Interaction between CDDLM & JSDL
EMS Architecture Document Series
7 June 2006, Draft 04

1 Introduction

This document provides a more detailed description of Scenarios 3.1 and 3.2 (Deployment and Configuration of Application and BES Instance) of the "OGSA EMS Architecture Scenarios Version 1.0".

This is a DRAFT DOCUMENT and is intended for WG review only.

2 Definitions

For the purposes of this document we define:

- Life Cycle: Both provisioning and deployment actions have a life cycle – they are created, used, and eventually destroyed. A mechanism is needed for activities to 'register' their use and interest in a particular activity. Before an activity is destroyed there have to be no outstanding references. [Look at EGA provisioning state model.]
- Provisioning: The instantiation of an environment on a resource (i.e. a deployment activity) that may be used by more than one activity. Generally these deployments are 'heavyweight' in that they may take substantial time to instantiate. Provisioning actions may be triggered by a specific activity (e.g. following a manual job submission) or as a result of general response (as defined by a policy) to the state of current resources in the system.
    - For example, "if my available free RHEL4 resources have dropped to 5%, dynamically provision new nodes until the total free nodes have increased to 15%."
- Deployment: The instantiation of an environment on a resource to meet the needs of a specific activity such as a submitted job. This is generally a 'lightweight' action in that may just provide a binary, dataset, etc., onto the resource. Once used by the specified activity that environment may be removed immediately; or left as part of a 'cache' to be cleaned up or reused at a later date. (A 'keep alive' model might be needed in the latter case.)
    - For example, an activity wants to run Gaussian: download and install the required version for this platform.
- Activity: The smallest part of possibly a larger sequence of activities generated from a single submitted job.
- Job Manager (JM): The entity that accepts a 'job' (defined by a JSDL document) from the user agent. The JM coordinates further invocations and interactions with other elements of the EMS architecture.
- JSDL: Job Submission Description Language (GFD.56)
- Deployment Service (DS): The entity that accepts a 'deployment' action (defined by a CDL document) to instantiate an environment on the resource to support a

BES job. The DS provides the bootstrapping for all deployment/provisioning actions. (GFD.69)

- CDL: Configuration Description Language
- CDL document template: A CDL document that still contains unresolved type references, e.g., lazy elements. (Note that this is not a CDDLM defined term. It is used for convenience.)
- Basic Execution Service (BES): The entity that accepts a JSDL document and instantiates an Activity. BES does not do provisioning/deployment.

3 Scenarios

3.1 Deploying an Application (OGSA EMS Scenarios §3.1)

In this scenario a simple POSIX application (BLAST) needs to be deployed to support an activity that is to be initiated within a BES container. In this scenario the BES instance has already been determined and no BLAST application resides on that platform.

[NB: This (and other steps) should be converted to a UML diagram once agreed.]

1. The User Agent submits a JSDL document (which requires the BLAST application) to the JM requesting that the job (as specified by the JSDL document) be sent to a specific BES instance.
    a. For the purposes of this scenario the BES instance is specified as an argument to the submission operation and is outside the JSDL document.
2. The JM queries the Information Service (IS), or the platform directly, to discover if the specified BES instance resides on a resource which already has the BLAST application.
    a. For the purposes of this scenario it is assumed that an empty set is returned meaning that the specified BES instance does not have the BLAST application located on it. The BLAST application has to be installed so that the job (as described by the JSDL document) can be successfully executed.
3. A BLAST binary suitable for the platform on which the BES instance resides is discovered from an IS and a CDL document describing its deployment requirements is retrieved.
    a. For the purposes of this scenario we assume the CDL document has a number of ‘lazy elements’, i.e., it is a ‘template’.
    b. Later scenarios will refine this discovery step further, for example, by detailing an interaction with an ACS repository.
4. The *CDL document template* MAY have some of its ‘lazy elements’ filled in by the JM using information retrieved from some IS.
    a. For example, the name of the resource (host) on which to deploy may be defined as a lazy element. The JM knows the BES container and may have a specific requirement from the user to execute the job on a specific resource (host).
5. The JM queries the Information Service (IS) to discover a DS that can deploy to the platform hosting the BES instance.

    a. In the simplest case the DS is located on the platform hosting the BES instance or there is some other well known (pre-configured) relationship.
    b. The DS MAY act as a proxy for deployment on nodes OTHER than the one it is running on.
    c. The DS MUST advertise (somewhere in a currently unspecified manner) which nodes it is capable of deploying stuff onto, e.g., front end node for a cluster.
6. The CDL document (which may still contain lazy elements) is then passed to the DS that can deploy to the platform hosting the BES instance.
    a. The DS MUST complete the remaining lazy elements in the CDL document. For example, the name of the resource (host) on which to deploy may be defined as a lazy element. If the JM has no specific requirement on which resource to deploy it may leave the element as lazy for the DS to decide. [CDDLM example.]
7. The DS returns an EPR to the JM providing a reference to the installed software.
8. The IS entry related to the resource is updated with the location of the BLAST application by the JM.
    a. This prevents the removal of the resource as 'something'—the BLAST application—is 'using' it.
9. The JM can use the EPR from the DS to find the defined environment variables to be pulled back in order to refine the JSDL document. This is effectively an operation to retrieve relevant chunks of the XML deployment tree (i.e. the instantiated CDL document) back to the JM for insertion into JSDL document.
10. The complete JSDL document is then sent to the BES instance on that platform.
11. The IS entry related to the BLAST application is updated with the information that the BES Activity now has an 'interest' in the application.
    a. If another BES Activity uses this BLAST application and performs this registration it will prevent un-deployment of the BLAST application even if the initial activity has completed.
12. The BES Activity completes successfully.
13. The interest of the BES Activity in the BLAST application is removed from the IS.
14. If no other activities are registering an interest in the BLAST application on this resource then the undeploy operation MAY be invoked on the Deployment Service to remove this BLAST application from the resource.
    a. The application may remain installed for some period of time if it is expected that it may be re-used.
    b. If the BLAST application is removed then its entry in the IS is also removed.
15. The User Agent is informed that the job is complete.

Missing bits:

- May be multiple things that need to be deployed for the activity
- ~~And deployed things may be cached for later re-use or cleanup.~~

3.2 Provisioning a BES instance (OGSA EMS Scenarios §3.2)

Provisioning is an activity initiated by policy and not directly in response to a specific job or some other activity. In the following scenario a provisioning service (PS) monitors 'the system' (in this particular example a compute centre) and undertakes provisioning actions in response to established policy. In effect the PS provides an autonomic capability.

A prerequisite is that there are available free platforms that a DS can deploy on.

1. The PS monitors the BES instances within the compute centre. It determines the current capacity of the BES instances registered in the IS and checks that the capacity is within the limits defined by its policy.
2. The policy indicates that more BES instances need to be provisioned.
3. The PS retrieves a list of platforms (bare OS's) that can be used from the IS and their associated DS.
4. The PS retrieves a CDL document describing the instantiation of a BES instance.
    a. Later scenarios will refine this step further, for example, by detailing an interaction with an ACS repository.
5. For each selected platform, the PS submits a CDL document to the DS that can deploy to that platform.
6. The endpoint returned on success by the DS is used to register the presence of the BES instance in the IS.

At stage 2 the policy could also determine that there are too many BES instances, in which case the next stage would generate a undeploy action to the appropriate DS to remove the BES instance which would then be un-registered from the IS.

3.3 Selected Job Execution (OGSA EMS Scenarios §2.3)

In this scenario a simple POSIX application (BLAST) needs to be deployed to support an activity that is to be initiated within a BES container. In this scenario the BES instance has not been determined at submission time.

[NB: This (and other steps) should be converted to a UML diagram once agreed.]

1. The User Agent submits a JSDL document (which requires the BLAST application) to the JM requesting that the job (as specified by the JSDL document) be executed. The BES instance on which the job is to be executed is *not* specified.
2. The JM queries the Execution Planning Service (EPS) for a list of candidate execution plans for the job, and uses the first plan returned.
    a. For the purposes of this scenario the first plan returned is considered to be the best choice.
    b. A plan is made up of at least a JSDL document and a reference to a BES container.
    c. For the purposes of this scenario, it is assumed that the execution plan does not specify a BES container that already has the BLAST application

located on it, and that the application has to be installed so that the job (as described by the JSDL document) can be successfully executed.
    d. Note that the EPS may have returned an alternate (refined) JSDL document to use instead of the one originally submitted. This is to allow the rewriting of things such as abstract resources (such as one identifying the abstract name of some sequence database) into a set of requirements for making that resource available at the suggested BES container (e.g. by extending the set of DataStaging elements, or by providing an appropriate CDL document).
    e. Because the application is not installed (as per assumption above) the candidate execution plan includes a CDL suitable for making the application available on that BES container. This will have already identified a suitable BLAST binary for the platform in question.
    f. It is up to the implementation of the EPS to query whatever Information Service (IS) that is necessary to provide the candidate execution plans as described.
    g. For the purposes of this scenario we assume the CDL document has a number of 'lazy elements', i.e., it is a 'template'.
    h. Later scenarios will refine this discovery step further, for example, by detailing an interaction with an ACS repository.
3. The *CDL document template* MAY have some of its 'lazy elements' filled in by the JM using information retrieved from some IS.
    a. For example, the name of the resource (host) on which to deploy may be defined as a lazy element. The JM knows the BES container and may have a specific requirement from the user to execute the job on a specific resource (host).
4. The JM queries the Information Service (IS) to discover a DS that can deploy to the platform hosting the BES instance.
    a. In the simplest case the DS is located on the platform hosting the BES instance or there is some other well known (pre-configured) relationship.
    b. The DS MAY act as a proxy for deployment on nodes OTHER than the one it is running on.
    c. The DS MUST advertise (somewhere in a currently unspecified manner) which nodes it is capable of deploying stuff onto, e.g., front end node for a cluster.
    d. If this fails, the next best execution plan (from those provided in step 2 above) is chosen instead as a failover option, or if there are no other plans left, the job execution overall fails.
5. The CDL document (which may still contain lazy elements) is then passed to the DS that can deploy to the platform hosting the BES instance.
    a. The DS MUST complete the remaining lazy elements in the CDL document. For example, the name of the resource (host) on which to deploy may be defined as a lazy element. If the JM has no specific requirement on which resource to deploy it may leave the element as lazy for the DS to decide. [CDDLM example.]

    b. If this fails, the next best execution plan (from those provided in step 2 above) is chosen instead as a failover option, or if there are no other plans left, the job execution overall fails.
6. The DS returns an EPR to the JM providing a reference to the installed software.
7. The IS entry related to the resource is updated with the location of the BLAST application by the JM.
    a. This prevents the removal of the resource as 'something'—the BLAST application—is 'using' it.
8. The JM can use the EPR from the DS to find the defined environment variables to be pulled back in order to refine the JSDL document. This is effectively an operation to retrieve relevant chunks of the XML deployment tree (i.e. the instantiated CDL document) back to the JM for insertion into JSDL document.
9. The complete JSDL document is then sent to the BES instance on that platform.
    a. If this fails, the next best execution plan (from those provided in step 2 above) is chosen instead as a failover option, or if there are no other plans left, the job execution overall fails.
10. The IS entry related to the BLAST application is updated with the information that the BES Activity now has an 'interest' in the application.
    a. If another BES Activity uses this BLAST application and performs this registration it will prevent un-deployment of the BLAST application even if the initial activity has completed.
11. The BES Activity completes successfully.
12. The interest of the BES Activity in the BLAST application is removed from the IS.
13. If no other activities are registering an interest in the BLAST application on this resource then the undeploy operation MAY be invoked on the Deployment Service to remove this BLAST application from the resource.
    a. The application may remain installed for some period of time if it is expected that it may be re-used.
    b. If the BLAST application is removed then its entry in the IS is also removed.
14. The User Agent is informed that the job is complete.

3.4 Deploying an Application using ACS (OGSA EMS Scenarios §3.3)

In this scenario a simple POSIX application (BLAST) needs to be deployed to support an activity that is to be initiated within a BES container. In this scenario the BES instance has already been determined and no BLAST application resides on that platform.

[NB: This (and other steps) should be converted to a UML diagram once agreed.]

1. The User Agent submits a JSDL document, which requires the BLAST application, to the JM. The BES container is specified as an argument to the submission operation. (see previous scenarios)
2. The JM determines that the BES container does not have BLAST. (see previous scenarios)

    a. It is assumed that an Application Archive (AA) that contains all information required to install and configure BLAST is already registered in an Application Contents Services (ACS) Repository in the system.
    b. Note that an AA instance is addressable by an EPR.
    c. For simplicity, it is assumed that there is a single ACS repository in the system.
3. The JM uses the contents of the JSDL ApplicationName element to search for a matching AA in the ACS Repository using the *LookupArchives* operation. The EPR of the AA describing the matching BLAST application is returned.
    a. It is also possible for the AA EPR to be provided as part of the submission (e.g., inside the JSDL document).
    b. For simplicity it is assumed that name matching between ApplicationName and AA is sufficient.
    c. In scenarios using the EPS this search-and-match is done by the EPS.
4. The JM uses the AA *GetContents* operation to retrieve the CDL document describing the BLAST deployment requirements.
    a. For the purposes of this scenario we assume the CDL document has a number of 'lazy elements', i.e., it is a 'template'. (see previous scenarios)
5. The *CDL document template* MAY have some of its 'lazy elements' filled in by the JM using information retrieved from some IS. (see previous scenarios)
6. The JM queries the Information Service (IS) to discover a DS that can deploy to the platform hosting the BES instance. (see previous scenarios)
7. The CDL document (which may still contain lazy elements) is then passed to the DS that can deploy to the platform hosting the BES instance. (see previous scenarios)
    a. There is behind-the-scenes interaction between the DS and the ACS Repository. For example, the DS retrieves the binary corresponding to the BLAST application from the ACS Repository.
8. The DS returns an EPR to the JM providing a reference to the installed software. (see previous scenarios)
9. …remaining steps as in previous scenarios…

4 Worked Example (mainly for §3.1)

4.1 Running a BLAST (Basic Local Alignment Search Tool) job

4.1.1 JSDL Document Submitted to the Job Manager

According to the scenario above the JSDL document MUST contain at a minimum:

1. The application name

Highlighted text indicates portions that may need refinement or have linkage to CDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.ggf.org/jsdl/2005/11/jsdl jsdl.xsd
```

```
http://schemas.ggf.org/jsdl/2005/11/jsdl-posix jsdl-posix.xsd">
  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>Blast1</jsdl:JobName>
      <jsdl:Description>Blast query number 1</jsdl:Description>
      <jsdl:JobProject>BlastProject</jsdl:JobProject>
    </jsdl:JobIdentification>
    <jsdl:Application>
      <jsdl:ApplicationName>BlastN</jsdl:ApplicationName>
      <jsdl:ApplicationVersion>2.2.13</jsdl:ApplicationVersion>
      <jsdl:Description>BlastN performs nucleotide similarity
         searching</jsdl:Description>
      <jsdl-posix:POSIXApplication>
         <!-- EXECUTABLE LOCATION NOT KNOWN YET -->
        <jsdl-posix:Argument>-p</jsdl-posix:Argument>
        <jsdl-posix:Argument>blastn</jsdl-posix:Argument>
        <jsdl-posix:Argument>-d</jsdl-posix:Argument>
        <jsdl-posix:Argument>est</jsdl-posix:Argument>
        <jsdl-posix:Argument>-T</jsdl-posix:Argument>
        <jsdl-posix:Argument>T</jsdl-posix:Argument>
        <jsdl-posix:Input filesystemName="HOME">
              sequences1.txt</jsdl-posix:Input>
        <jsdl-posix:Output filesystemName="HOME">
              sequences1.html</jsdl-posix:Output>
        <jsdl-posix:Error filesystemName="HOME">
              sequences1.err</jsdl-posix:Error>
        <jsdl-posix:WorkingDirectory filesystemName="HOME">
              blastqueries</jsdl-posix:WorkingDirectory>
         <!--ENVIRONMENT TO BE INSERTED BY JM POST DEPLOYMENT-->
         <!--Limits to be defined based on what the user is allowed-->
        <jsdl-posix:UserName>csmith</jsdl-posix:UserName>
        <jsdl-posix:GroupName>bio</jsdl-posix:GroupName>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <!--Once the host is determined it is included here-->
      <jsdl:FileSystem name="TMP">
        <jsdl:FileSystemType>temporary</jsdl:FileSystemType>
        <jsdl:Description>Well-known ‘name’ for temporary space that
              does not necessarily persist after the job terminates.
         </jsdl:Description>
        <jsdl:DiskSpace>
         <jsdl:LowerBoundedRange>10737418240.0</jsdl:LowerBoundedRange>
        </jsdl:DiskSpace>
         <!--Detailed setup to be provided by a CDL document.
              Some values may be added here when refining.-->
      </jsdl:FileSystem>
      <jsdl:FileSystem name="HOME">
        <jsdl:FileSystemType>normal</jsdl:FileSystemType>
        <jsdl:Description>Chris's home directory</jsdl:Description>
         <!--Detailed setup to be provided by a CDL document.
              Some values may be added here when refining.-->
      </jsdl:FileSystem>
      <jsdl:ExclusiveExecution>true</jsdl:ExclusiveExecution>
      <jsdl:TotalCPUCount>
        <jsdl:Exact>1.0</jsdl:Exact>
      </jsdl:TotalCPUCount>
```

```
    </jsdl:Resources>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.txt</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:Source>
        <jsdl:URI>
          http://csmith.otherhost.com/blastqueries/sequences1.txt
        </jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.html</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:Target>
        <jsdl:URI>
          http://csmith.otherhost.com/blastqueries/sequences1.html
        </jsdl:URI>
      </jsdl:Target>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.err</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>append</jsdl:CreationFlag>
      <jsdl:Target>
        <jsdl:URI>
          http://csmith.otherhost.com/blastqueries/sequences1.err
        </jsdl:URI>
      </jsdl:Target>
    </jsdl:DataStaging>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

4.1.2 CDL Document retrieved by JM to prepare for deployment

The initial CDL template retrieved by the JM.

4.1.3 CDL Document submitted by JM to DS

The CDL template after the JM has (possibly) added some values, e.g., hostnames.

4.1.4 CDL document after DS has finished deployment

The completely resolved CDL document (no lazy elements).

4.1.5 JSDL Document submitted to the BES Instance

The final, refined, JSDL document submitted to BES for execution. Refinements of the initial document are highlighted.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.ggf.org/jsdl/2005/11/jsdl jsdl.xsd
```

```
http://schemas.ggf.org/jsdl/2005/11/jsdl-posix jsdl-posix.xsd">
  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>Blast1</jsdl:JobName>
      <jsdl:Description>Blast query number 1</jsdl:Description>
      <jsdl:JobProject>BlastProject</jsdl:JobProject>
    </jsdl:JobIdentification>
    <jsdl:Application>
      <jsdl:ApplicationName>BlastN</jsdl:ApplicationName>
      <jsdl:ApplicationVersion>2.2.13</jsdl:ApplicationVersion>
      <jsdl:Description>BlastN performs nucleotide similarity
         searching</jsdl:Description>
      <jsdl-posix:POSIXApplication>
        <jsdl-posix:Executable>/usr/local/bin/blastall
              </jsdl-posix:Executable>
        <jsdl-posix:Argument>-p</jsdl-posix:Argument>
        <jsdl-posix:Argument>blastn</jsdl-posix:Argument>
        <jsdl-posix:Argument>-d</jsdl-posix:Argument>
        <jsdl-posix:Argument>est</jsdl-posix:Argument>
        <jsdl-posix:Argument>-T</jsdl-posix:Argument>
        <jsdl-posix:Argument>T</jsdl-posix:Argument>
        <jsdl-posix:Input filesystemName="HOME">
              sequences1.txt</jsdl-posix:Input>
        <jsdl-posix:Output filesystemName="HOME">
              sequences1.html</jsdl-posix:Output>
        <jsdl-posix:Error filesystemName="HOME">
              sequences1.err</jsdl-posix:Error>
        <jsdl-posix:WorkingDirectory filesystemName="HOME">
              blastqueries</jsdl-posix:WorkingDirectory>
        <jsdl-posix:Environment name="PATH">
              /usr/bin:/usr/local/bin:/usr/local/bio/bin
              </jsdl-posix:Environment>
        <jsdl-posix:Environment name="TMPDIR" filesystemName="TMP"/>
        <jsdl-posix:WallTimeLimit>60</jsdl-posix:WallTimeLimit>
        <jsdl-posix:FileSizeLimit>1073741824</jsdl-posix:FileSizeLimit>
        <jsdl-posix:CoreDumpLimit>0</jsdl-posix:CoreDumpLimit>
        <jsdl-posix:DataSegmentLimit>32768
              </jsdl-posix:DataSegmentLimit>
        <jsdl-posix:LockedMemoryLimit>8388608
              </jsdl-posix:LockedMemoryLimit>
        <jsdl-posix:MemoryLimit>67108864</jsdl-posix:MemoryLimit>
        <jsdl-posix:OpenDescriptorsLimit>16
              </jsdl-posix:OpenDescriptorsLimit>
        <jsdl-posix:PipeSizeLimit>512</jsdl-posix:PipeSizeLimit>
        <jsdl-posix:StackSizeLimit>1048576</jsdl-posix:StackSizeLimit>
        <jsdl-posix:CPUTimeLimit>30</jsdl-posix:CPUTimeLimit>
        <jsdl-posix:ProcessCountLimit>8</jsdl-posix:ProcessCountLimit>
        <jsdl-posix:VirtualMemoryLimit>134217728
              </jsdl-posix:VirtualMemoryLimit>
        <jsdl-posix:ThreadCountLimit>8</jsdl-posix:ThreadCountLimit>
        <jsdl-posix:UserName>csmith</jsdl-posix:UserName>
        <jsdl-posix:GroupName>bio</jsdl-posix:GroupName>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:CandidateHosts>
        <jsdl:HostName>cluster1</jsdl:HostName>
```

```
      </jsdl:CandidateHosts>
      <jsdl:FileSystem name="TMP">
        <jsdl:FileSystemType>temporary</jsdl:FileSystemType>
        <jsdl:Description>Well-known 'name' for temporary space that
              does not necessarily persist after the job terminates.
         </jsdl:Description>
         <!--Mount point could also be defined in the initial JSDL
              Added here because this value may be needed to construct
              full paths for other elements.-->
        <jsdl:MountPoint>/tmp</jsdl:MountPoint>
        <jsdl:DiskSpace>
         <jsdl:LowerBoundedRange>10737418240.0</jsdl:LowerBoundedRange>
        </jsdl:DiskSpace>
      </jsdl:FileSystem>
      <jsdl:FileSystem name="HOME">
        <jsdl:FileSystemType>normal</jsdl:FileSystemType>
        <jsdl:Description>Chris's home directory</jsdl:Description>
         <!--Mount point could also be defined in the initial JSDL
              Added here because this value may be needed to construct
              full paths for other elements.-->
        <jsdl:MountPoint>/home/csmith</jsdl:MountPoint>
      </jsdl:FileSystem>
      <jsdl:ExclusiveExecution>true</jsdl:ExclusiveExecution>
      <jsdl:TotalCPUCount>
        <jsdl:Exact>1.0</jsdl:Exact>
      </jsdl:TotalCPUCount>
    </jsdl:Resources>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.txt</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:Source>
        <jsdl:URI>
           http://csmith.otherhost.com/blastqueries/sequences1.txt
        </jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.html</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:Target>
        <jsdl:URI>
           http://csmith.otherhost.com/blastqueries/sequences1.html
        </jsdl:URI>
      </jsdl:Target>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>blastqueries/sequences1.err</jsdl:FileName>
      <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
      <jsdl:CreationFlag>append</jsdl:CreationFlag>
      <jsdl:Target>
        <jsdl:URI>
           http://csmith.otherhost.com/blastqueries/sequences1.err
        </jsdl:URI>
      </jsdl:Target>
    </jsdl:DataStaging>
```

```
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

4.2 Other Notes

Older notes, gathered here for now.

~~Define the CDL document submitted to the CDL Deployment Service that will deploy the BLAST application onto the system.~~

~~<BLASTSystem>~~
~~<!-- Component wakes up in the locally specified installation space →~~
~~<!-- cd dir="$INSTALL_ROOT" />~~
~~<ant:untar archive="http;//host:port/blast.tar" />~~
~~<exportAtttributeValuePair atrribute="BLASTALL_EXE" value="$INSTALL_ROOT/blast-x.y.z/bin/blastall</exportAttributePair>~~
~~<!-- ASSUME SOMETHING WILL EXPAND & INSTER THE INSTALL_ROOT →~~
~~<!-- Advertise the populated JSDL:FileSystem (elements) so they can be pulled back from the CDL Deployment Service and inserted into thje JSDL document.~~
~~</BLASTSystem>~~

~~Deploy the BLAST application~~
~~CDL Document going in to the CDL Deployemnt Service~~

~~<BLASTSystem>~~
~~<!-- Component wakes up in the locally specified installation space →~~
~~<!-- cd dir="$INSTALL_ROOT" />~~
~~<ant:untar archive="http;//host:port/blast.tar" />~~
~~<exportAtttributeValuePair atrribute="BLASTALL_EXE" value="$INSTALL_ROOT/blast-x.y.z/bin/blastall</exportAttributePair>~~
~~<!-- ASSUME SOMETHING WILL EXPAND & INSTER THE INSTALL_ROOT →~~
~~<!-- Advertise the populated JSDL:FileSystem (elements) so they can be pulled back from the CDL Deployment Service and inserted into thje JSDL document.~~
~~</BLASTSystem>~~

~~<Server cdl:extends="d:Server"/>~~
~~<cmp:CodeBase>blast.tar</cmp:CodeBase>~~
~~<d:FileSystem>~~
~~<d:ID cdl:ref="/File/d:ID"/>~~
~~<d:Dir>data</d:Dir>~~
~~</d:FileSystem>~~
~~<d:Env name="ROOT" cdl:ref="/Server/RootDir"/>~~

~~<d:Env name="DATA">$ROOT/data</d:Env>~~
~~</Server>~~
~~<File cdl:extends="d:FileSystem">~~
~~<cmp:CodeBase>blastdb.tar</cmp:CodeBase>~~
~~</File>~~
~~</BLASTSystem>~~

~~CDL Document representing the deployed BLAST application~~

~~Completed JSDL document~~

~~BLASTSystem element~~

~~Components help deployment for a specific component for deployment through a particular API.~~

~~Deployment API services do not declare what they are (.NET/Unix). They do not declare what components exist.~~

~~Need to deploy 'Components' into the CDL system to enable the deployment of applications.~~

~~ACS: Binary, CDL document, Components needed to support CDL deployment~~

~~Common case component class – assume~~
~~What can be returned~~

~~Need codebase definitions~~
~~Need to better define the message body more tightly.~~