GFD-I.050
Configuration description, Deployment and Lifecycle Management Working Group
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

Draft: April 9, 2004
Final: August 7, 2005

Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation Document

Status of this Memo

This document provides information to the community regarding the specification of the Configuration Description, Deployment, and Lifecycle Management (CDDLM) Language. Distribution of this document is unlimited. This spec is related to a set of four specs that together comprise service configuration description, deployment, and lifecycle management. These specs are described in the following three paragraphs. For more details please refer to these specs.

Copyright Notice

Copyright © Global Grid Forum (2002, 2005). All Rights Reserved.

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 Web 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 Specs

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

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

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

The CDDLM Configuration Description Language (CDL) is an XML-based language for declarative description of system configuration that consists of components (deployment objects) defined in the CDDLM Component Model. The Deployment API uses a deployment descriptor in CDL in order to manage deployment lifecycle of systems. The language provides ways to

describe properties (names, values, and types) of components including value references so that data can be assigned dynamically with preserving specified data dependencies. A system is described as a hierarchical structure of components. The language also provides prototype-based template functionality (i.e., prototype references) so that the user can describe a system by referring to component descriptions given by component providers.

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

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

Table of Contents

List of Figures 5
List of Tables 5
1 Introduction 6
2 CDDLM WG and the Purpose of this Document 6
3 Functionality Requirements 9
3.1 Basic Functionality Requirements 9
3.2 Security Requirements 10
3.3 Resource Management Requirements 10
3.4 System Properties Requirements (Non-Functional Requirements) 11
3.5 Configuration Description Language (CDL) Requirements 11
3.6 Service Interface Requirements 13
4 Use Cases 13
4.1 Web Services in the Grid Environment 13
4.1.1 Summary 13
4.1.2 Users 14
4.1.3 Scenarios 14
4.2 Commercial Data Center 15
4.2.1 Summary 15
4.2.2 Users 15
4.2.3 Scenarios 16
4.2.4 Involved resources 16
4.3 IT Infrastructure and Management 17
4.3.1 Summary 17
4.3.2 Users 18
4.3.3 Scenarios 18
4.3.4 Involved resources 19
4.3.5 Use case situation analysis 19
4.4 Utility Computing Model 19
4.4.1 Summary 19
4.4.2 Users 20
4.4.3 Scenarios 20
4.4.4 Involved resources 21
4.5 Complex Business Service Use Case 21
4.5.1 Summary 21

4.5.2 Users 22
4.5.3 Scenarios 22
4.5.4 Involved resources 23
4.6 Web Service Development Project 24
4.6.1 Summary 24
4.6.2 Users 24
4.6.3 Scenarios 24
4.6.4 Involved resources 25
4.6.5 Use case situation analysis 26
4.7 Functional requirements for OGSA platform 26
4.8 OGSA Platform Services Utilization 27
4.9 Security considerations 27
4.10 Performance considerations 28
5 High-Level CDDLM Architecture 28
5.1 Configuration Description Language Specification 29
5.2 Basic Services Specification 30
5.3 Component Model Specification (Representation & Implementation) 30
6 Related GGF and other Standard Bodies Working Groups 30
6.1 GGF OGSA 30
6.2 GGF OGSA Basic Execution Services 31
6.3 OASIS WSDM 31
6.4 OASIS SPML & WS-Provisioning 31
6.5 CIM 32
6.6 DCML 32
6.7 WS-Notification 32
6.8 GRAAP 32
6.9 JSDL 32
7 Related Work 32
8 Future Work 33
9 Security Considerations 34
10 Editor Information 34
11 Contributors 35
12 Acknowledgements 35
Intellectual Property Statement 35
Full Copyright Notice 36

References 36

List of Figures

Figure 1. Provisioning Layers 7
Figure 2. Use Case in Program Execution Environment.....8
Figure 3. IT Infrastructure and Management 17
Figure 4. Utility Computing Model.. 20
Figure 5. Complex Business Service Diagram 22
Figure 6. The CDDLM High Level Architecture. 29
Figure 7. Interaction of CDDLM and Other Management Components.....29

List of Tables

Table 1. Scope of CDDLM. 8

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 can automatically and repeatedly 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 Web Services, and composite systems made up of many co-operating Web Services.

CDDLM relates to other areas in GGF and OASIS, such as Basic Execution Services (BES) within OGSA (BES may make requests to CDDLM for deployment), WS-Agreement (facilitates resource allocation), and OASIS WSDM (facilitates management).

CDDLM should not be confused with provisioning, which is a broader term. Provisioning is the process of acquiring and managing the resources required to complete a planned task. The resources required to run a batch job generally include hardware (for example processors, memory, storage and networking resources), software (for example applications, data and any applicable licenses) and other configurable items (for example user accounts). The provisioning cycle encompasses the following activities:

1. *Scheduling* the resources from an available pool some time prior to the start of the task.
2. *Allocating* the specific set of resources to be used when the task starts.
3. *Deploying* the resources after they have been allocated.
4. *Managing the lifecycles* of the deployed resources from the time of allocation to the time of task completion.
5. *Releasing* the resources following completion of the task.

Of the steps outlined above, CDDLM is covering 3, 5, and deployment-related parts of 4. CDDLM does not deal with 1 and 2. These and other terms are described in more detail in the OGSA Glossary [1].

1 CDDLM WG and the Purpose of this Document

The CDDLM WG 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 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) SmartFrog-based Configuration Description Language Specification [2]
2) XML-Based Configuration Description Language Specification [3]
3) Component Model Specification (component representation and implementation) [4]
4) Deployment APIs Specification [5]

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 expectations for the forthcoming deliverables.
2) Offer to the community a preview of what the WG will deliver and enable them to provide us some early feedback.
3) Set a stage for collaboration with other RG/WG within and outside GGF.

To further expand on the relationship of CDDLM and the provisioning cycle, deployment can be positioned as a part of the 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 that consists of several phases. These phases can be categorized into (see Figure 1): 1) Execution and Monitoring, 2) Analysis and Projection, 3) Resource Allocation Planning, 4) Deployment. The provisioning process manages various layers of the target systems from hardware to application layer. The application management layer addresses components such as application program, content data or DB schema for specific applications. The infrastructure management layer addresses *middleware components* such as Web server, Application server or DBMS*; operating systems*; and *hardware components* (resources) such as server, storage, firewall, load balancer or network switch.

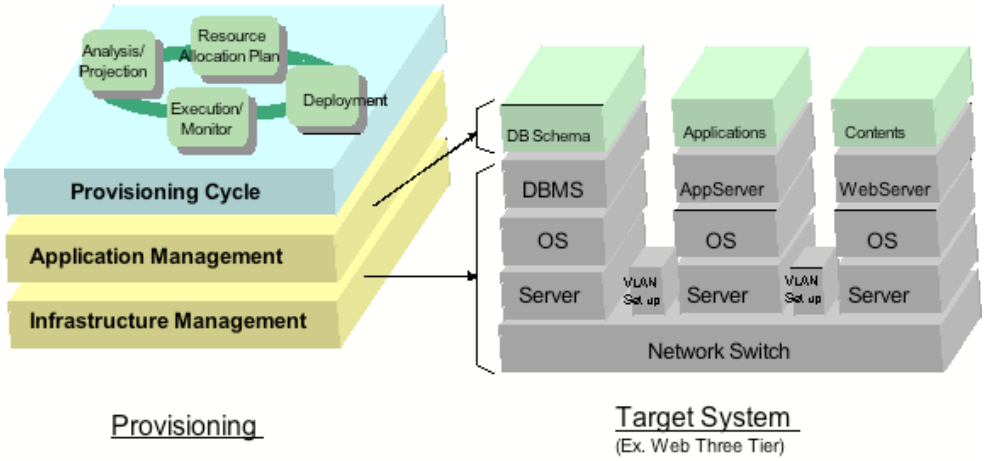

Figure 1. Provisioning Layers

In the remaining text, we differentiate *service components,* which represent software that is configured and deployed from *resources*, which represent hardware resources on top of which services are deployed.

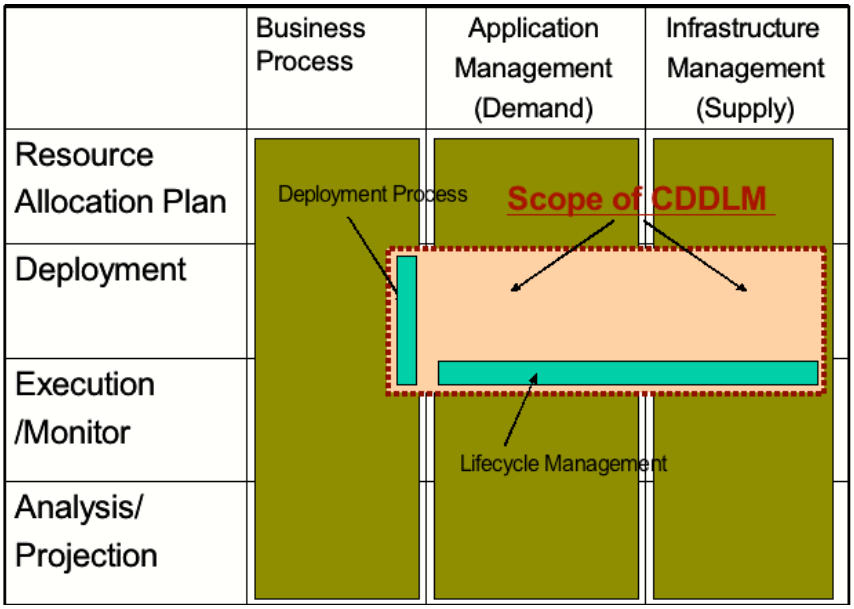

Table 1. Scope of CDDLM.

Table 1 describes 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.

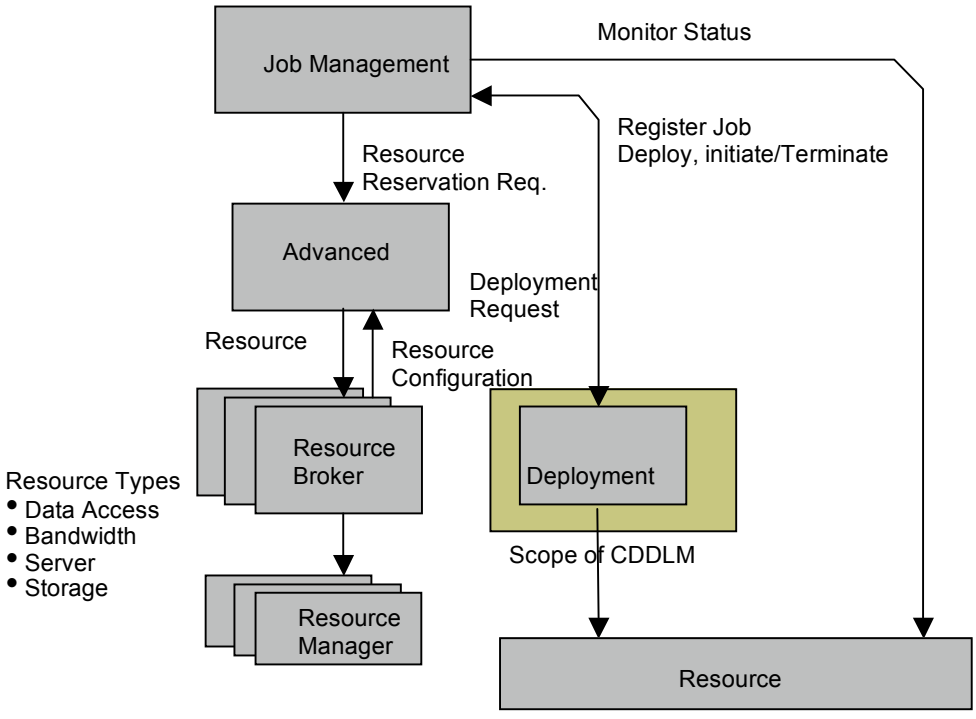

Figure 2. Use Case in Program Execution Environment

Figure 2 describes the assumed use case of CDDLM in a 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 from outside of CDDLM by services, such as resource brokers that are provided as a set of component descriptions. CDDLM assumes that management of respective target components will be performed through OASIS WSDM management interfaces.

2 Functionality Requirements

2.1 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 host in one place only, but used 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 undeployed from their resources
  - A means to allow service components to interface with the deployment mechanism in order to deploy and undeploy 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 so 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
  - A means of recovering components from failure (restarting)

2.2 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 –they 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 should leverage, the standard Grid and Web services security mechanisms.
- In the first CDDLM set of specifications, 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.

2.3 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. It is too early to finalize the exact target of the deployment, because some of the related WGs have not been finalized, for example, OGSA Basic Execution Services (BES). The target of the deployment could be the physical resources, or the containers for resources. CDLLM WG will continue to interact with other groups, such as GRAAP and BES in determining exactly the interfaces and target objects for the deployment.

- **Resource description.** The description of the resources onto which the service will be deployed.
- **Parameterization.** The parameters describing resource capacities, such as memory size, type and speed of processors, size of disks, etc.
- **Relationships.** The relationships between certain hardware resources, such as computers in the farms.
- **Composition.** Combining resources to compose a bigger entity.
- **Inheritance.** Making incremental changes to resources while inheriting the description of the previous resource.
- **Run-time binding.** Enabling binding of components at run-time, for example, the addresses of Web servers and database servers, which will be known only at run-time.

2.4 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 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 resolution of lazy dependencies.
- **High-availability.** CDDLM should be highly available to the requests for deployment. In particular, it should respond well to high request loads and 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, provide support for replication of services and possibly more complex support for recovery if significant parts of the deployment services themselves fail.
- **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, recover from failures, or 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.

2.5 Configuration Description Language (CDL) Requirements

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.

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 (WXS) should be the preferred description of the syntax of the language, and yet troublesome parts of that format (e.g. 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.

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

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

- some configuration parameter values cannot be determined before deployment time
- configuration may change at run time
- there exist changes in environments (e.g., system failure)
- there exist changes to the CDDLM implementation

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

Integrity. The CDDLM implementation should manage integrity of 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 CDL should support the following integrity aspects:

- 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 detects 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 the 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)

Composability. The total service configuration description should 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.

Discovery (reference and binding). CDL should support:

- reference to external configuration descriptions

- reference to components (packages to be deployed)
- reference to services that provide deployment functionality of components

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

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.

2.6 Service Interface Requirements

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

- Support for remote access through firewalls
- Should integrates with Grid and Web Services 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; undeploy will clean the resources of any residual data
- 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 Use Cases

For use cases, we were motivated by the OGSA use cases [6]

3.1 Web Services in the Grid Environment

3.1.1 Summary

Web services can be used to represent and virtualize any resource in the environment. The resources may be actual working application program, middleware, operating system or any kind of hardware such as server, 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 initiating 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 settings, downloading and setting up all those defined setups in appropriate sequence, and enforcing the setting.

3.1.2 Users

Since Web services are abstracted models, one can consider a whole spectrum of users of grid computing. In this particular section, we consider both large scale application development and Quality Assurance (QA) scenarios. 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 future uses.

In typical application development or QA department, they have to deal with number of applications for a long time period. 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 difference unique to the application in question. This feature is also useful when they want to examine many different configurations which have a lot of commonality as well.

3.1.3 Scenarios

1) Configuration Definition

   The engineer first defines configuration of the application and system using a tool with graphical user interface or textual definition language. He can refer to other definitions done in the past or other places. He can extend or modify existing definitions and derive new definitions. This increases productivity and eliminates new errors that might be introduced in the definitions. He also has to define appropriate procedures covering 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 has 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 (including sequencing) the transfer and initiation of the software components.

   The engineer wants to manage partial transfer of the contents to avoid taking a 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 for various reasons. During the debugging, they may find some errors with 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 environments of other companies. 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.

3.2 Commercial Data Center

3.2.1 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 in at least the following three aspects. In today's data center, it can take some time – often a matter of weeks - from the user's request to complete deployment and set up all the equipment and software. The lead time is crucial to both the users and the data center because it directly affects the 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 data must not only be 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.

3.2.2 Users

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 deployment components varies from a few tens to thousands, depending on the range of the deployment. When ready, the software, data

and hardware configurations are deployed to the target environment. This may be done either with the operator's manual operation or through automated operation by the administration system.

3.2.3 Scenarios

1) Configuration Definition

The user talks to a data center engineer about requirements for the 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 reside. 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 definitions by the data center engineer using either graphical user interface 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 a particular application so that it can switch between the configurations, such as production or system testing.

2) Deployment

Based on the configuration definition, the data center operator sets 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 an installation process.

The operator has to allocate required hardware available to the deployment. This may be done manually by the operator or automatically by the 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 have to be transferred to the data center or to a place reachable from the data center before the deployment. Deployment should be initiated either by the operator or automatically.

3.2.4 Involved resources

Types of the resource involved in the deployment depend on the types of services provided by the data center. With the minimum scenario, the deployment system deals only with deployed application software. With this type of scenario, it controls versions of application software and 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.

3.3 IT Infrastructure and Management

3.3.1 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 wide scale 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 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 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 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.

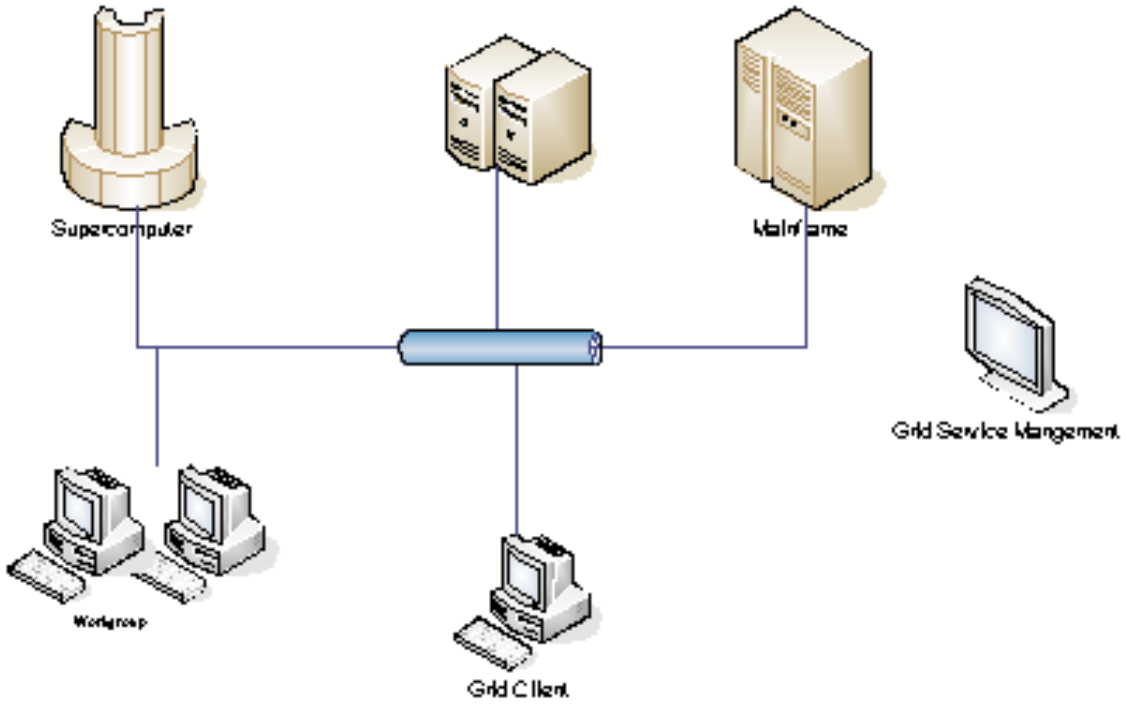

Figure 3. IT Infrastructure and Management

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.

3.3.2 Users

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*** plays 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 submission 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.

3.3.3 Scenarios

1) Generalized 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 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. The trader 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.

2) End User Device Deployment

The currency trader, happy with his model's performance, decides that he needs to do some On-Line Analytical Processing (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 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.

3) 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 service to be deployed on several machines at once. They propose the solution to Operations, which 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, that is 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.

3.3.4 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 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 Web 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 Web Service lifetime.

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

3.3.5 Use case 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 has actually been implemented so far and to what extent.

3.4 Utility Computing Model

3.4.1 Summary

Utility computing has 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.

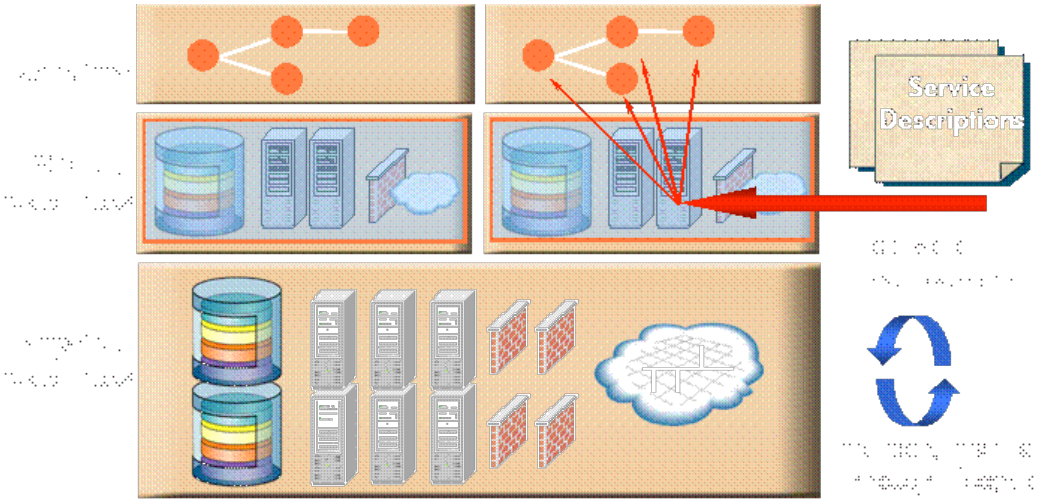

Figure 4. Utility Computing Model. Based on virtualization the model enables consolidation of data center resources. Automatic configuration, deployment, and lifecycle management play critical role.

3.4.2 Users

The customers for CDDLM in the utility computing model consist of the following participants with a 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 to 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.

3.4.3 Scenarios

1) 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 a 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 then un-deploying them as necessary.

2) 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 then un-deploying them as necessary.

3) 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 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 unplanned failures. In such a case, the service is re-deployed after it has crashed making sure that all dependencies are respected: services depending on the failed service are also restarted.

3.4.4 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.

3.5 Complex Business Service Use Case

3.5.1 Summary

Complex business collaboration—within this grid use case—is a 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 diverse resource sets, but also requiring various abstractions – high level for some users and intricate detail for others. The essence of this use case is the dynamic provisioning of existing services in order to undertake a meaningful business task. The services combine software (packaged and in-house), data and hardware. The dynamic provisioning is often repeated in many settings—countries, regions or centrally—warranting common configuration in order to standardize (driving configuration models that can built from existing work).

This deployment moves away from more traditional hardware and infrastructure software deployment into a model of dynamic system provision though dynamic deployment of existing resources.

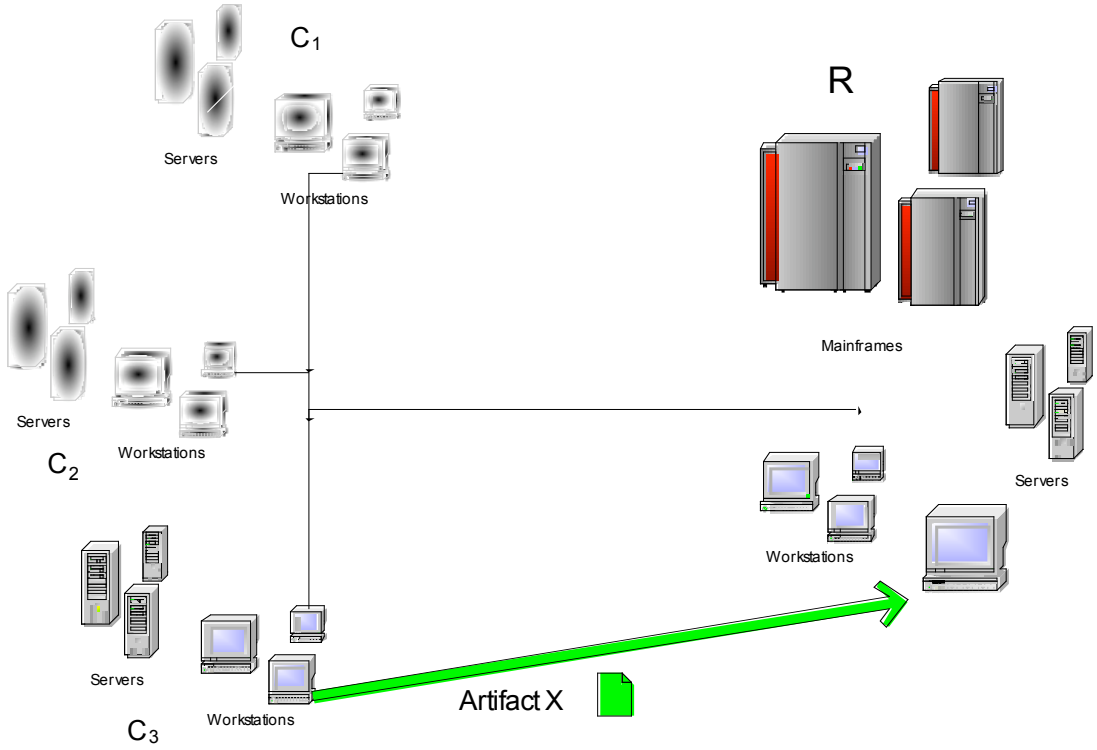

Figure 5. Complex Business Service Diagram

This diagram depicts four inter-related user groups in a global organization. A country group C_3 undertakes processes involving a mix of human, application and hardware resources to undertake a task. A regional group R, dependent on the country group, undertakes a similar task on completion of, and using an artifact X from, the country task. Two other country groups are depicted but not involved in the scenario, except in the provision of computational capacity (machines).

3.5.2 Users

The users in this case are the ***service consumers*** – monitoring the processes from either a technical or business perspective, or waiting for process completion (and resulting online and printed task reporting). The service consumer will often work with a ***business analyst*** in order to define the configuration from formal or ad hoc business requirements.

The ***System Manager*** will also use the configuration and life cycle management in order to both monitor the live systems and investigate problems.

3.5.3 Scenarios

1) Regional and Local Risk System Dynamic Deployment

This use case provides a generalized example of a global investment bank's risk analysis process (and is an exemplar for other organizations with global, regional, country structures that undertake common processing at differing organizational levels). This use case requires that services, often from different parts of the organization (trading datasets from business lines, market data from various internal and external sources), are combined in order to undertake the task at hand. The scenario involves a country-based trader who requires some risk analysis of a trading portfolio. This dictates the configuration of a trading data set, computational services and market data services. Added complexity to the scenario results from a regional risk manager who needs to carry out a similar analysis, but using data sets (composite portfolio) from all countries in the region and a more diverse computational and market data service resource set. Furthermore, the regional process can only start when a particular state is reached using country-based trader analysis.

In this use case, resource dependency and prioritization are prevalent. Adequate switching provision is required (e.g. if the regional process has started, it has priority usage over many of the country based resources). Methods must be in place to support predictive network, fault, and computational monitoring (e.g. Network Weather Service NWS [7][6]). This will be used, along with deployed service state information, by the deployment service to undertake re-deployment.

2) Test Deployment

The complexity and reliance on the risk analysis process requires that a test system is deployed. This system will consist of known input data and result sets. On deployment and resulting analytic completion, the known and calculated result sets can be compared. This testing may also be a dependency on production processing.

3.5.4 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. The resources are all used in 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 started via command line interfaces), services and legacy services. Thus, in the context of this use case, examples are:

1) Country-based computational services that use a packaged application with command line interfaces

2) Regional computational services that are in-house Web services

3) Regional Market data services are legacy Message-Oriented Middleware (MOM) systems with a Java connector API [8].

4) Country Market data that is held in a database.

The confluence of market and trading data sets, analytical code, packaged application and execution infrastructure (machine and network) is at the heart of grid computing. The deployment of services that provide this heterogeneous combination is critical to the managed business service oriented architecture.

3.6 Web Service Development Project

3.6.1 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 [9].

3.6.2 Users

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.

3.6.3 Scenarios

1) Development System Deployment

A Web service project is being developed by a software team for external customers. The application is being built as a Web 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. 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 of *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 souce code 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.

2) Live Deployment

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 has 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.

3.6.4 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 that 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.

3.6.5 Use case situation analysis

The scenario is a hypothetical descendent of an existing (and documented) Web Services project [9], with the development processes explored therein used to transform how projects are tested and deployed [10]. 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.

The remaining text will address the use case presented in Sections 3.1-3.6, by summarizing OGSA functional requirements, platform services utilization, and security and performance considerations.

3.7 Functional requirements for OGSA platform

Discovery and brokering: In presented scenarios it is assumed that resources (hardware and software) are ready for deployment. Discovery and brokering of the hardware resources is out of the scope of the deployment.

Deployment: Covered by this document.

Virtual organizations: Deployment is a part of the process to establish and maintain a virtual organization.

Policy: Deployment should have a set of policies for particular events such as system fault and recovery, execution errors or other exceptional situations.

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

Resource Virtualization*:* In order to allow the management of a pool of servers, it must be presumed that the Web 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 may be necessary in this type of environment for service owners 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, it is important for users to be able to ensure that service levels are met and that they have a means to troubleshoot any problems with the Web Service. Banking environments, in two of the above use cases, demand swift resolution to problems, so as to avoid money-losing situations through technology outages. Lifecycle management relies on the monitoring service for maintaining status of the service health.

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

Self-Healing*:* Required in some cases. The deployment service must be able to detect a failed state and redeploy.

Legacy Application Management*:* It will be commonplace for Web 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*:* User 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.

Authentication, Authorization, and Accounting (AAA). The requests for service deployment have to be authenticated, authorized and accounted for. We will comply with the standard security requirements and solutions for the Web Services.

Resource allocation. It is assumed that the CDDLM will contribute to the overall provisioning solution by means of deployment; however resource allocation is expected to be provided separately.

Fault Handling. Once a failure is discovered through monitoring, the CDDLM interacts with the fault handling modules in order to redeploy the service.

3.8 OGSA Platform Services Utilization

Service Interaction Services*:* In many cases services coexist and interact with each other within a shared infrastructure. In these cases, there must be provisions for services 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, that interaction with various data sets will be an integral part of the operations of Web Services within these and other scenarios.

Basic Execution Services*:* The use of a Execution Management is a prerequisite for enabling the P2P scenario and Complex Business 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.

3.9 Security considerations

The deployment service in these scenarios is able to configure secure environments for particular application execution appropriately, in line with specified policies, isolating and protecting the service and data from outside of VO boundaries or other user-specified systems boundaries.

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 against interference 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 require that even a data center operator cannot access the data.

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.

3.10 Performance considerations

The system should consider various aspects of the performance including:

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 with respect to the size and number of deployed components or number of servers and other hardware devices

The performance impact of deployment within each defined scenario is different. In the Generalized Web 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 High-Level CDDLM Architecture

The CDDLM is comprised of three main components: configuration language which is used to describe services to be deployed; basic services, which are used to request deployment; and component model, which supports lifecycle management of services. These three components are presented in Figure 6. In the rest of this section, we describe these three components in more detail. The two language specs (SmartFrog and XML-based) and the run-time binding spec will describe the Configuration Description Language. Basic Services and Component Model will be described in the corresponding specs.

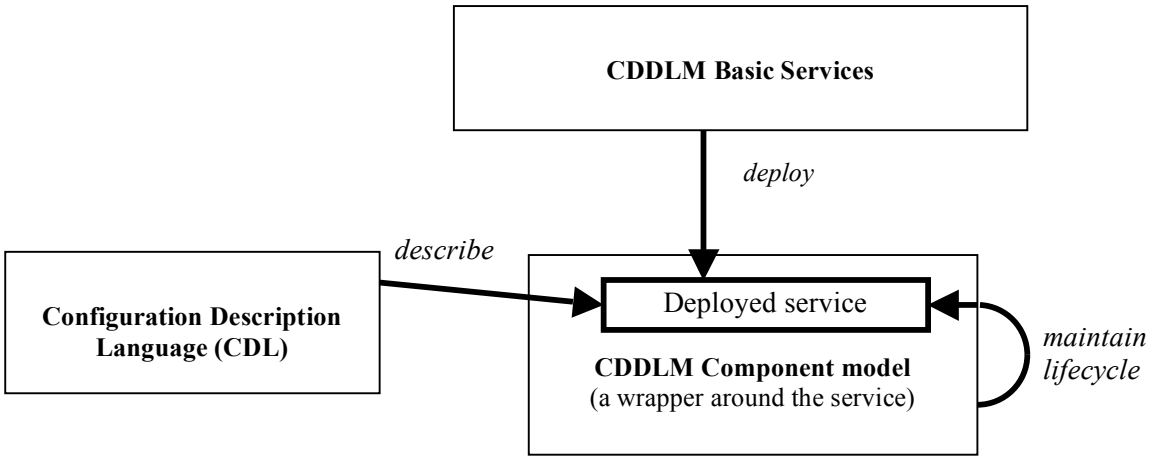

Figure 6. The CDDLM High Level Architecture.

Figure 7, describes how CDDLM relates to other components in Grid, in particular program execution and resource reservation. In particular, program execution is higher in the execution stack and deployment requests come from program execution to CDDLM basic services. These requests pass the deployment templates written in the CDL. Resources onto which services will be deployed are reserved prior to invocation of CDDLM, using other reservations protocols, such as WS-Agreement and the resources are identified when CDDLM services are invoked.

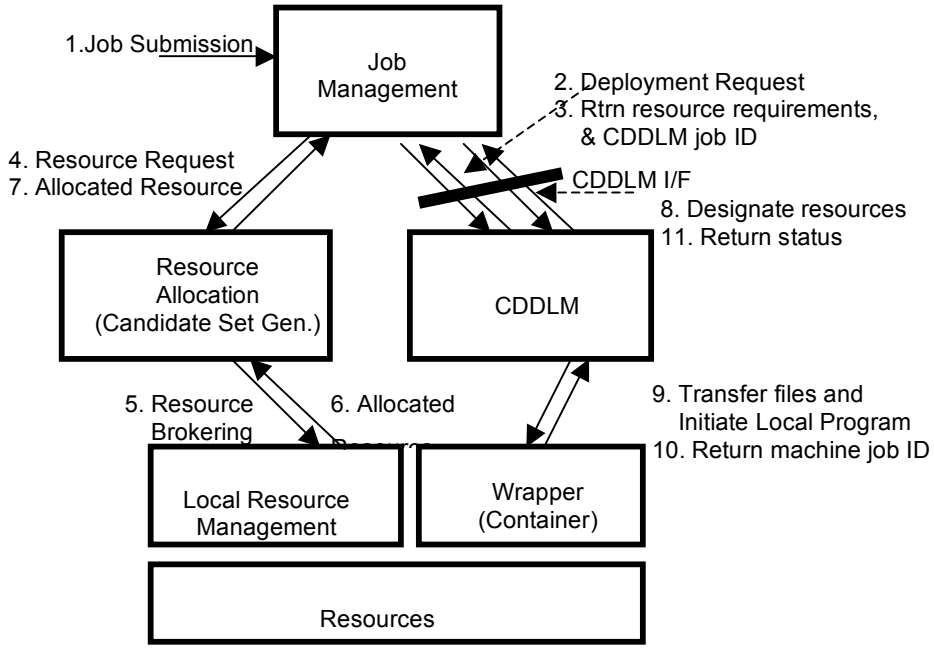

Figure 7. Interaction of CDDLM and Other Management Components.

4.1 Configuration Description Language Specification

There exist at least two versions of configuration description languages. One is SmartFrog-based and the other one is XML-based. The service configuration is expressed in the template written in the CDL. Each configuration consists of translation from the front-end language, including 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 will be resolved during the deployment or initialization of the execution.

4.2 Basic Services Specification

The primary CDDLM service is deployment. The deployment service accepts the configuration descriptions of target services expressed in CDL, it parses it, verifies consistency, resolves component references as far as it is resolvable and composes a package ready for deployment. It then invokes its sub-components responsible for instantiating the target services and conducts the target service through its lifecycle: installs, initiates, and starts. Should the service fail it will be restarted.

CDDLM offers only deployment service, even though it also supports configuration and lifecycle functionality. The main reason why CDDLM does not also offer configuration and lifecycle is that these areas are covered by other standards bodies, in particular WSDM. In addition, both configuration and lifecycle management functionality are application-specific. Configuring a service can only be done with the assistance of the service in question. We strongly believe that configuration is really a management function and should be treated as such, outside of the scope of CDDLM. The existing configuration and lifecycle functionality that CDDLM offers is primarily in support of deployment. That means that clients can request complex configurations that CDDLM will execute; similarly, should any of the deployed services fail, CDDLM will detect failures, it will resolve dependencies and restart the failed and all other depending services as appropriate.

4.3 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 Web 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 Web 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 Related GGF and other Standard Bodies Working Groups

5.1 GGF OGSA

From OGSA-WG Charter: “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).”

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 is as follows. These are regarded as basic conditions standard component to be positioned at the OGSA architecture.

1) Operating within the Grid framework (OGSI, WSRF, or whatever gets adopted at the time of the CDDLM implementation).
2) The work fit into the Architecture as defined by the OGSA document.
3) Engage actively with us and other OGSA-related WGs.
4) Use OGSA as an “adjective”.

5.2 GGF OGSA Basic Execution Services

Basic Execution Services is one of the basic components of OGSA; it includes functionalities required for program execution. Detail discussions are still undergoing in the working group, but functionalities of CDDLM should be positioned and used appropriately in the architecture.

5.3 OASIS WSDM

CDDLM is related to OASIS/WSDM (Web Services Distributed Management), but it is sufficiently focused on the deployment that. 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 Web service, including its deployment and configuration on provisioned hardware. In that light, only deployment and configuration are covered by the CDDLM. In order to handle physical IT resources through Web 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 resource model), and W3C (the Web services architecture committee). Also liaison with other OASIS TCs, including the Security TC and other management oriented TCs.

5.4 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-Provisioning 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.

5.5 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).

5.6 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.

5.7 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 Web 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.

5.8 GRAAP

The Grid Resource Allocation Agreement Protocol (GRAAP) deals with the resource allocation of distributed resources within the Grid. This is accomplished through a "Super-Scheduling" service that encompasses various resources managed by different schedulers in use within a Grid. The Grid Resource Allocation Agreement Protocol Working Group addresses the protocol between a Super-Scheduler (Grid-Level Scheduler) and local Schedulers necessary to reserve and allocate resources in the Grid as a building block for this service. The CDDLM and GRAAP complement each other to support the provisioning. GRAAP represents the left side of Figure 7.

5.9 JSDL

The Job Submission Description Language (JSDL) provides a specification for an abstract standard Job Submission Description Language independent of language bindings. Compared to JSDL, CDDLM deals with the deployment aspects of which JSDL is agnostic. On the other hand, JSDL is aware of the scheduling specifics of the popular batch systems. In a similar way to GRAAP and OGSA Basic Execution, the JSDL and CDDLM complement each other in contributing to the overall provisioning model.

6 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 (http://www.redhat.com/), 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 (http://www.altiris.com/) or Rembo (http://www.rembo.com/), 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 (http://www.lcfg.org/), 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 their 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 [10], 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.

The goal of CDDLM is primarily related to the scope of GGF and more specifically OGSA architecture. However, there is no reason that the high level CDDLM architecture cannot be applicable to other architectures, such as primarily p2p JXTA architecture, or similarly SmartFrog-based architecture. However, these explorations are beyond the scope of the documents and specifications that this working group will produce initially.

7 Future Work

1. Complete all planned Specs
2. Link with OGSA (get feedback on the foundation document, follow-up specs)
3. Maintain relationship to other GGF groups, in particular DAIS-WG, CMM-WG, GRAAP-WG, JSDL-WG, OGSA-WG Basic Services, WFM-WG, EG-RG. DRMAA-WG, and CGS-WG.
4. Establish relationship to standard bodies outside of GGF, in particular DMTF Applications and Utility Computing, OASIS WSDM and WSRF, and DCML.
5. Establish relationship to industries and academia
6. Come up with at least two interoperable reference implementations

7. Visualization and sensors for deployed services
8. Provide examples of configuration profiles

8 Security Considerations

Security issues have been broadly covered through the document

9 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
Mail Stop 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
Mail Stop 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
Mail Stop HPLB
Filton Rd.
Stoke Gifford
Bristol BS34 8QZ
United Kingdom
Phone: +44 117 312 8728
Email: peter.toft@hp.com

10 Contributors

We gratefully acknowledge the contributions made to this specification by Jem Treadwell, Hiro Kishimoto, and Greg Newby, the GGF Editor.

11 Acknowledgements

This work was supported in part by HP, Softricity, and NEC.

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.

Full Copyright Notice

Copyright (C) Global Grid Forum (2002, 2005). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

References

[1] Treadwell, J. (ed.) Open Grid Services Architecture Glossary of Terms. Global Grid Forum OGSA-WG. GFD-I.044, January 2005. http://www.ggf.org/documents/GWDI-E/GFD-I.044.pdf.

[2] Configuration Description, Deployment, and Lifecycle Management SmartFrog-Based Language Specification, http://forge.gridforum.org/projects/cddlm-wg/document/CDDLM_SmartFrog_CDL/en/1.

[3] Configuration Description, Deployment, and Lifecycle Management (CDDLM) XML-Based Language, Document in preparation

[4] Configuration Description, Deployment, and Lifecycle Management (CDDLM) Component Model, Document in Preparation.

[5] Configuration Description, Deployment, and Lifecycle Management (CDDLM) Deployment API, Document in Preparation.

[6] 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

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

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

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

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

[11] SmartFrog reference manual
http://www.hpl.hp.com/research/smartfrog/papers/sfReference.pdf.