An Initial Assessment of CDDLM

A. Dantas, F. Santos, G. Germoglio, M. I. Oliveira, W. Cirne, F. Brasileiro
Universidade Federal de Campina Grande {ayla,flavors,guiga,iury,walfredo,fubica} @dsc.ufcg.edu.br
S. Rafaeli, K. Saikoski, D. Milojicic
Hewlett Packard
{sandro.rafaeli,katia.saikoski,dejan.milojicic}@hp.com

Abstract
As Web Services and Service Oriented Architectures are adopted, it is increasingly important to have standard and interoperable means to deploy and configure Web Services. Today, most services have ad-hoc, proprietary means for configuration and deployment, which makes them hard to integrate, manage, and support. Within the Global Grid Forum, HP, NEC, and Softricity have been developing a standard for Configuration Description, Deployment, and Lifecycle Management (CDDLM). In order to prove its feasibility, they have each developed a reference implementation based on their prior experience in this area. This paper describes an initial assessment of this standard based on the development of an independent reference implementation of CDDLM and on its use in the OurGrid project.

Keywords
Grid standards, deployment, configuration and management.

1. Introduction
Software deployment is defined as the delivery, assembly, and maintenance of a particular version of a software system at a site [16]. This is a complex process which addresses some issues, such as: (i) the complexity due to many tasks involved (e.g., distribution, configuration, management) and interdependency between multiple components deployed, making manual deployment inadequate (time consuming and error-prone); (ii) scalability in targeting a large set of nodes, e.g. for data centers, Grids and planetary-scale testbeds, such as PlanetLab, where each may require installing and configuring a whole new environment; and (iii) the customization of each deployment, e.g., there exists dependency of the configuration process on individual target machines.

The GGF Configuration Description, Deployment and Lifecycle Management (CDDLM standard) offers Web Services-based automation for the deployment of systems. Three reference implementations are being developed by the companies involved in the CDDLM specification: Softricity, NEC and HP. Our task as a research
group on Grid computing at UFCG was to build an independent prototype to understand the standard and evaluate it, providing feedback for the authors of the specifications and for its users as an external group. For these evaluations, besides observing general points, we have analyzed CDDLM use considering the deployment and management of OurGrid, a grid solution that has been developed by our group.

The contribution of this paper is three-fold: (i) to illustrate the use of the standard; (ii) to present the advantages observed in experimenting with CDDLM and (iii) to raise some issues that should be addressed while using CDDLM.

2. Related Work
CDDLM provides a set of specifications for automating the deployment and management of systems using Web Services. It is based on the Smart Framework for Object Groups (SmartFrog) [8], developed by HP Labs Bristol. This framework is used for deploying applications in a distributed environment. SmartFrog presents an human-readable language intended for configuration description and deployment. This language has support for some features such as configurations extensions between components, references between parts of configurations and the definition of properties resolved only after deployment. In order to have a SmartFrog environment able to deploy components, the system administrator has to install its daemons in all environment machines. After that, the user has to create a description file specifying the components to be deployed and where they should run. There are two main differences between CDDLM and SmartFrog: the first is that CDDLM uses an XML-based language (CDL) for the description of components, and the second is that the SmartFrog communication infrastructure is based on RMI protocol and CDDLM uses the SOAP protocol to exchange messages.

There exists another competing standard, Solution Deployment Descriptor (SDD) [2], within OASIS standard organization. The purpose of the SDD OASIS Technical Committee is to define an XML schema to describe the characteristics of an installable unit (IU) of software that are relevant for core aspects of its deployment, configuration, and maintenance. This schema was referred to as the IUDD (Installable Unit Deployment Descriptor) schema and is now called SDD schema. SDDs are intended to describe the aggregation of installable units at all levels of the software stack. The resulting XML schema shall be partitioned to allow for layered implementations. These implementations should cover the range of applications from the definition of atomic units of software to complex, multi-platform, heterogeneous solutions. SDDs are intended to provide a consistent model and semantics to address the needs of all aspects of the IT industry dealing with software deployment, configuration, and lifecycle management. Compared to CDDLM, this is a less mature standard, with significantly slower pace of development due to the wider industry participation. Because of the reference implementations of CDDLM, such as the one we have developed, CDDLM stands good chances to become one of the first formally approved GGF standards, demonstrating independent interoperable reference implementations of adopted specs.

There are also efforts such as OASIS DCML [1] and DMTF Applications WG [3]
that address deployment, but that are not primarily focused on it. Finally, there are numerous products that support deployment, including HP’s Radia, Altiris’ RDP, IBM’s Tivoli provisioning, REMBO, etc. These are proprietary solutions that do not comply with WS-based standards.

Another related work is GLARE [12]. GLARE is a framework that provides dynamic registration, automatic deployment and on-demand provision of application components (called activities) that can be used to build Grid applications. GLARE is designed and implemented as a distributed framework that stores information about application components. The main difference between this work and CDDLM is that CDDLM is basically a set of specifications intended to become standards instead of a framework. Besides, CDDLM focuses on the description of the configuration of components and on the definition of services for their deployment on the Grid and for the management of their lifecycle.

Talwar et al [13] discusses several approaches to service deployment. The authors compare manual, script-, language-, and model-based deployment solutions but in a much more generic fashion. In our work we analyze a specific approach for automatic deployment (CDDLM) pointing out advantages and issues to be considered. Another difference is that [13] has not explored Web Services standards for deployment yet, which is a planned future work.

3. CDDLM Overview

CDDLM is likely to become the standard for WS-based service deployment and lifecycle management. With CDDLM, one can specify configurations of components that can be deployed in a distributed environment and that can be managed using open standards. In addition, dependencies between the components can be expressed within CDDLM and notifications can be triggered to modify their behavior.

The CDDLM standard is comprised of three specifications:

- a Configuration Description Language (CDL) [14], which is an XML-based language that declaratively describes the configuration of a software system in terms of a hierarchy of components;
- a Component Model [11], which defines the components that comprise a system, and;
- a Deployment API [10], which is a set of interfaces to deploy and manage the lifecycle of a system described in a CDL document.

The Deployment API defines two deployment services: the Portal and the System. The Portal is a service that enables one to create and/or discover a System service. The System includes operations for controlling the lifecycle of components (deployable objects): Initialize, Run, Terminate and Destroy. The Initialize operation is used to deploy a configuration of components. The Run operation is invoked to start the execution of the components that have been deployed. The Terminate operation is used to finish the execution of components. Finally, the Destroy operation is used to undeploy the components. Besides providing these operations, the Portal and the System services must also follow the Web Services Resource Framework (WSRF) [4] and provide operations such as GetResourceProperty and
The CDL language is used to describe components that follow the CDDLM Component Model specification [11] and their properties. An example of system configuration described as part of a CDL file is shown next:

```xml
<cdl:cdl>
  <cdl:system>
    <WebServer>
      <port>80</port>
      ...
    </WebServer>
  </cdl:system>
</cdl:cdl>
```

The example above describes a system composed by the WebServer component. When the System initialize operation is invoked, the component is deployed and configured to use 80 as its port attribute value.

A CDL document may also contain special constructions such as extends, ref and lazy. The extends construction allows reuse of component definitions within other components. The ref represents a reference to a particular property in a CDL document. The lazy construction is used to indicate that a certain part of a CDL document will only be resolved after deployment. In the following we show an example of the use of these constructions:

```xml
<cdl:cdl>
  <cdl:configuration>
    <WebServer>
      <address>
        <hostname cdl:lazy="true"/>
        <port>8080</port>
      </address>
    </WebServer>
  </cdl:configuration>

  <cdl:system>
    <Tomcat cdl:extends="WebServer">
      <maxThreads>200</maxThreads>
      <servername cdl:ref="/Tomcat/address/hostname"/>
    </Tomcat>
  </cdl:system>
</cdl:cdl>
```

The CDL above describes a Tomcat component that reuses a component template definition (WebServer). As we can see, there are two main parts in this CDL: the configuration and the system. The components to be deployed are placed in the system part and can reuse many of the definitions made in other components definitions or in the configuration section of the CDL. In this example, the Tomcat component, through the extends construction, reuses the address property from WebServer,
including hostname and port definitions. As we can observe, the hostname element presents a lazy attribute, which indicates that just after deployment this property value will be defined. It can store, for instance, the hostname dynamically chosen for the deployment. Besides being used in the address definition, the hostname value can also be used in other points of a component configuration. In such cases, we can use the ref attribute. In the example above, ref is used to indicate that the servername property from the Tomcat component will have the same value obtained for the hostname of the address property.

Besides being used for components configuration, CDL is also used to describe the workflow pattern for lifecycle operations on the components. By using the sequence, flow, switch and reverse constructions we describe the order for the invocation of these operations on the components. The sequence indicates that certain lifecycle operations must be performed in the sequence used for the components definition. The reverse means the opposite order followed using sequence. The flow indicates that the order does not matter. The switch operation indicates a certain order to be followed according to conditions stated in the CDL document.

The lifecycle operations are invoked on the components through service Endpoint References (EPR) that must be provided for each deployable component described in a CDL. Each endpoint must be WS-ResourceFramework (WSRF) compliant and must provide a set of operations described by the Component Model specification [11]. Besides property handling operations defined by the WSRF specification and those regarding lifecycle, there are also WS-Notification specific operations such as Subscribe and GetCurrentMessage. These services are generally invoked by the System, which is the service responsible for the management of its underlying components.

Considering this set of services (Portal, System and Components services), we can identify some scenarios for human interactions with them. Administrators interested in several deployments on a set of machines can use a Portal endpoint and request the creation of a System with a given configuration of components described in a CDL. Users can benefit from the deployed components and can request from machines administrators or from an automatic system an specific environment configuration. CDLs with components frequently used can be made available to make easy the process of creating a configuration. If new components must be present in a configuration, developers must be involved in this process in order to implement the service to deploy and manage a given component lifecycle following the Component Model specification [11].

4. Implementing and Using CDDLM

Based on our own experience in developing an independent prototype of CDDLM, we can summarize the development of the CDDLM reference implementation with the development of a CDL parser and of two WSRF-based services, the Portal and the System. For the implementation of the services we used Apache Web Services tools [5] as the underlying development environment to:
(i) speed up the development of the necessary infrastructure, since there was a limited time before final specs were released; and,

(ii) conform to the CDDLM specs that required support for the Web Services Distributed Management [15] stack, which is already provided by Apache.

In order to try our CDDLM implementation, we have deployed OurGrid [7][6] components. OurGrid is an open, free-to-join, cooperative grid in which labs donate their idle computational resources in exchange for accessing other labs’ idle resources when needed. One of the OurGrid’s components is the UserAgent (UA) service, which represents a virtual machine over a computational resource. Each UserAgent is a Java RMI service responsible for executing tasks of jobs submitted to the grid. Several UserAgents can be installed on the same machine making it able of executing more than one task at the same time. Therefore, we have prepared the deployment of 3 UserAgents per machine and we have deployed them over several nodes.

From this experience, we concluded that for using CDDLM, someone needs to:
(i) create WSRF-based services representing components (in our case the components were the UserAgents) that follow the interface proposed by the Component Model specification; (ii) implement in these services the properties configurations, lifecycle operations, and the points where notifications should be sent; (iii) prepare packages with such services and make them available from URIs that can be accessed by the CDDLM deployment engine (Portal and System services); (iv) prepare a CDL document with the components configurations and references to components code bases (URIs for the components services in our implementation); and (v) access a Portal service, create a System and submit the created CDL document.

By using such scheme, we reach with CDDLM a standardized way of deploying and managing properties and lifecycle of systems spread around the world. This happens due to the use of a WS-based solution. Besides, as each component is accessed by an EPR (Endpoint Reference), its properties can be queried and changed when necessary. Other visible advantages are the ability to express relations between components and the possibility to reuse component definitions using the CDL special constructions. This ability improves the configuration and management of systems composed by several elements with inter-dependencies, or with properties only defined after deployment time.

However, there are some issues to be considered while adopting CDDLM that we discuss next. These issues were elicited based on our experience with the development of both the reference implementation and the components for the OurGrid solution.

**Learning Curve**

The first requirement to use CDDLM is to prepare a CDL document. This CDL document can vary from a simple definition to a case that presents specifications of dependencies between components and their shared properties, besides definitions of notifications. As CDL is an XML-based language and it presents rich constructions, it is not easy to read and edit it. Therefore, tools that aid in editing CDL documents and
also help in identifying configuration errors would be of great value.

**Development Effort**

Although there are tools to aid in the creation of services for each needed component, there is still a great effort in the development of these services. The first challenge is to deal with tools that are not sufficiently mature, especially because the standards on which CDDLM is based are very new and still evolving. The input for such tools is generally a WSDL file describing the service. Writing a WSDL is an error-prone task. Some XML editors provide support in avoiding common errors. However, for more specific errors it is still difficult to debug problems in the services definitions. Besides the effort for creating WSDL files, there is also the effort required for implementing code for the generation of notifications at specific code points. While developing a component service for the UserAgent component in particular, we have noticed that the development effort was bigger than using script-based approach if the deployment to be managed was only intended for the UserAgents. This happens because UserAgents deployment is simple and it is usually done for one administrative domain (a single site).

However, if we consider the deployment of UserAgents and also other services required by the tasks submitted to the grid, or if we consider a case on which an administrator needs to manage several OurGrid sites, the effort necessary to use CDDLM could be justifiable. It is important to notice that the effort for the implementation of services for components tends to reduce as built-in components for common services (such as in SmartFrog [8]) begin to be implemented and made available.

**Need for parallelism**

To evaluate the performance of our CDDLM reference implementation, we have performed several deployments of a CDL representing three UserAgent components from OurGrid and one component that is a Web service that monitors the status of these components. This CDL is available at [http://www.lsd.ufcg.edu.br/~ayla/cddlm/ua.componentandmanager.cdl](http://www.lsd.ufcg.edu.br/~ayla/cddlm/ua.componentandmanager.cdl). We have used different machine configurations varying from Pentium 4, 1.8GHz, with 512M of memory to Pentium 4, 3GHz, 1GB of memory. All of them were located in the same local network.

On average, the time for creating a System and initializing the four components was 31 seconds per machine during a period with small network traffic. This time regards sequential deployment. The average total time for deploying ten machines was 310 seconds.

![Figure 1: System Deployment Time](http://www.lsd.ufcg.edu.br/~ayla/cddlm/ua.componentandmanager.cdl)
Performing these deployments in parallel takes considerably less time. Figure 1 illustrates System sequential and parallel deployment times with the same CDL document varying the number of nodes. For these initial experiments, the average time for the System parallel deployment in 10 nodes is 83 seconds. Therefore, we have concluded that it is interesting that brokers for CDDLM deployment services provide parallelism.

In CDL it is possible to determine the order of lifecycle operations in components from the same System using constructs such as flow and sequence. With sequence we define the exact order, and with flow, none is defined. Due to this fact, in our implementation, when a lifecycle operation is defined as flow, we perform it in parallel. In Figure 2 we illustrate the times for performing all lifecycle operations on components using flow and sequence and varying the number of UA components. As we can see, providing parallel deployment with flow was a good approach to be followed. However, we must notice that not all deployments can be performed in parallel because sometimes there are interdependencies between components.

Therefore, the flow construct must be used with care.

Resource Provisioning

The CDDLM specs do not directly cover resource provisioning in several nodes, but it is a point that must be addressed for real deployments. A possible alternative is to implement a CDDLM broker that is able to request deployment nodes from resource providers and to automatically create CDL documents that would express the number of resources obtained. The CDDLM specification considers the possibility of using a simpler approach of deploying new Systems and their underlying components on the same machine where the used Portal is deployed. While using CDDLM in OurGrid, we have provided a simple distributed resource management scheme. In this scheme, when a request for the creation of a System in a Portal is performed using a hostname hint, a verification is performed to verify if the hint corresponds to the Portal host. If so, the System and its components will be deployed in the same machine of the Portal. If not, the System will use a directory service to find a
registered service identified by the hint to perform the external deployment. If it does not find it, the System will be locally deployed.

Security

Because the Portal and the System allow fast changes in several nodes, security is an important issue to be addressed. The deployment API specification states that “Only callers with the relevant rights MUST be allowed to deploy systems”, however it does not specify how these security requirements must be fulfilled. Since security was not our focus in the current stage of our CDDLM reference implementation, we have designed a simple (minimal) security architecture basically focusing on authentication and authorization of the services’ users and also on the verification of services code bases to avoid the deployment of malicious code. We consider that the development of such mechanisms both in CDDLM reference implementations and also in components used by them can avoid several problems regarding security.

5. Conclusions and Future Work

Considering alternative solutions, such as the ones presented in Section 2, and the observed advantages and issues of CDDLM, we still believe it is a good approach for deployment and management of components as it is an effort in the standardization of the mechanisms for deployment, configuration and lifecycle management. Besides, the fact of being WS based may make its adoption easier as industry is migrating to such approaches. Some of CDDLM’s drawbacks will be overcome in the future as its underlying specs and tools become more mature and when new supporting tools are provided, but the issues considered above should be addressed.

As future steps, we will continue updating our implementation according to newly released versions of the specifications and test plans. We will also work on interoperability with other implementations. In addition, we plan to explore CDDLM use for autonomic computing, especially focusing on self-management [9] of systems.

ACKNOWLEDGMENT

This work has been partially developed in collaboration with HP Brazil R&D. We are indebted to Steve Loughran, Jun Tatemura and Stuart Schaefer, members of the CDDLM-wg, for answering our many questions through the mailing lists and conference calls.

References


