GWD-R (draft-ggf-cddlm-foundation8.doc)

Configuration description, Deployment and Lifecycle Management

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

Editors:

D. Bell, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, and P. Toft

January 28, 2003

Configuration Description, Deployment, and Lifecycle Management

Status of this Memo

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

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 maintaining their lifecycle management. This document, produced by the CDDLM working group within the Global Grid Forum (GGF), provides a high level overview of the CDDLM space by describing requirements, analyzing use cases, and by providing an overview of the related work within GGF, other standard bodies, as well as in industry and academia in general. The document sets the stage for the follow-up documents that will present in more detail the language, component model, and basic CDDLM services.

cddlm-wg@ggf.org

GLOBAL GRID FORUM

office@ggf.org
www.ggf.org

Full Copyright Notice

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

Table of Contents

List of Figures

List of Tables

1 Introduction

Deploying any complex, distributed service presents many challenges related to service configuration and management. These range from how to describe the precise, desired configuration of the service, to how we automatically and repeatably deploy, manage and then remove the service. Description challenges include how to represent the full range of service and resource elements, how to support service "templates", service composition, correctness checking, and so on. Deployment challenges include automation, correct sequencing of operations across distributed resources, service lifecycle management, clean service removal, security, and so on. Addressing these challenges is highly relevant to Grid computing at a number of levels, including configuring and deploying individual Grid Services, as well as composite systems made up of many co-operating Grid Services.

TBD Add here a short description of the relationship of CDDLM to other areas.

2 CDDLM-WG and the Purpose of this Document

The proposed WG (CDDLM) addresses how to: describe configuration of services; deploy them on the Grid; and manage their deployment lifecycle (instantiate, initiate, start, stop, restart, etc.). The intent of the WG is to gather researchers, developers, practitioners, and theoreticians in the areas of services and application configuration, deployment, and deployment life-cycle management and to explore the community need for a broader effort in this area. The target of the CDDLM WG is to come up with the following four specifications.

1) Configuration Description Language Specification
2) Intermediate Configuration Description Language Specification
   (Configuration expressed in 1. with inheritance, dependencies, and variables resolved)
3) Basic Services Specification
4) Component Model Specification (component representation and implementation)

The purpose of this document is the following:

1) Gather the CDDLM WG members behind the same agenda, with the same set of understanding and expectation for the forthcoming deliverables.
2) Offer to the community a preview of what the WG will deliver and enable them to provide to us some early feedback.
3) Set a stage for collaboration with other RG/WG within and outside GGF.

Deploying any complex, distributed service presents many challenges related to service configuration and management. These range from how to describe the precise, desired configuration of the service to how to automatically and repeatably deploy and remove the service onto and from the resources available. Description challenges include how to represent the full range of service and resource elements, how to support service "templates", service composition, correctness checking, and so on. Deployment challenges include automation, correct sequencing of operations across distributed resources, service lifecycle management, clean service removal, security, and so on. Addressing these challenges is highly relevant to Grid computing at a number of levels ─ from configuring services that comprise the Grid infrastructure, up to describing and deploying Grid applications themselves.

Hence, the proposed WG (CDDLM) will address how to: describe configuration of services; deploy them on the Grid; and manage their deployment lifecycle (instantiate, initiate, start, stop,

restart, etc.). The intent of the WG is to gather researchers, developers, practitioners, and theoreticians in the areas of services and application configuration, deployment, and deployment life-cycle management and to explore the community need for a broader effort in this area.

The deployment can be positioned at a part of provisioning cycle that optimizes data center resource allocation and configuration to the changing system environment or usage load from time to time. The provisioning is a conceptual cyclic process consists of several phases. Those phases can be categorized into 1) Execution and Monitoring, 2) Analysis and Projection, 3) Resource allocation Planning, 4) Deployment. The provisioning manages various layers of the target systems from hardware to application layer. The application layer includes components such as application program, content data or DB schema for specific application. The infrastructure layer includes middleware component such as Web Server, Application server or DBMS, Operating systems or hardware components such as server, storage, firewall, load balancer or network switch.

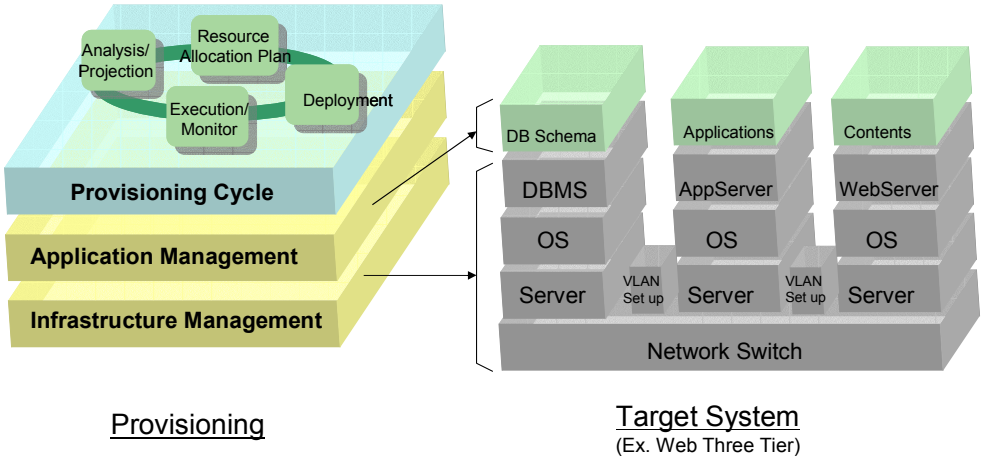

Figure 1 Provisioning Layers

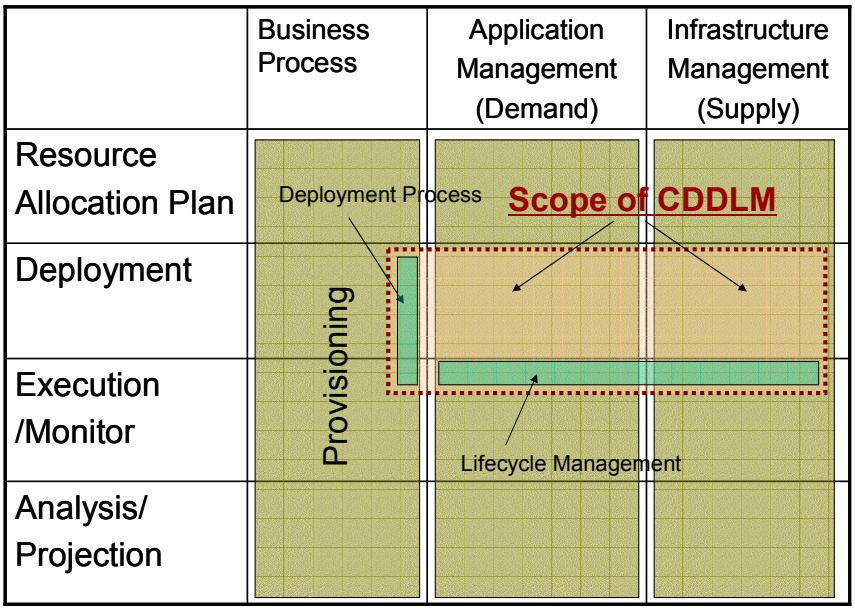

Table 1 Scope of CDDLM

Table shows the scope of CDDLM. It covers the deployment phase for both application and infrastructure layers. It also includes rudimentary process management covering the deployment and lifecycle management of the deployed software running on top of the target environment.

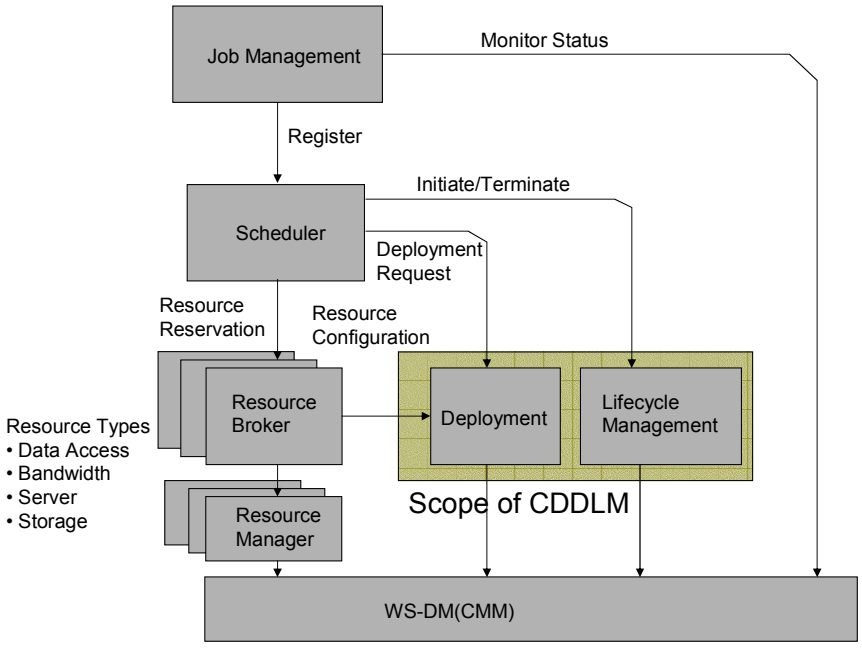

Figure 2 Use case in Program Execution Environment

Figure describes the assumed use case of CDDLM in program execution environment. The deployment and lifecycle management services are triggered by either the scheduler or by manual or automated job management. Configuration has to be determined by outside of CDDLM by

services such as a resource broker and given as a set of component description. CDDLM assumes manipulation mechanism of respective target components through WS-DM interface.

3 Functionality Requirements

3.1 Basic Functionality Requirements

- Describing service configuration: the CDDLM specification will describe how to represent the configuration of distributed services. The description will include:
  - A means to specify the components of a service – i.e. what elements combine to make up a given service
  - A means to describe the configuration data relevant to each of the service components
  - A means to link together configuration data elements – for example, this might be used to specify the name of a required in host in one place only, but use it consistently throughout the description
  - A means to extend/modify service descriptions by overwriting or extending selected configuration data elements
  - A means to combine service descriptions – for example, allowing a composite service to be specified from a combination of component service descriptions
- Deploying services: CDDLM will specify the mechanisms by which service descriptions are deployed to become running services. The deployment mechanism will include:
  - A means to interpret service descriptions in order to determine which service components need to be deployed
  - A means for loading service components onto the correct resources and for supplying them with their configuration data
  - A means to allow services to be un-deployed from their resources
  - A means to allow service components to interface with the deployment mechanism in order to deploy and un-deploy their own services
  - A means to deploy and manage many services simultaneously across the same set of resources
- Service Lifecycle Management: CDDLM will specify the behavior required from service components such that they can be managed by the deployment mechanism. This includes:
  - A specified component lifecycle model with well-defined component states
  - Specified component entry points to correspond to lifecycle state transitions
  - A means to determine the lifecycle state of each component
  - A means to discover all deployed components via a management interface

3.2 Security Requirements

- The deployment system must offer security properties such that:

  - Deployed services should be no less secure against external attack than if they had been installed manually onto resources. Automated deployment should not introduce additional security vulnerabilities.
  - Deployments can only be made from trusted sources onto trusted target resources
  - Management communication within the deployment system is restricted to that between mutually trusted resources and service components.
  - Authorized users can remotely submit requests to the service, particularly from behind firewalls.
  - Having the right to deploy an application cannot imply unrestricted access to the application or its data.
  - The application and data may be sensitive –it could contain proprietary algorithms or critical data, such as credit card information. This implies that the mechanism used to get this information to the target hosts must be adequately secure.
- The security framework adopted must be fully consistent with, and will leverage, the standard Grid security mechanisms (GSI)
- In the first instance we assume only a single level of trust across a deployed service (or indeed a set of deployed services), such that a resource or service component is either fully trusted or fully untrusted.

3.3 Resource Management Requirements

While the CDDLM system does not directly manage resources, it needs to know the logical and physical descriptions of the resources asked for by the user and granted by the resource manager. This information is used to determine where components in the application are to be deployed.

- Resource description
- Parameterization
- Relationships
- Composition
- Inheritance
- Run-time binding
- More?

3.4 System Properties Requirements (Non-Functional Requirements)

The CDDLM and in particular basic services need to adhere to the following non-functional requirements:

- **Scalability.** CDDLM should impose no or minimal limits on the scalability. Scalability should be limited by the underlying network infrastructure and the transport protocols it supports.. One exception is run-time binding which can slow down the deployment due to the resolving lazy dependencies.
- **High-availability.** CDDLM should be highly available to the requests for deployment, in particular, it should respond well to high load of requests and to failures.

- **Self-healing.** CDDLM should be able to heal its services in cases of failures, such as restart its own infrastructure if parts fail or redirect requests to other replicated CDDLM basic services.
- **Disaster recovery.** CDDLM should be able to function and recover even in the presence of catastrophic failures. This means at minimum support for replication of the services and possibly more complex support for recovery if significant parts of the deployment services failed themselves.
- **Full-automation.** CDDLM should be fully automated, without the need for user input, unless desired. In particular, it should be possible to deploy other services in an automated way, to recover from failures, or to satisfy provisioning requests in an automated way. CDDLM basic services should accomplish this at different levels of time granularity, starting from short term requests to long-term execution. It must also be possible for deployment to be initiated programmatically, such as during an automated build process.
- **Usable**. The system must be simple enough to be usable by end users without deep knowledge of the system. Both the language and the management processes must be relatively easy to learn and use.
- More?

3.5 Configuration Description Language (CDL) Requirements

3.5.1 Declarative Description

The configuration description should be declarative: it should not be a sequence of operations but a set of declarations that describes dependency between resources. The declarative description should provide enough information for the CDDLM implementation to dynamically generate correct sequencing of operations across distributed resources for deployment and life cycle management.

3.5.2 XML-based

The language should be XML-based. A well-formed description should be a well-formed XML document. It should also be in the subset of XML that is Web Service friendly and broadly supported across popular platforms. Specifically, XML Schema (cite: WXS) should be the preferred description of the syntax of the language, and yet troublesome parts of that format (facets, unsigned longs), should be avoided where possible.

The XML Format should also be namespace aware: each version of the CDL will define its own namespace for elements and attributes. CDL implementations must expect to handle nested data in other namespaces, where appropriate. The language must also be designed for interpretation in a secure context, in which entry expansion and remote schema/file inclusion are permitted, but only in a very controlled manner.

3.5.3 Modularity

A service deployed by CDDLM may consist of multiple subsystems. CDL should be able to represent each subsystem in a modular way.

The description should include

- configuration of each subsystem (configuration parameter definition)

- dependency between subsystems (in terms of configuration parameters and life cycle status)

3.5.4 Dynamic Configuration

CDL should be able to be applied to dynamic configuration use cases where:

- some configuration parameter values can not be determined before deployment time
- configuration may change at run time
  - changes made by environments (e.g., system failure)
  - changes made by the CDDLM implementation

3.5.5 Consistency

CDL should be able to specify dependencies between configuration parameter values so that a CDDLM implementation can manage consistency between parameters.

For dependency description, CDL should support:

- assignment of value derived from other configuration parameters.
- integrity constraints (or assertions) that values should satisfy

(note: this means that propagation of value/change is one way)

Parameter values should be able to be assigned and validated

- at definition time (binding time)
- at deployment time
- at run time (configuration change)

CDL should support (both/either?) strong consistency and weak consistency:

- strong consistency: the CDDLM implementation can set/modify a particular set of parameter values in a transactional (ACID) way.
- weak consistency: Configuration changes are correctly propagated so that the system will become consistent again

CDL should enable the CDDLM implementation to detect unresolvable inconsistency.

3.5.6 Integrity

The CDDLM implementation should manage integrity on life cycle states (instantiated, initialized, running, etc) of multiple components. CDL should ensure that the CDDLM implementation correctly generates sequences of operations to deploy, start, recover, stop, and remove the service.

- The CDDLM implementation can confirm that required configuration has been correctly done before changing the state of a component (e.g., component A should be running before B is initialized).
- When the CDDLM implementation detect that the service is not deployable during deployment operations, it can safely cancel the operations (e.g., remove components already deployed).

- When a component state is changed by environment (e.g., a component is failed due to system failure) and, as a result, integrity is broken, the CDDL implementation can generate a sequence of operations to recover integrity (e.g., restart components in correct order).

3.5.7 Composability

The total service configuration description should be able to be composed by combining multiple descriptions that may be provided by multiple configuration vendors. CDL should provide a way to define a new composed description by referring to existing descriptions. For example, the user wants to customize a configuration description provided by system vendors by adding/overriding configuration parameters and/or configuration dependencies.

3.5.8 Discovery (Reference, Binding)

CDL should support

- reference to external configuration descriptions
- reference to components (packages to be deployed)
- reference to services that provides deployment functionality of components

Standard discovery/binding mechanisms should be applied to resolve these references. In the context of Web Services, UDDI is invariably touted as the way to discover components. However there is no requirement that this is the case. UPnP1.0 uses its own subnet-scoped multicast discovery protocol before sending SOAP0.9 messages around, and the J2EE1.4 specification uses LDAP to register and locate services. There is some appeal in supporting both LDAP and UDDI servers.

The OGSA architecture uses Grid Service Handles as location independent references to service instances –these are resolved on request to an instance of a service. The acquisition and resolution of these handles should be part of the discovery mechanism. For example, an application could somehow discover the Grid Service Handle of other components in the system; resolution of these handles would return URLs to the endpoints they offer.

3.5.9 Security

The security requirement of CDDLM should be achieved by incorporating Web service/Grid/XML security standards into CDL.

3.5.10 Extensibility

CDL should allow the user to add extensibility elements in a description. For example, the user may need to add security and policy descriptions.

3.6 Service Interface Requirements

Communication to the CDDLM service will be via a Web Service/Grid Service SOAP interface. Here are the requirements of this service interface.

- Supports remote access through firewalls
- Integrates with the GSI security model
- Messages shall not be listened to, spoofed, replayed or vulnerable to other well known attacks.
- Messages should be idempotent.

- Status information shall be provided in machine readable form
- Operations to deploy/undeploy shall be supported
- Interfaces shall be extensible (language, maybe actions)
- There will be enough functionality for a service to deploy via another service (chaining)
- Notifications of lifecycle events/status changes will be propagated
- Upload of the files to the remote system will be supported.

3.7 Other Functionality Requirements

TBD

4 Use Cases (Kojo, Stuart, Dejan, David?)

4.1 Grid Services (Kojo)

4.1.1 Summary

Grid Services are abstract model of virtually any kind of distributed resources in grid environment. The resource may be actual working application program, middleware, operating system or any kind of hardware such as servers, storage, firewall, load balancer or hub switch. In the application development context, deployment may include several actions including defining the configuration of the application and servers to be deployed, gathering all contents to be deployed, transferring the contents to the hosting environment in appropriate sequence, then initiate middleware and the application. In the provisioning context, deployment may include actions, on top of application deployment, such as defining the system configuration including network geography or access policy for firewall or any other hardware setting, downloading and setting up all those defined set ups in appropriate sequence, and enforce the setting.

4.1.2 Users

Since the Grid Services are abstracted model, one can consider whole spectrum of users of grid computing. In this particular section, we consider Large scale Application Development and QA Scenario. The users are application development engineers or QA engineers. Data center operators are potentially another type of users in the Data Center Provisioning Scenario. This scenario will be examined in the following section.

During the application development and QA of the large scale system scenario, the engineers repeatedly update and download the application program to appropriately configured hardware, operating systems, middleware and network setting. They also want to examine the application with many different configurations, so that they can guarantee correct execution in many different environments which are anticipated in the future use.

In typical application development or QA department, they have to deal with number of applications for long time. They need consistent and reusable means of defining all those different configurations so that they can reuse the definitions previously used in other systems and they need to define only the unique portion of the application. This feature is also useful when they want to examine many different configurations which have a lot of commonality as well.

4.1.3 Scenario

1) Defining Configuration

The engineer first defines configuration of application and system using a tool with graphical user interface or textual definition language. They can refer other definitions done in the past or other places. They can extend or modify existing definitions and derive new definitions. This increase productivity and eliminate new errors that might be introduced in the definitions. They also have to define appropriate procedure how the system can be configured appropriately and the application can be transferred and initiated.

2) Writing or modifying Application Program

This is more like conventional software development. The engineer can write and compile virtually any application with any development environment they are accustomed with.

3) Set up system configuration

Once the engineer completed initial coding, the hosting environment should be configured appropriately with the hardware, operating system and middleware. The entire process of setting up the configuration is done automatically when the engineer initiates the deployment.

4) Transferring the Program and Testing

After the system is appropriately set up, the system automatically starts transferring the application program. According to the process definition, the system organizes sequence of the transfer and initiation of the software components.

The engineer wants to manage partial transferring of the contents so that they can avoid long time for transferring the unchanged or unnecessary portion of the contents for the following phases.

5) Testing another configuration

The engineers may want to change the configuration in various reasons. During the debugging, they may find some errors on the configuration and fix them. They also want to examine the allocation with different configurations. By specifying new configuration, the system can change the configuration automatically in appropriate sequence of the actions.

6) Operation support and maintenance

Because the configuration definition is compliant to the standard, it should be compatible with other company's environment. It can be handed over to the customer or operation support or maintenance company along with the application code, so that the application can be seamlessly transferred to the companies and supported and maintained by other companies.

4.2 Commercial Data Center

There is a possibility that we eventually consolidate 4.2-4.4, or a subset of these subsections.

Summary

Today's commercial data center provides various types of hosting services. They can be classified into four categories: Shared, Collocation, Managed Infrastructure and Application hosting. A shared service is a service that provides low-cost, entry-level Web hosting services. Instead of requiring a separate computer for each user Web site, dozens of sites co-reside on the same physical server. With collocation service, users lease racks, cages, or cabinets from the service provider; put their own servers in the data centers; and send their own technicians in to perform maintenance and respond to alarms. A managed infrastructure service is a dedicated hosting service that includes Web, database, and applications servers, server configurations, systems operations, security, monitoring, reporting, and network. An application hosting service is in an application hosting environment, the data center –utilizing dedicated servers and applications –

takes responsibility for all day-to-day operations and maintenance of applications and the underlying infrastructure.

The deployment of software, contents and set up of the system configuration plays a critical role in the managed infrastructure or application hosting services. Upon the user's requests, the data center provides an appropriate system configuration for both the hardware and software. The hardware configuration includes the configuration of the load balancer, firewall or VLAN switches so that the user can have a safe isolated network partition appropriately protected from/connected with the outside of the data center. The software configuration should include operating system and middleware. It also includes application software if it is an application hosting service.

Automated deployment is important to the data center business at least following three aspects. In today's data center, it take some time –a matter of weeks- from the users request to complete deployment and set up of all the equipment and software. The lead time is crucial to both the users and the data center because it directly affects its resource utilization. The accuracy of the deployment is also critical to the data center business. Today's typical data center deployment processes require a lot of manual operation, operations with plenty of opportunity for mistakes that can lead to a misconfigured service. The data center has to spend significant time testing the new configuration. Security is another critical issue for the business. The software and date must not be only protected from outside of the data center, but also, in some cases, from the data center operators, because it contains information confidential to the data center user.

Customers

The data center operators deal with the deployment in two aspects. When they receive a new system configuration request, they interpret the user's request into configuration definition which becomes the input of deployment service. The number of the deployment components varies from a few tens to thousands, depending on the range of the deployment. When it is ready, the software, data and hardware configurations are deployed to the target environment. This may be done either with operator's manual operation or automated operation by administration system.

Scenarios

Configuration Definition Scenario

The user talks to a data center engineer about their requirement on their system configuration. The performance, reliability or availability requirements are interpreted into specific system configurations such as types of server hardware, data storage or bandwidth between servers or channel to outside data center. It also derives requests for specific configurations for high reliability such as duplex servers or RAID of storage, or requests of array of servers for performance scale out.

Software requirements include a type and version of operating systems and middleware such as web server, application server or DBMS on which the applications sit on. The user also specifies the initial set of data contents of the web server, DBMS. The contents information includes initial set up data for the middleware or user registry for the middleware. Some of the contents may be confidential to the data center operators.

Those requirements are interpreted into a set of configuration definition by the data center engineer by either graphical user interface for the definition or textual definition language.

In the near future before the fully automated utility computing vision is realized, the user may require a set of versions of the configuration for particular application so that it can switch between the configurations depends on necessary system operation scenarios such as normal operation and system testing, or busy time configuration and minimum load configuration.

Deployment Scenario

Based on the configuration definition, the data center operator set up the configuration and deploys the software and contents on the configured hardware. Some of the software such as operating system or middleware may be pre-installed with the configured servers and can avoid installations.

The operator has to allocate required hardware available to the deployment. This may be done manually by the operator or automatically by administration system.
The operator sets up the hardware and deploys all the necessary software on the hardware based on the configuration definition. This operation is supposed to be fully automated. All the application software and contents owned by the user has to be transferred to the data center or the place reachable from the data center before the deployment. Deployment should be done at appropriate timing either triggered by the operator or automatically done by setting up timer.
Involved resources
Types of the resources involved in the deployment depend on type of services provided by the data center. With the minimum scenario, the deployment system deals with only application software onto servers. With this type of scenario, it controls versions of application software, initial parameters or data required by the software. The medium range scenario includes deployment of various types of infrastructure software such as web server, application server or DBMS. Most of those scenarios require initial data or user registry deployment along with the software. The full range scenario includes various types of hardware set ups such as servers, storage, load balancer, firewall or network switches.
Functional requirements for OGSA platform
Discovery and brokering: The scenario assumes a set of hardware ready for deployment allocated by appropriate means before the timing. Discovery and brokering of the hardware resources is out of the scope of the deployment.
Deployment: This use case is specifically regarding with the deployment.
Virtual organizations: Deployment is a part of the process to establish a virtual organization required to an application.
Policy: Deployment should have a set of policies for particular events such as system fault and recovery, execution errors or other exceptional situations.
OGSA platform services utilization
Service Interaction Services
The deployment service in this scenario assumes services of the service interaction including

- Virtual organizations
- Service group and discovery services
- workflow
- Transaction

1) Service Management Services

   The deployment service in this scenario is a part of this category of the services.

2) Service Communication Services

   The service assumes this category of services especially messaging service.

3) Security Services

   The service assumes full range of security services.

4) Program Execution Services

   The service utilized by this category of the services.

5) Resource Management Services

   The service implement some functionalities of this category of services.

4.2.1 Security considerations

The deployment service in this scenario is to configure secure environment for particular application execution appropriately isolated and protected from outside of data center or other user's systems in the data center.

The deployment should guarantee end to end security from storage of the Configuration Definition, Deployable software and contents, transferring all those software and contents to the Deployed software and contents.

The data regarding the deployment should be secured from outside the data center and from other data center users and their systems. Some of the contents are confidential to the user company who requires that even a data center operator cannot access the data.

4.2.2 Performance considerations

The system should consider various aspects of the performance which includes,

1) Scalability in terms of number of defined components, servers and other hardware devices
2) Compiling or Interpreting speed of the component description
3) Deployment speed regarding with the size and number of components or number of servers and other hardware devices
4) Speed of the operations regarding with lifecycle management

4.2.3 Use case situation analysis

4.2.4 References

4.3 IT Infrastructure and Management

4.3.1 Summary

Within a large Enterprise, there are typically distinct groups, Operations and IT, that are separately tasked to manage data center operations versus end-user computational resources including departmental or workgroup servers. In smaller companies, these services are often combined in a single functional group. This usage case focuses on the domain of IT and in some cases its broader interface to services provided by the data center such as defined earlier in the data center usage case.

In widescale use currently, Systems Management software is used to address the concerns of IT typically in the areas of: Configuration Management, Software Distribution, Asset Management, License Management, and Help Desk. In the realm of CDDLM, we will focus primarily on Configuration Management and Software Distribution, though it is understood that much of the remaining functionality will be serviced elsewhere in the Grid.

In a typical usage pattern, an end user will need to connect to some representative Grid Service. First, the user must have access to the correct software for interaction with the service. This may imply the need to deploy dependent components to the end user's workstation a priori, or in the event of use of a particular service. Also, the Grid service may be one of several configurations of services. It may be housed on a defined server or set of servers, a mainframe, or supercomputer. It may also be run on an available hive of workstations in a workgroup. In any of these scenarios, it will be important to ensure that the correct software is deployed, configured and managed through the duration of the Grid service. These instantiations must also be managed within the larger context of the Grid, with multiple jobs spanning LAN and WAN over multiple compute clusters.

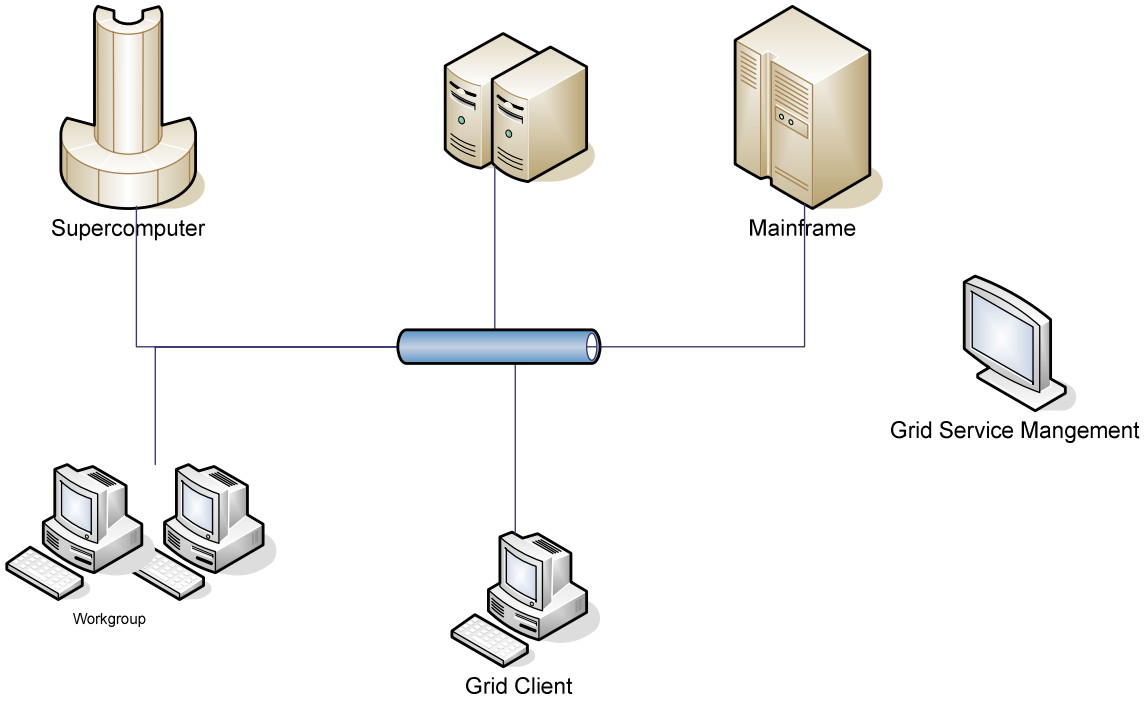

Figure 3 IT Infrastructure and Management

The role of IT will be to ensure that end user's needs are met. They will assume the responsibility of the deployment or coordinating that deployment with the Operations staff in the data center. Even if the deployment is primarily done in the data center, IT will need to access the configuration, deployment, and event information.

4.3.2 Customers

The ultimate customer in this use case is the corporate knowledge worker consuming the physical service itself. The corporate IT and operations staff are also participants. And in many Enterprises, the software development team may play a role in the deployment and configuration of services for the customer.

Knowledge workers on a daily basis will interact with abstract services on the Grid in order to perform their industry specific tasks. These services may be batch oriented, or real-time submissions of commands to be processed. The user will require that a physical execution environment be created for their service, and software configured and deployed. During the execution, the end user will monitor the progress of the execution, or await its final disposition.

In the configuration of most Enterprises, these usage scenarios will mostly be within a single physical site. However, access to central or remote repositories is often needed for access to data or specialized computational resources such as mainframes or dedicated compute clusters. The number of users participating in this scenario can be quite substantial, typically several thousand users per office facility or site, and tens or hundreds of sites around the world. Additionally, there are typically one or more central data center sites where large computation facilities are housed.

4.3.3 Scenarios

Generalized Grid Service – A currency trader at a global investment bank has thought of a new model for evaluating intra-day pricing fluctuations. He discusses this model with the Engineering group and requests a Grid service be developed. The service is designed and its requirements are

discussed with the Operations team. It is decided that the service will live in the data center and be placed in a dedicated pool of servers that have access to the historic database of global currency prices.

The trader awaits the availability of the service. The Engineering team builds a Web-based user interface for allowing the trader to create instances of the service with different configurations. He is also given screens for submitting different types of queries with different parameters and formulas. However, there is no requirement other than a general purpose web browser for his own personal machine.

End User Device Deployment – The currency trader, happy with his model's performance, decides that he needs to do some OLAP analysis of the results sets from his service runs. The Engineering team suggests that they build a Visual Basic application to front-end his Grid service. As his bank provides floating desktops, he needs to ensure that he will always have access to his service.

As part of the development of the service, he asks the Engineering team to model and configure the application as a service itself that can be deployed as needed, to whatever workstation he may be using. In this manner, when he invokes the service from the Web, his workstation will be configured with the OLAP application.

High End P2P Computation Service - The currency trader decides that his model is perfect for his extended length trades, but does not help him during daily trading activities. He then comes up with a modification of his model that does not require any historic data, but requires a lot of instantaneous compute power. He discusses this with his supervisor, who suggests using the team's extra workstations that are not in use during his trading period.

The Engineering team discusses how to modify his Grid service to be deployed on several machines at once. They propose the solution to Operations, who forwards the request to the IT staff. The IT staff and Operations team discuss how to bring the team's workstations under management of the data center's resource manager, or if it is more feasible to build a separate resource pool.

The team decides to utilize a local resource manager, able to only dedicate resources from the currency trading team's workstations. At the time the trader needs to run a simulation, the resource manager allocates available workstations, or partitions of the workstations, and invokes the deployment and activation of the service.

4.3.4 Involved resources

In these scenarios, the Grid system is responsible for managing a fairly large and diverse set of resources. In the data center several different hardware systems are used for storage, query, and computation. However, the system is primarily responsible for ensuring the correct configuration of the Grid service components. It must validate the availability and/or connectivity to other components such as the storage system, but it is often presumed that the historical market data is preserved on a stable storage facility.

The Grid service must be deployed and managed through its lifecycle and the end-user software components managed for any of the scenarios touching a user's desktop. In the case of the peer-to-peer system, the entirety of the peer workstation must be managed from operating system through application software and the Grid service lifetime.

By their nature, the components are highly distributed. They are often on various network segments and often geographically disperse.

4.3.5 Functional requirements for OGSA platform

Multiple Security Infrastructures: The services described here can utilize more than one disparate system that may have its own or legacy methods of security, differing from the access methods to the Grid service itself.

Resource Virtualization: In order to allow the Operations team to manage a pool of servers, it must be presumed that the Grid services used here are not attached to dedicated resources. In addition, the P2P usage scenario outlines the need for dynamic provisioning.

Provisioning: These usage cases are targeted toward the deployment portion of provisioning.

Metering and Accounting: It is often necessary in this type of environment for IT to be able to charge back their services to the end user on different types of accounting bases.

Monitoring: As short-lived and long-lived services exist in the infrastructure, it is important for IT to be able to ensure that service levels are met and that they have a means to troubleshoot any problems with the Grid service. Banking environments demand swift resolution to problems, so as to avoid money losing situations through technology outages.

Disaster Recovery: In most global trading environments, disaster recovery is a basic prerequisite for any service to become operational.

Self-Healing: This type of functionality is a nice to have.

Legacy Application Management: It will be commonplace for Grid services in this environment to interact with legacy applications. The Grid deployment system must be able to communicate with legacy management systems to ensure deployments are successful and accurately report on failures.

Administration: The IT and Operations groups must be able to simply manage the process of deployment and report on its ongoing status. There must be significant abilities to manage different configurations running within the same infrastructure and validate their correctness and compliance with set system policies.

Policy: It must be possible for Operations or IT to codify and validate the enforcement of policy within the Grid service during deployment all the way through the destruction or end of a service lifetime.

4.3.6 OGSA platform services utilization

Service Interaction Services: It is understood that many services may coexist and interact with each other within a shared infrastructure as defined above. In this case, there must be provisions for service to discover and coordinate with one another.

Security Services: Within an environment defined here, security of information is a key component to the operation of a service. The service must be able to fit within the security model defined by its host environment.

Data Services: It is presumed, as shown above, that interaction with various data sets will be an integral part of the operations of Grid services within these and other scenarios.

Program Execution Services: The use of a Program Execution Grid is a prerequisite for enabling the P2P scenario outlined above in order to allocate the VO resources of the currency trading team.

Resource Management Services: The current scenarios define the implementation of several services that are part of this category of OGSA platform services.

4.3.7 Security considerations

The basic premise of CDDLM is to deploy and configure services within the Grid. There are two basic security issues that must be considered. First, the deployment of the service itself must not violate any security policies of its host network. Second, the deployed service configuration must be as secure as its defined installation policy implies.

CDDLM Service

Many corporate servers and services involve setup that requires root level privileges in order to properly configure the service. Also, many servers such as a web server require a password or keyed encryption certificate. The services responsible for configuration of the environment will potentially have access to these passwords and privileges. It is important for the implemented deployment services to have several properties to ensure their end-to-end integrity.

Attack Vulnerability: The deployment services must be hardened to ensure that they are not susceptible to common attacks such as buffer overruns. If they are to have the ability to gain privileged account access, they must be prevented from deferring this right to an attacker.

Sensitive Data Handling: It is important that the deployment services use sound principles to ensure the integrity and security of sensitive data. Data such as passwords should be never stored in the clear in any type of memory. If they must be decrypted, they should be in the clear for as short a duration as possible. Code guards should be put in place to protect the algorithms, modules and processes that use any sensitive data. Additionally, use of digital signatures and secure key exchanges should be used to protect the integrity of stored and transmitted data.

Trust and Privileges: In order to perform installations at elevated privilege, the deployment services may need trusts or privileges of their own. It is important that any trust that is given be done authentically, and any privileges granted be as granular to the task as is possible.

Policies: Commonly, services such as the deployment services, are given exceptions to standard network policies due to the tight integration requirements they possess. This should be avoided at all costs, and strict maintenance of system wide policies should be enforced.

Configured Services

In corporate IT, many systems are or should be prevented from co-existing with other services due to the sensitive nature of data. During the configuration of a service, the deployment services must ensure that system policies are obeyed not just for the service itself, but for the entire network of services. Thus, the deployment services must keep in mind that the information concerning configuring one service may not be enough information to consider in a Grid of co-existing services.

4.3.8 Performance considerations

The performance impact of deployment within each defined scenario is different. In the Generalized Grid Service or End User Device scenario, the deployment is done statically, prior to the use of the service. In this case, the primary performance issues are scalability and relative deployment speed.

The system must be able to scale to consider the ability to address a significant number of physical devices or endpoints, as well as a reasonable number of configuration and deployment dependencies and process steps. In doing so, the deployment must complete within a reasonable time period in order for the system to be usable in practice. In the case of a large Enterprise, deployments must be able to be parallelized and/or broadcast to meet the potential scale.

In the case of the peer-to-peer distributed computation, the instantiation time of the service cannot be appreciable. It can only be a fraction of the total processor and elapsed time of running the service. Otherwise, it makes no sense to deploy the service in this manner. Thus, the deployment service must be able to fit within some framework of cost analysis or deployment scheme decision.

4.3.9 Usecase situation analysis

Many of the background services required to build on top have been considered to this point, such as resource management services. However, it is unclear what actually exists and to what extent.

4.3.10 References

4.4 Utility Computing Model

4.4.1 Summary

The utility computing gained a lot of visibility lately. It is a model where the computing resources are "wired-once, provisioned-many times" in support of better utilization of a data center (consolidation) and treated like any utility. The primary benefits are in reducing the costs: supporting many more systems with same number of IT, in the same physical space, with same power dissipation, etc. In this model, it is essential to be able to automatically and rapidly install systems (repartition, provision, install apps and services, manage, etc.).

One way to relate utility computing to deployment is similar to the needs of other architectures, such as PlanetLab, virtualized machines, or Grids in general. On all these architectures, there is a need to acquire a subset of resources, to deploy required software and to continue to provision resources if the demand fluctuates. Figure 4 presents a utility computing model.

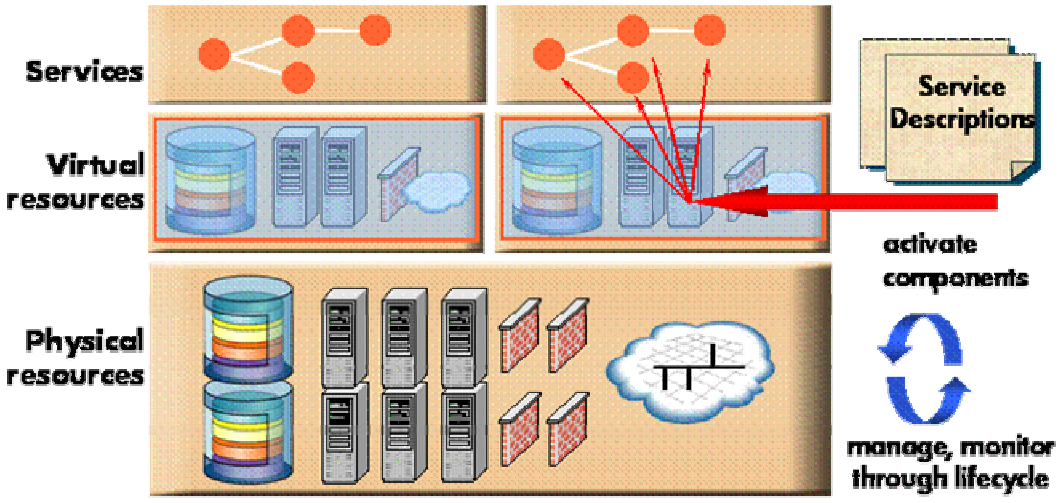

Figure 4 Utility Computing Model.

Based on Virtualization it Enables Consolidation of Data Center Resources. Automatic Configuration, Deployment, and LifeCycle Management Play Critical Role.

4.4.2 Customers

The customers for the CDDLM in the utility computing model consist of the following participants with the different level of sophistication and control:

- **The utility administrators** are probably most sophisticated customers, who have most control over the utility and the use of CDDLM. Owners of the utility are able to repartition the parts of the utility and re-allocate it among different service providers. They can also wipe out these partitions after they are used by some of the customers and install on them any of the default or customized installations. In addition, utility owners can use CDDLM for administrative purposes for managing utility–specific services.
- **The Service Providers (SP)** lease or own parts of the utility in order to provide services within partitions. Within allocated partitions, SPs install required services that they provide to users. As the load fluctuates, the SPs can lease more or less of the resources, on which they will install and deploy the required services.

The service users have the minimal access to CDDLM (compared SPs & utility owners) and it is typically an ability to extend service on demand with additional components, to change configuration parameters, etc. Most of this functionality is enabled by SPs and invoked by service users.

4.4.3 Scenarios

Support for Dynamic Provisioning of a Utility Data Center

The service provider has experienced increasing load based on the popularity of her service. Fortunately, the SP has an agreement with the owner of the computing utility to expand the capacity on demand. As the load crosses the threshold, additional server is requested from the pool of available resources; and within minutes, it is installed with the requested system and application software. After some time, the interest in the same service reduces below a lower threshold and extra resources become redundant. Using CDDLM, the resources are released and wiped out by the CDDLM in the matter of minutes.

In this scenario, CDDLM relies on other provisioning tools for allocating resources, but it takes care of configuring the services, deploying them and them un-deploying them as necessary.

Support for Dynamic Provisioning across Utility Data Centers

A service provider has installations across the world. As his service is becoming popular and the prices oscillate across the world, he has realized that he can save a lot of money by moving the load to less utilized sites. Upon the increase of the load in Europe, he is able to deploy some more services in America without significantly compromising the response time of average users. Only a subset of requests that would violate SLAs and would normally trigger on-demand allocation of premium resources locally are migrated to a remote site where the same service is deployed on demand. Using CDDLM, the resources are deployed and released across geographic locations on-demand in accordance with the best interest of the service provider.

In this scenario, CDDLM relies on other Grid provisioning tools for allocating resources, but it takes care of configuring the services, deploying them and them un-deploying them as necessary.

Development of Service

A company is deploying a large complex service that relies on a number of external services. While each individual service is reliable, the newly deployed service compound may not be initially. (The same scenario can be imagined during initial service development, while the code is not yet stable.) The utility is partitioned for a beta-testing configuration and particular care is

paid to the services that execute there. They are extra monitored and if any service fails, it is redeployed, along with any other services that depended on that service. As the service becomes more stable, the monitoring continues, but with less frequent heartbeat, replication, and other reliability support.

In this scenario, the lifecycle management accounts not only for planned events, but also for the unplanned failures. In such a case, the service is re-deployed after it has crashed making sure that all dependencies are respected: service depending on the dialed service are also restarted.

4.4.4 Involved resources

In this scenario, CDDLM is responsible for managing a unified set of utility computing resources. The primary benefit of using a Grid standard for this kind of resources is in inter-utility submissions, in which case, the services can be deployed across utilities. In this case, the distributed utilities are likely to be geographically dispersed, even on different continents.

4.4.5 Functional requirements for OGSA platform

- **Discovery.** Requests for the deployment are submitted to a well-known service that needs to be discovered first.
- **Authentication, Authorization, and Accounting (AAA).** The requests for service deployment have to be authenticated, authorized and accounted for.
- **Resource allocation.** It is assumption that the CDDLM will contribute to the overall provisioning solution by means of deployment, however resource allocation is expected to be provided separately.
- **Monitoring.** Lifecycle management relies on the monitoring service for maintaining status of the service health.
- **Fault Handling.** Once a failure is discovered through monitoring, the CDDLM interacts with the fault handling modules in order to redeploy the service.
- **Policy.** Functionality such as provisioning, usage, fault handling, etc. are specified using policy.

4.4.6 OGSA platform services utilization

- **Name resolution and discovery service**

  This service is used for the Grid as discovery functionality.

- **Security service**

  This service is necessary for OGSA AAA functionality.

- **Provisioning and resource management service**

  This service is used for provisioning.

- **Fault handling service**

  This service is used for fault handling.

- **Policy service**

  This service is used for policy-related functionality.

- **Monitoring service**

This service is used for monitoring functionality.

4.4.7 Security considerations

This use case relies on the existing security services as well as on the CDDLM security requirements for secure deployment and lifecycle management and does not introduce any new security considerations. Of primary concern in this use case is not to leak any information across the partitions of the computing utility.

4.4.8 Performance considerations

As opposed to traditional commercial data centers, the performance requirements for redeployment are higher. They are in the range of minutes for the deployment of an operating system, services and applications.

4.4.9 Use case situation analysis

There are a number of products on the market that address utility computing. These are alternatives and or additions to traditional data centers because of their ability to be reprogrammed based on the user demand. There also exist prototype implementations for accessing some of the computing utilities through the Grid

4.4.10 References

[1] Foster et al. "Open Grid Services Architecture Use Cases," https://forge.gridforum.org/docman2/ViewProperties.php?group_id=42&category_id=0&document_content_id=923

4.5 Complex Business Service Use Case

4.5.1 Summary

Complex business collaboration - within this grid use case - is the combination of grid types, grid users and usage models (batch and real-time). The management and co-ordination of this collaboration has the challenge of not only combining wide-ranging resources, but also in needing abstractions for some users and intricate detail for others.

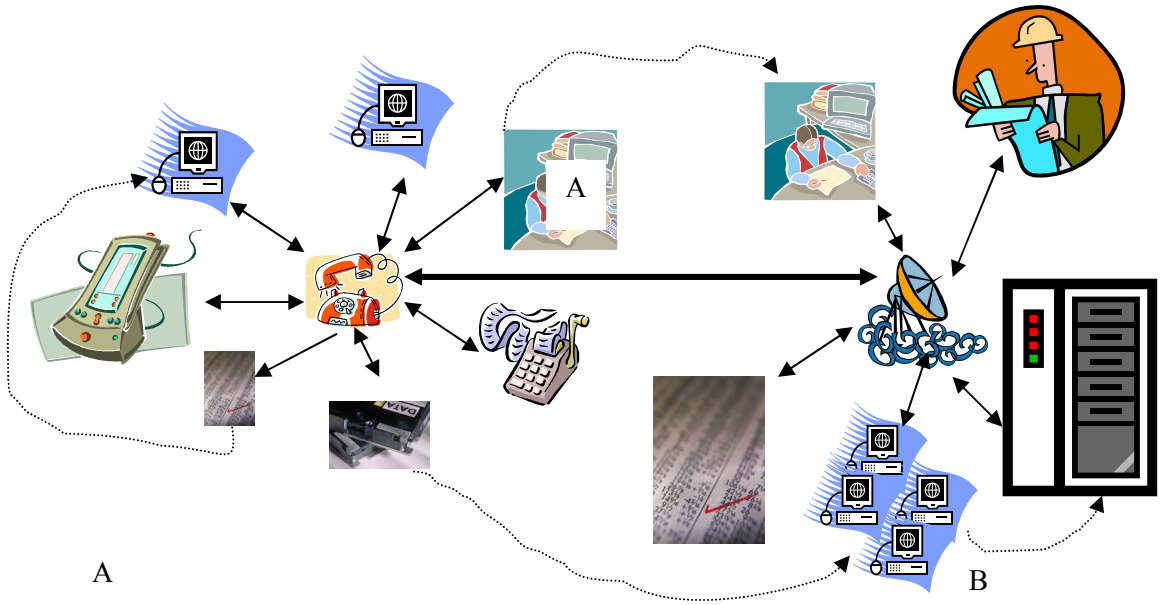

Figure 5 Complex Business Service Diagram

This diagram is trying to depict the varied user groups using varied network and computational capability, with automated and human dependency (dotted lines). In additional, the jobs are of a hierarchical nature (i.e. application X at A is using small data set and application X at B is using a larger global data set). The data, computation, and applications are distributed by both usage and control (i.e. applications that are under constant change may be held the development group, otherwise they are kept closer to the user).

This use case provides a normalized example of global investment banking (and is an exemplar for other organizations with global, regional, country structures) with its rich mix of service utilization in the trading and risk management process. In exploring this domain we meet a complex set of inter-related service types: new short-lived services, legacy services, computation services, data services, process services. It is the combination of these services in both short and long-lived business processes that requires the broad support in both grid deployment control and management. The use case is generalized, as it aims is to bring out the architectural complexity and not disappear into business intricacies. The use case covers a day in the life of this organizational type, focusing on the end of day global risk management. The case is better understood with the following diary of events:

7:30AM Traders start trading, pricing and risk analysis – accessing historical data and computational resources.

7:30AM-7PM Trader continually builds new trade types – persisting data locally and using computational resources to determine the price of these new trades

11AM Country Manager analyzes risk – extracts data from each trading data set (book) and using computational resources to generate risk reporting

5PM Country Manager analyses risk and sets the state to ready

6PM Regional Manager – extracts data for all regional trading data set and uses computational resources to generate reporting (using regionally calibrated scenarios). The result sets are compared to aggregate country result sets.

7PM Global Manager – as regional manager

The complexity of the above scenarios is compounded by:

- Complex topology of analytical code
- Fragmented data
- Large variations in CPU and network availability by country and by time of day

4.5.2 Customers

The customers in this case are service consumers – monitoring the processes from either a technical or business perspective, or waiting for process completion. The application, data or hardware owners are not covered because of scope.

The use case revolves around users in an organizations hub. These users (typified by the Global Manager) utilize computation resources (usually locally), data resources globally and are dependant on system and human events (i.e. calibration completion and reviews carried out at remote sites).

4.5.3 Scenarios

This scenario (Global Risk Management) is carried out by a Global Manager, when all data is persisted and validated, quality reviews have been carried out by data owners, and market data is available from data providers (including external providers – e.g. Reuters, Bloomberg via proprietary network devices).

The scenario is wide ranging covering many dependencies (let us consider the global hub to be in the US and the country to be Brazil):

- Trading Activity is persisted on local resources (Brazil)
- Market Data is calibrated (computation) and persisted in Brazil.
- Brazilian risk management jobs are executed on both local and globally available computation resources (with analytics that are held in Brazil). The results for these risk management jobs are persisted locally in Brazil.
- Market data and risk results are reviewed in Brazil (validation creates BRAZIL OK event).
- US Trading activity is persisted on local resources (US) and as each trader finalizes the days activity an event BOOK_OK is generated.
- Market data from external providers is persisted, and event GLOBAL_MARKET_DATA_OK is sent.
- Brazilian trading activity data set is replicated in the US.
- Each countries market data is validated (validation creates GLOBAL_MARKET_DATA, <COUNTRY>_REVIEWED event when BRAZIL_OK has been received).
- Global Risk Jobs are executed on all available computational resources (when <country>_REVIEWED and <Book>_BOOK_OK are received).

4.5.4 Involved resources

This is a simplified, but fairly standard, scenario with diverse groupings of resources. This diversity covers a range of computational capability (from Desktop PC to Large Clusters), Network diversity and distributed data. These resources are all used in a varied ways by varied collaborations – from standard batch runs (the crux of this scenario) to more ad-hoc utilization – with distributed state management services.

When considering higher level services, the complex collaboration will be a mix of computation, data, packaged application (e.g. code accessed via command line API), grid services and legacy services. Thus, when considering this use case:

Local Risk Process is a packaged application with command line API

Global Risk Process is a Grid Service

US Market data is a legacy system (with a Java connector API [2])

Brazilian Market data is held in a database.

4.5.5 Functional requirements for OGSA platform

Discovery: A distributed discovery capability is required. The complexity of service collaboration (especially when utilized by higher level users) requires that abstract collaborations can be discovered (collections of services).

Authentication, authorization, and accounting: Standard – maybe layered.

Data sharing: it is critical that data is described and discoverable in this distributed case, containing many dependant processes.

Scheduling of service tasks is the crux of the scenario. The scheduling of best available computational and data services is a primary requirement. The prioritization of jobs requires that both hedging strategies (duplicate job execution when available resource is available) and resource switching (with support for persisting and moving resource state). This prioritization is of a dynamic nature, supporting both fixed scheduling as sell as real-time short-lived schedules.

Load balancing: same as above.

Fault tolerance: Standard.

Transport management: this is key to the optimization of the schedule

Legacy application management: large global organizations require that many legacy applications be integrated into the managed grid environment. It this scenario, the legacy applications are market data sources and trade entry applications (being examples of a network device and a packaged application).

Self-Healing: Scheduled jobs must always complete, unless no supporting resources are available.

4.5.6 OGSA platform services utilization

Inter Service Communication: communication between the user and a service set (and between services) is required to support both large data set publications (i.e. when calibrated market data is available) and small event based traffic (e.g. acknowledgements).

State Management: Both high level (e.g. instantiate, initiate, start, stop, restart, etc) and sub-state (e.g. stop-begin, stop-persist, stop-snapshot, stop-persist-ok, stop-cleanup etc.) must be available both for query and messaging.

Predictive metrics: Methods must be in place to support predictive network, fault, and computational monitoring. This will be used as services are resolved into physical deployment statements (e.g. NWS[1]).

MDS: Registration of services will be core to the resolution on service requests.

4.5.7 Security considerations

Both application work and data must remain hidden (in-line with policies).

4.5.8 Performance considerations

The performance considerations for complex collaboration are both cost and time based with a need for fixed timelines. In the use case the following performance metrics must be applied:

- Local risk management (Brazil) must complete by X
- Global risk management (US) will always have priority of central resources
- Global risk management (US) will have priority on all resources after X
- Local risk management (Brazil) will have priority on local resources prior to X
- Each risk process must complete in 10 minutes
- Trade Persistence must be completed within 3 seconds

- Trade Pricing must always take priority
- WAN utilization in Brazil must be minimized unless X is likely to be breached

4.5.9 Use case situation analysis

The situation is one of both fixed and dynamic resources, both from a computational and data perspective. Core central computation and database facilities exist in the US. Market data calibration and data sources enter the network sporadically.

4.5.10 References

[1] University of California, Network Weather Service, http://nws.cs.ucsb.edu/

[2] Sun Microsystems, Java Connector API, http://java.sun.com/j2ee/connector/

4.6 Web Service development project

4.6.1 Summary

This use case demonstrates how deployment can be integrated into a development cycle. It shows how CDDLM and automated deployment can have a greater role than simply deploying a working system -they can be central to a process that delivers the working system itself. It is based on the lessons of a past project [1].

4.6.2 Customers

The direct customers are the development and operations team members, who are either deploying to their local systems (developers), or to the remote production site managed by an operations team. There is an indirect customer, who is calling an externally accessible SOAP endpoint. This customer is trying to integrate their own application, and is the primary source of bug reports. To aid the customer's development schedule, the production site with restricted access needs to be live and regularly updated from an early stage of the project.

4.6.3 Scenarios

A Web Service project is being developed by a software team for external customers. The application is being built as a Grid Service, though that is an implementation detail that is kept hidden from the callers -the customers only get to see sanitized WSDL.

The Web Service renders SVG images -it is computationally intensive and makes use of a remote image store (read only access via HTTP) and a temporary data cache (r/w access from all systems). A call can take 2-10 seconds, and when under load, the number of concurrent requests a single machine can handle is quite low. A grid architecture is used to handle peak loads by allocating resources dynamically as needed. One of the key components will be a dynamic load balancing front end that will forward inbound requests to back end execution engines. When load is light the front ends will do the work themselves; when busy they will allocate new resources and forward them. Pooling will be used to do this.

Core to delivering the service on schedule is a leading-edge deployment centric development process that is based upon the concept *Continuous Integration*, extending it with that of *Continuous Deployment*.

Developers work on their own systems, and test and debug deployment to their local machines. To be precise, they deploy onto virtualized machines with OS images supplied by operations. This is done to increase the isolation between developer systems and production units: developers will always be testing on platforms that resemble the production systems, not development boxes.

A server system continually watches for changes in the SCM repository, triggering rebuilds and unit tests when its content changes. After the unit tests pass, the system is deployed to a local grid, where functional tests run against the system.

Key to the development process is a policy of treating all deployment problems -even configuration ones- as defects for which tests can be written. Thus immediately after deployment is complete, a set of tests are run against the server to look for regressions. These are reported to the development team. Because the rebuild and redeploy is fully automated, and triggered on SCM check-in events, the system can deploy up to four or five times an hour.

Every night, while more CPU resources are freed, a larger grid is created, including all the development machines and full load tests run on the system. In the morning, if the results of the nightly build are adequate, the code is deployed to a preproduction site. This is done during midday to better field any unexpected side effects of the update -usually interoperability issues with the client application. Deployment to production systems is fully automated, but can only be done by the operations team. The production system will be a high-specification server with the option of sharing some resources across other servers in the DMZ.

Once the service goes live, the system will still be monitored and updated. One early problem found during development was that unless the load-balancer was fully aware of the health of other nodes, it would route requests to systems that were unwell, because their request queues were shorter. This problem was fixed by adding more health checks to each node, and have the load balancer probe a URL on each node that triggers an internal health check. Thus *liveness* is not defined by the existence of the application on a node, but by the ability of the application to probe its own health by:

- Rendering test images and comparing them against 'golden' bitmaps. The test images verify that the required fonts are all present.
- Saving a file to the temporary filestore and verifying the timestamp of the saved file is close to that of the local machine, as having timezone or clock differences causes confusion.
- Making DNS requests of configured helper sites.
- Any other deployment-time defect that can be easily tested for, or which can be used to test for a critical system failure.

When delivered, the service will provide computationally intensive functionality, under variable load, but at a low operational cost which comes from being able to share both development and production computing resources.

4.6.4 Involved resources

In this scenario, CDDLM is the means by which developers, operations, and the continuous integration build tool deploy applications to development, testing and production systems. The initial benefit is a fast and simple deployment mechanism that is independent of the final target - deploying to a production site is no more complex than deploying to a development system, although the production deployment descriptor is somewhat more complex. The long term benefit is that the use of a grid architecture with dynamic resource allocation will make running the service more cost effective.

The development boxes will be the initial resources will be hosted on the local systems, inside virtual PC VMs. The VM images will be supplied by operations; developers connect to and deploy their code to these local images. The automated test deployment process (which may itself be deployed using CDDLM), uses a LAN-wide grid of available systems, making use of idle

workstations when possible. The production system can use whatever operations choose to provide -either dedicated hardware or shared resources; there is no need for their choice to remain constant over the life of the service.

Most of the systems are computational nodes; the router component and the file store are different. Development and test systems can use local filestores; production needs a better solution, but still only of LAN-scope. A networked file server may be the basic solution. As it is only for transient storage of the output, a full RAID-5 storage device is overkill.

4.6.5 Functional requirements for OGSA platform

Resource Virtualization: The testing and production systems should be virtualized so that the underlying hardware is isolated. Because development systems use a VM product for hosting their test platforms, they are already isolated.

Web Service Integration: As the production service is a SOAP-based Web Service, it is essential for the OGSA platform to integrate with the existing Web Service standards. In this scenario, one would envisage the public WSDL description being hand-written; the GWSDL description used by the OGSA platform would be auto-derived from this in the build process. A related issue is how to route incoming SOAP requests from a public endpoint to the many back ends. Some kind of router service will have to be written.

Self-Healing: Because of the broad set of tests used to assess the health of the functional system, while not self-healing, the application nodes are self-aware of their own health. This could be used by the OGSA framework to trigger fault recovery.

Administration: The IT and Operations groups must be able to simply manage the process of deployment and report on its ongoing status. Merging status logs from running programs into a unified status log, and getting a full view of the health of the production system are both important needs.

Security: Only operations are allowed to deploy to the production site; only the automated build tool can deploy the test system -though users can request this action from the tool. The developers' boxes have minimal security needs -they may even offer each other access for debugging purposes.

4.6.6 OGSA platform services utilization

Service Interaction Services: Most nodes do not need to interact with each other, but the front end routing services need to be aware of what computation nodes are available and healthy.

Security Services: Security is needed to control access to the system by different users and the roles they have within the organization. Callers also need to be authenticated.

Data Services: It is presumed, as shown above, that interaction with various data sets will be an integral part of the operations of Grid services within these and other scenarios.

Program Execution Services: The use of a Program Execution Grid is a prerequisite for enabling the test and production deployments

Resource Management Services: The scenario depends on resource management during testing and production deployments.

4.6.7 Security considerations

The deployment process restricts who can deploy to what system. Even developers, with the ability to deploy builds to their development machines, are not given the rights to fully configure the virtualized servers they run there -that is the privilege of operations. Certainly developers are

not given access to the production system. Developers do have the indirect ability to run arbitrary code on the production systems, and so some of this 'no developers near the server' philosophy is a tradition, rather than one based on an absolute mistrust of developers. Operations are worried that developers will do things to the machines that will break them or make them less secure.

The publicly accessible front end is where greater security concerns are placed -the server is a visible web service with all the implications of that. Whichever SOAP toolkit the server is built from, the developers need to know that it does not have any well known security bugs, and that if one is found, a patched version of the toolkit can be redeployed. The developers have also to take care in defending their service from malicious data. For example, the XML document of SVG to render could request the importing of the file /etc/password through some malicious entity expansion, or include requests to render images that could access other parts of the local file system. Such issues are beyond the scope of the OGSA framework, but fundamental risks associated with XML messaging and the processing of XML documents from untrusted sources. What can be done to minimize the risks is configure the systems to be 'locked down' in terms of all security options, and to run the services in a sandbox that has restricted rights.

Assuming the production site is co-located at a remote site, all management access had to go over a secure channel -presumably SSH, although SSL/TLS with client-side certificates would also work. During the preproduction phase, the public server should only be accessible by the software team, and the external customer -IP address filtering or a VPN could be used here. Configuration of the router would also be something that the operations team would like declarative control over, as it is hard to do by hand. A deployment component that would talk SSH to the (Cisco) router would be required for this.

4.6.8 References

[1] *When Web Services Go Bad*, Loughran, 2002.
http://www.iseran.com/Steve/papers/when_web_services_go_bad.html

[2] *Making Web Services that Work*, Loughran, 2002.
http://www.hpl.hp.com/techreports/2002/HPL-2002-274.html

4.6.9 Use case situation analysis

The scenario is a hypothetical descendent of an existing (and documented) Web Services project [1], with the development processes explored therein used to transform how projects are tested and deployed [2]. There are three separate deployment destinations: developer systems, a local test grid and the production server complex, each with their own configurations -but all sharing a common deployment mechanism: CDDLM. A notable feature of this scenario is its rate of deployment -it can happen every few minutes on a development system, and a few times an hour on the test framework. Because the customer is an external user of the Web Service, they also need access to the daily builds of the system, which means that even the pre-production site is updated on a daily basis.

The fact that many of the systems are really virtualized OS images hosted on development systems may or may not complicate the equation. We'd envisage some centralized deployment mechanism to even get the preconfigured OS images out to developers, letting operations provide emulations of their locked down production systems. One could imagine that the overnight tests would also make extensive use of virtualized operating systems, so that the team could better simulate the physical architecture of the production site.

5 High Level Architecture

Need text here describing architecture in more detail. Also use this place to provide an introduction to the four forthcoming specs. These four subsections will be written by the four specs respective owners. In a way it will be introduction for what the specs will be about and an opportunity for the authors of these documents to get an early feedback from the community on what they plan to accomplish.

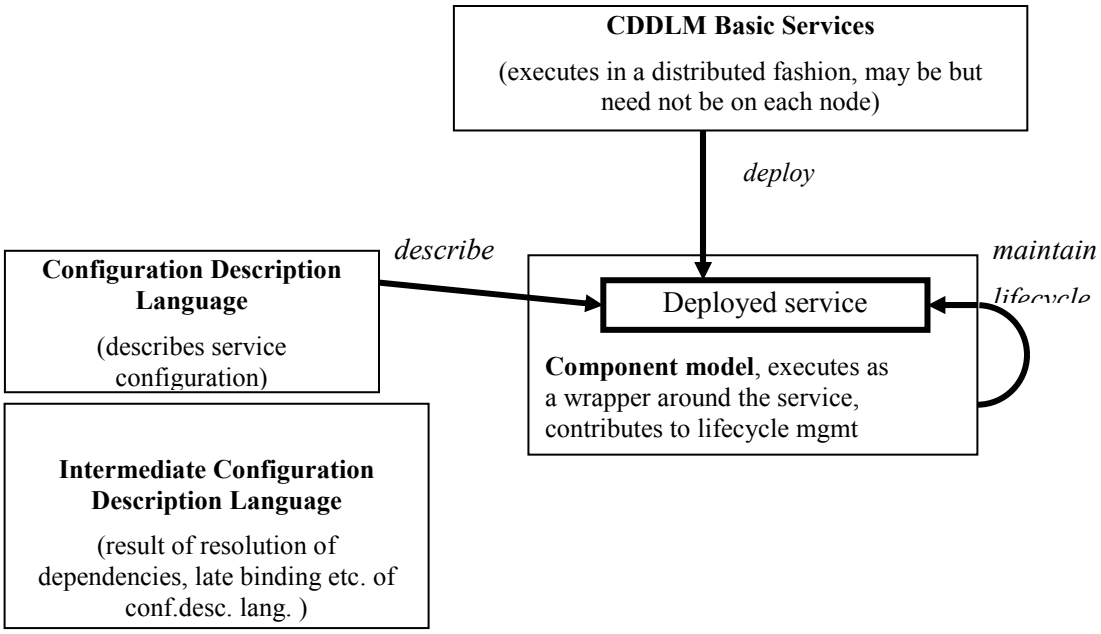

Figure 6: The CDDLM High Level Architecture.

Four key components contributing to the CDDLM Spec and their relationship to the deployed service.

5.1 Configuration Description Language Specification

Figure 7 describes CDL language processing flow from front end languages to target environment. CDDLM assumes multiple front end languages as well as multiple target . The front end may be a textual language or graphical user interface. XML based CDL is normative language specification that may be used for a front end language or intermediate language that is translatable from any front end language. The component descriptions literal translation from the front end language includes unresolved parameters and references to other components used as template to generate more specific descriptions of the target. Those parameters and references are partially resolved before the deployment. Some of them are retained as unresolved and resolved during the deployment or initialization of the execution.

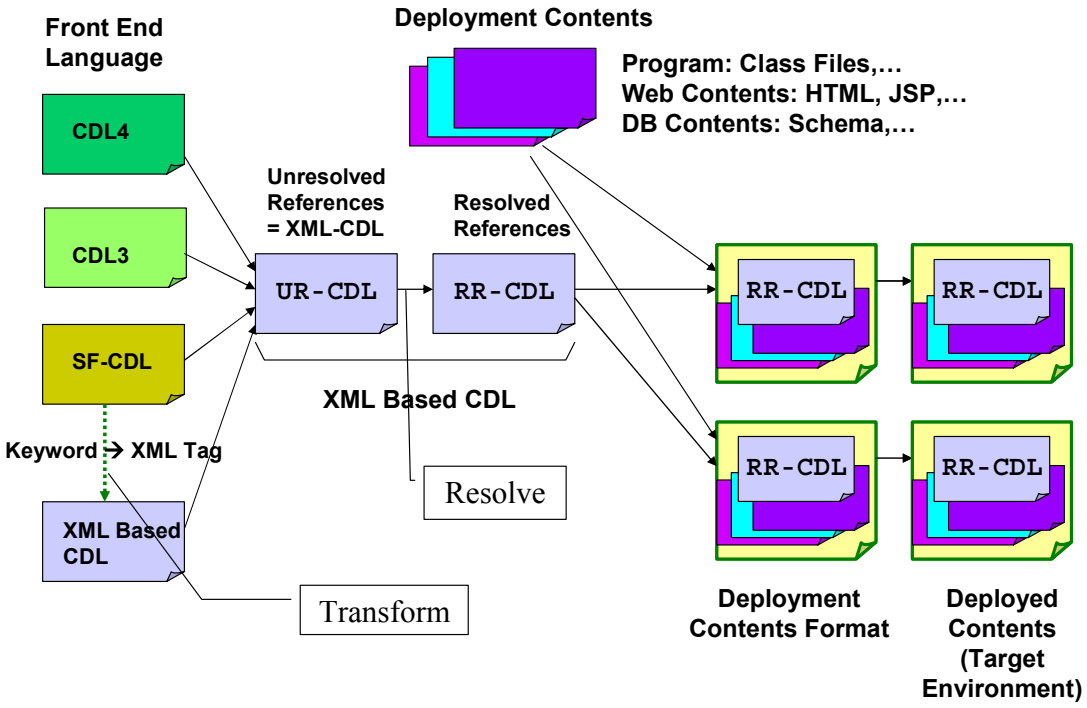

Figure 7 Language Processing Flow

5.2 Basic Services Specification (Kojo)

Figure 5.3 depicts assumed deployment procedure performed by basic services. The services include following categories of services.

1) CDL Compilation Services

   This category of services parses descriptions of components in CDL, verifies consistency of the descriptions, resolves component references as far as it is resolvable before the deployment and composes a package ready for deployment.

2) Deployment Services

   This category of services carries the deployment package to hosting environment which include servers, storage or network devices.

3) Lifecycle Management Services

   This category of services manage lifecycle of the deployed system. The operations include initialize, start and terminate the system.

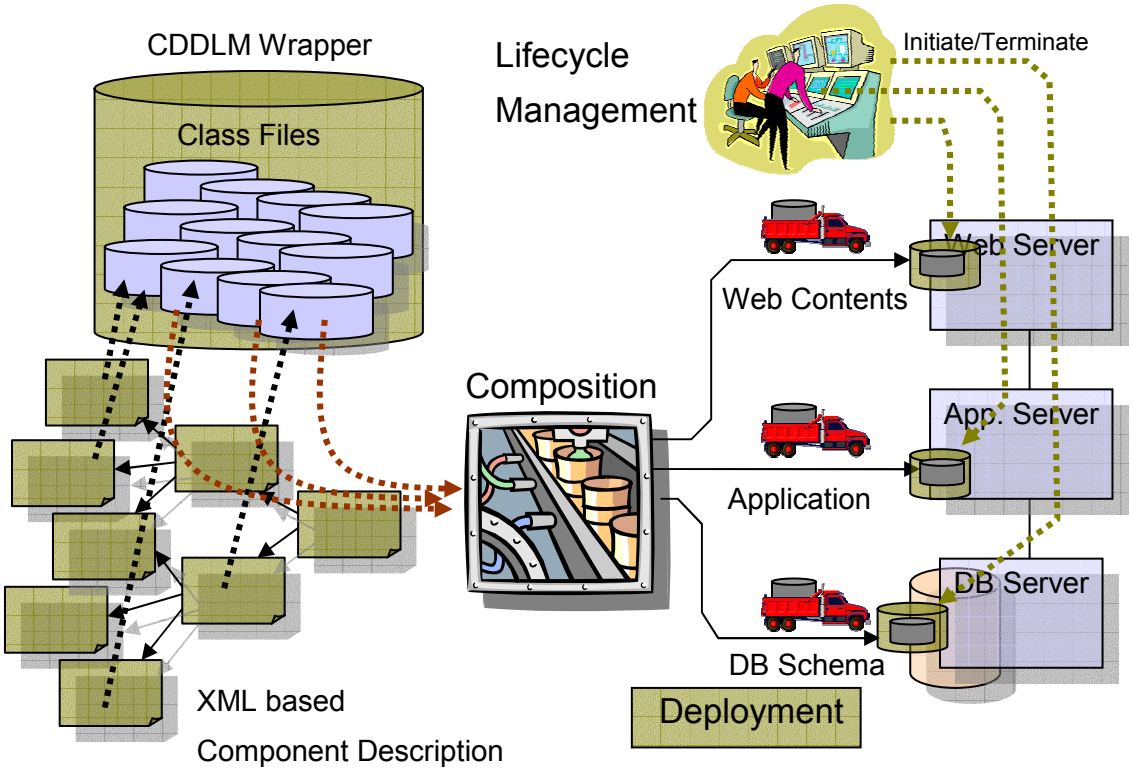

Figure 8 Deployment Procedure

5.3 Component Model Specification (representation & implementation)

Every deployed service managed by the CDDLM service will implement a base set of interfaces, defining a core abstract component model. As there may be a many-to-many relationship among Grid services and actual instantiated software resources, the CDDLM component model defines a means to loosely group a hierarchy of services within a deployment container. Each Grid service to be managed will implement interfaces in three categories: Configuration, Lifecycle Management, and Maintenance. The model also provides support for the relationships between services and their components.

5.4 The Bootstrap Problem

A CDDLM infrastructure, deployed across a set of resources, will be able to deploy other applications and components onto the resource set. How does the CDDLM infrastructure instantiate itself?

This is one of the problems to address. Some options are:

1. Tightly integrate the CDDLM server components with the GTK toolkit or other Grid framework, such that deployment requests can be made to any running node.
2. Use the CDDLM server to instantiate the Grid framework itself, and then handle incoming requests to deploy application components onto the instantiated framework. This has the benefit of keeping the Grid framework up to date.

3. Have a bootstrap architecture that is independent of the final CDDLM implementation, but which, by virtue of being deployed across all nodes in a resource set, be used by the CDDLM implementation to deploy itself across the fabric.

5.5 Service Interface

The core service interface provides a means for remote users to submit requests to the CDDLM deployment infrastructure. One goal of the interface would be to support composition of systems from other systems. The interface needs to be designed so that it provides enough information and control for one CDDLM instance to deploy via another, in a complex deployment configuration.

The core request is a message requesting deployment of a system -this would contain the deployment descriptor in the CDL language. The system also needs access to the system to deploy -this could be done as an attachment, though attachment support in SOAP frameworks is inconsistent. An alternative is to get the binary data to an intermediate store where it can be accessed by the CDDLM service, but not readable by others.

A submitted deployment request will either result in deployment or a failure. We will need to define the standard failures and provide appropriate SOAPFault declarations.

A deployed application may need to be stopped, and have a status page that can be probed for system health. A classic Web Service would support this via actions on the single SOAP endpoint, but as this is a Grid Service, we may prefer to create a service instance for every deployed application, and provide a GSH to locate this service instance.

6 Related GGF and other Standard Bodies Working Groups

6.1 GGF OGSA

“The purpose of the OGSA Working Group is to achieve an integrated approach to future OGSA service development via the documentation of requirements, functionality, priorities, and interrelationships for OGSA services. Topic areas that we expect to scope and discretize early are common resource model and service domain mechanisms, but the precise set to be addressed will be determined in early discussions.

The output of this WG will be an OGSA architecture roadmap document that defines, scopes, and outlines requirements for key services. It is expected that the development of detailed specifications for specific services will occur in other WGs (existing or new). When this document is produced, the WG will consult internally and with the GFSG to determine whether to close or, alternatively, to work to produce a second edition.” [from OGSA-WG Charter]

CDDML is to be positioned at the architecture, taking the role of deployment and lifecycle management of the grid architecture. The WG defined conditions for OGSA branding as follows. Those are regarded as a basic conditions standard component to be positioned at the OGSA architecture.

1) Operating within the OGSI framework.
2) The work fit into the Architecture as defined by the OGSA doc
3) Engage actively with us and other OGSA related WG
4) Use OGSA as an adjective.

6.2 GGF OGSA Program Execution

Program Execution is one of the basic components of OGSA which includes Job submission, Scheduling and Resource Brokering and other functionalities required for job execution. Detail discussions are still undergoing in the working group, but functionalities of CDDLM should be positioned and used appropriately in the architecture.

6.3 GGF CMM

The CMM working group was initially formed in Spring, 2003 to define the Common Manageability Model (CMM) specification which defines the base behavioral model for all resources and resource managers in the grid management infrastructure. A mechanism is defined by which resource managers can make use of detailed manageability information for a resource which may come from existing resource models and instrumentation, such as those expressed in CIM, JMX, SNMP, etc, combined with a set of canonical operations introduced by base CMM interfaces. The result is a manageable resource abstraction that introduces OGSI compliant operations over an exposed underlying base content model. CMM does not define yet another manageability (resource information) model. [From CMM Charter]

Later, as OASIS-open launched Web Services Distributed Management (WSDM) Technical Committee, the attempt in CMM-WG was handed over to WSDM-TC. In collaboration with the TC, CMM-WG will be focusing on the gap analysis on the WSDM specification and the grid requirement.

6.4 OASIS WSDM

CDDLM is related to OASIS/WSDM (Web Services Distributed Management), but it is sufficiently focused on the deployment that neither of these management standards addresses the level of detail of the CDDLM. It is also related to provisioning as part of OGSA. However, CDDLM is not a provisioning tool. As a deployment tool, it can be used for provisioning. If we define "provisioning" as a task to assign or allocate resources to a Grid job to include: monitoring workload; analyzing climate and trend and forecast the future; planning for resource allocation; and deployment and configuration of newly allocated resources. Provisioning is then responsible to ensure that the resources meet a minimum capability set. Once this task is complete, CDDLM is responsible for the software lifetime of the Grid service including its deployment and configuration on provisioned hardware. In that light, only item (d) is covered by the CDDLM. In order to handle physical IT resources through Grid Service, CDDLM should be related to many existing management specifications; e.g. SMTP, GMPLS, etc. CDDLM also has close ties with CMM and OGSA, but there is no significant (if any at all) overlap between CDDLM and these other working groups.

The WSDM-TC was formed to define web services management. This includes using web services architecture and technology to manage distributed resources. This TC will also develop the model of a web service as a manageable resource. This TC will collaborate with various evolving activities within other standards groups, including, but not limited to, DMTF (working with its technical work groups regarding relevant CIM Schema), GGF (on the OGSA common resource model and OGSI regarding infrastructure), and W3C (the web services architecture committee). Also liaison with other OASIS TCs, including the security TC and other management oriented TCs. [From WSDM Charter]

6.5 OASIS SPML & WS-Provisioning

Service Provisioning Markup Language (SPML, www.openspml.org) is an OASIS (www.oasis-open.org) standard, which defines an XML- based framework for exchanging information between Provisioning Service Points. It is a framework for the exchange of user, resource, and

service provisioning information. SPML is being defined by the OASIS Provisioning Services Technical Committee. It is focused on user provisioning. Their definition is "Provisioning is the automation of all the steps required to manage (setup, amend & revoke) user or system access entitlements or data relative to electronically published services".

WS-Provioning has been proposed as an input for SPML v2.0. It describes the APIs and schemas to facilitate interoperability between provisioning systems and to allow software vendors to provide provisioning facilities in a consistent way. The specification defines a model for the primary entities and operations common to provisioning systems including the provisioning and de-provisioning of resources, retrieval of target data and target schema information, and provides a mechanism to describe and control the lifecycle of provisioned state.

6.6 CIM

CIM (Common Information Model) has been developed and standardized by the Distributed Management Task Force (DMTF, www.dmtf.org). CIM is a common data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment (http://www.dmtf.org/standards/cim). It is based on the Unified Modeling Language (UML), and within a single meta-model, contains models for numerous resources, applications and services. CIM models are represented in either UML or in textual form in MOF (managed object format).

6.7 DCML

Data Center Markup Language (DCML) is a newly formed consortium (www.dcml.org) which proposes a "vendor-neutral, open language to describe data center environments, dependencies between data center components and the policies governing management and construction of those environments." The DCML is about: automating *configuration and change* of a Data Center; improving *visibility* of the Data Center representation, such as configuration, usage, etc.; and *fault monitoring* of a data center. It is based on the concepts of: adaptability; representing best practices, standards, and policies; and heterogeneity (diversity). DCML is built around five different specifications addressing framework, hardware, network, storage, and software.

6.8 WS-Notification

The system will have need to send notifications back to users of the system, messages that cannot be made by synchronous calls to a users' own Grid Service instance, as the user may be beyond a firewall or offline. We therefore need to use an asynchronous notification mechanism that meets our needs. WS-Notification may be this mechanism, or another notification mechanism that gets recommended by the GGF for secure, through-firewall callbacks.

7 Related Work

The section is an overview of technologies related to the CDDLM problem space. CDDLM's domain of interest starts at the point at which resources have been allocated for the deployment of a service, so we are interested in technologies that can deploy and configure software onto the resources to deliver a running service. These technologies include OS installation automation tools, node imaging tools, and fabric management tools. The focus here is on compute nodes as resources, but the principles can be extended to other resource types.

OS installation tools, such as the Kickstart package for Red Hat Linux, automate the process of installing an operating system and application packages, generally using a configuration file or response file. This is automatic, but node preparation generally takes a significant amount of time. Node imaging tools, such as Altiris or Rembo, extend this capability by allowing complete

disk images to be customized and installed on a node, from a library of gold images – which is faster, but there is generally less flexibility in being able to adapt the configuration of the target node.

Both of the approaches above are rather static in their ability to deploy software. Systems that are capable of managing the ongoing, dynamic configuration of nodes, such as LCFG, maintain a database of the configuration of all nodes. When the configuration of one or more nodes is to be changed, this is entered in the database and the configuration of the affected nodes is changed appropriately. Such technologies can be given a node in a default (but fully-operational) state, and very quickly adjust its configuration to play a part in a specific service.

One drawback of the approaches discussed so far is in their ability to co-ordinate the deployment of a configuration across several resources. This capability is firmly in the domain of interest of CDDLM, and is embodied in systems such as SmartFrog, which allow the creation of configuration-driven systems, where complex software deployments can be orchestrated across distributed resources, based on service configuration data.

An additional weakness of static deployment methodologies is their need to maintain interdependency information for software packages to avoid installation conflicts, or to stovepipe software in order to guarantee successful deployments. The SoftGrid system from Softricity eliminates these problems by enabling software to be deployed on-demand to resources and operate without actual installation of the software onto the node. Using virtualization technology, the Softricity system prevents nodes from degrading over time as a result of the process of reconfiguration, and guarantees software is deployed successfully with the proper, specified configuration.

For a more substantial discussion of these concepts, we refer the reader to [4]. (ßNote: needs to be converted to proper cross-reference to the references, plus we need references for whatever list of technologies we settle upon.)

8 Future Work

1. Follow-up with four Specs
2. Link with OGSA (get feedback on the foundation document, follow-up specs)
3. Maintain relationship to other GGF groups, in particular DAIS, CMM, and
4. Establish relationship to standard bodies outside of GGF, in particular DMTF/WSDM
5. Establish relationship to industries and academia
6. Come up with at least two interoperable reference implementations
7. More

9 Security Considerations

Security issues have been broadly covered through the document

10 Editor Information

David Bell
Department of Information Systems and Computing,
Brunel University,
Uxbridge,

Middlesex,
UB8 3PH,
United Kingdom.
Phone: +44 (0) 1895 203397
Email: david.bell@brunel.ac.uk

Patrick Goldsack
Internet Systems and Storage Laboratory
Hewlett-Packard Laboratories
MailStop HPLB
Filton Rd.
Stoke Gifford
Bristol BS34 8QZ
United Kingdom
Phone: +44 117 312 8176
Email: patrick.goldsack@hp.com

Takashi Kojo
Solution Platform Software Division
NEC Corporation
2-11-5 Shibaura
Minato-ku, Tokyo Japan 108-8557
Phone: +81-3-5476-4386
Email: kojo@bk.jp.nec.com

Steve Loughran
Internet Systems and Storage Laboratory
Hewlett-Packard Laboratories
Filton Rd.
Stoke Gifford
Bristol BS34 8QZ
United Kingdom
Phone: +44 117 312 8717
Email: steve_loughran@hpl.hp.com

Dejan Milojicic
Internet Systems and Storage Laboratory
Hewlett-Packard Laboratories
MailStop 1183
1501 Page Mill Road
Palo Alto, CA 94304
Phone: (650) 236 2906
Email: dejan@hpl.hp.com

Stuart Schaefer
Chief Technology Officer
Softricity, Inc.
27 Melcher Street
Boston, MA 02210
617-695-0336
http://www.softricity.com
sschaefer@softricity.com

Junichi Tatemura
NEC Laboratories America, Inc.
10080 North Wolfe Road, Suite SW3-350
Cupertino, CA 95014-2515
Phone: +1-408-863-6021
Email: tatemura@sv.nec-labs.com

Peter Toft
Internet Systems and Storage Laboratory
Hewlett-Packard Laboratories
MailStop HPLB
Filton Rd.
Stoke Gifford
Bristol BS34 8QZ
United Kingdom
Phone: +44 117 312 8728
Email: peter.toft@hp.com

11 Contributors

We gratefully acknowledge the contributions made to this specification by TBD.

12 Acknowledgements

This work was supported in part by TBD.

Appendix: Information Sources

In identifying services we draw upon the following sources (references to be provided):

- TBD

References

1. Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf.
2. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, *15* (3). 200-222. 2001.
3. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S. and Kesselman, C. Grid Service Specification, 2002. www.globus.org/research/papers/gsspec.pdf.