events

A client can subscribe to system lifecycle events.

api-35: Receive lifecycle events when a system is destroyed

A client that is subscribed to lifecycle events, receives a notification when a system is destroyed.

api-36: Receive lifecycle events when a system fails

With the Fault component, we can expect to be notified when a system enters the failed state.

api-37: Support multiple event listeners per system.

More than one endpoint can subscribe to lifecycle events; all will receive them. A more advanced test would verify that the failure/blocking action of one endpoint will not cause the other recipients to be denied their messages.

api-38: Receive lifecycle events when a system enters the running state.

Subscribers to lifecycle events receive notifications when a system enters the running state.

Notes

Writing these tests has thrown up the interesting problem is that there is no formal way of defining equivalence of EPRs. The sole mechanism we have is to get the ResourceId value, and compare them. ResourceID comparison must be by case-insensitive string comparision. No attempt should be made to perform 'URL-aware' comparison logic, such as assuming that "`http://host:80/e`" is equivalent to "`http://HOST/%65`". This is not what RFC 2616 Section 3.2.3, URI equivalence, mandates.

Appendix A: test descriptors

api-test-doc-1: valid-descriptor

```
<cdl:cdl
  xmlns:w="urn:webapps"
  xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
  <cdl:configuration>
    <w:webapp-base>
      <app>security</app>
      <app>logging</app>
    </w:webapp-base>

    <w:webapps2 cdl:extends="w:webapp-base">
      <app>testing</app>
    </w:webapps2>

    <w:server>
      <port>8080</port>
      <security>true</security>
      <webapps/>
    </w:server>

    <w:tomcat cdl:extends="w:server">
      <basedir>
      </basedir>
    </w:tomcat>

  </cdl:configuration>
  <cdl:system>
    <w:deployment cdl:extends="w:tomcat">
      <w:webapps2/>
    </w:deployment>
  </cdl:system>
</cdl:cdl>
```

api-test-doc-2: parse-time-error in CDL

This document is valid CDL, but contains a resolution-time error and cannot be instantiated; the `cdl:extends` attribute refers to a nonexistent component.

```
  <api:initializeRequest>
     <api:descriptor
       language="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
       <api:body>
         <cdl:cdl >
           <cdl:documentation>
             This test extends a reference that is unknown.
             Resolution MUST fail
           </cdl:documentation>
           <cdl:system>
             <Component cdl:extends="unknown">
             </Component>
           </cdl:system>
         </cdl:cdl>
       </api:body>
     </api:descriptor>
   </api:initializeRequest>
```