

CDDL M Configuration Description Language Specification, Version 1.0

Status of this Memo

This document provides information to the community regarding the specification of the Configuration Description Language. Distribution of this document is unlimited.

Copyright Notice

Copyright © Open Grid Forum (2006-2007). 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 Grid services, their deployment, and managing their lifecycle. A major part of this framework is a language in which to describe the components and systems that are required. This document, produced by the CDDL M working group within the Open Grid Forum (OGF), provides a definition of the XML-based configuration description language and its requirements.

Table of Contents

Table of Contents	2
List of Figures	4
1 Introduction	5
1.1 CDDL M Specs	5
1.2 Notational Conventions	6
2 CDDL M-WG and the Purpose of this Document	6
3 Configuration Description in CDDL M	6
3.1 The CDDL M Framework	6
3.2 Use of Configuration	8
4 Requirements for the Language	9
5 Configuration Data Model	9
5.1 Property Lists	9
5.2 Configuration Description	10
6 Document Structure	11
6.1 Types	11
6.2 Configuration	11
6.3 System	11
6.4 Import	11
6.5 Documentation	11
7 Configuration Description	12
7.1 Property List Name	12
7.2 Prototype References	12
7.2.1 Reference Description	12
7.2.2 Reference Resolution	13
7.2.3 Resolvable Prototype Reference	14
7.2.4 Example	15
7.3 Value References	17
7.3.1 Reference Description	17
7.3.2 Resolution	19
7.3.3 Resolvable Value Reference	19
7.3.4 Prototype Resolution and Value References	19
7.3.5 Example	23
7.3.6 Expression	24
7.4 Schema Annotations	25
7.4.1 Property Type Declarations	25
7.4.2 Property Value Occurrence Constraints	25
7.5 Laziness Annotations	26
7.5.1 Lazy Value Resolution	26
7.5.2 Lazy Properties	27
7.5.3 Lazy References	28
7.6 Parameterization	28
8 System Description	29
9 Import	29
10 Documentation	30
11 Security Considerations	30

12 Editor Information 31
13 Acknowledgements..... 31
Intellectual Property Statement..... 31
Full Copyright Notice..... 31
References..... 32
Appendix A: XML Schema..... 32
Appendix B: Example 36

List of Figures

Figure 1: Use of CDL in the CDDLM Framework..... 8
Figure 2: Functionalities of the Configuration Description Notations..... 10
Figure 3: Example of path expression..... 19

1 Introduction

Deploying a complex, distributed service presents many challenges related to service configuration and management. These range from how to describe the precise, desired configuration of the service, to how we automatically and repeatably deploy, manage and then remove the service. This document addresses the description challenges, while other challenges are addressed by the follow-up documents. 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. Addressing these challenges is highly relevant to Grid computing at a number of levels, including configuring and deploying individual Grid Services, as well as composite systems made up of many co-operating Grid Services.

1.1 CDDL M Specs

There are five documents created by the CDDL M working group. They are described in the text below.

The CDDL M Foundation document [**CDDL M**] 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 working group with other working groups as well as with other related efforts in the industry.

The SmartFrog Language spec [**SF-CDL**] 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. Whereas CDL is primarily intended for machines, SmartFrog is for humans and translated to CDL.

The CDDL M Configuration Description Language (CDL) is an XML-based language for declarative description of system configuration where system consists of components (deployment objects) defined in the CDDL M Component Model. [**Component Model**] The Deployment API [**CDDL M-API**] uses a deployment descriptor in CDL in order to manage deployment lifecycle of systems. XML-CDL 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. XML-CDL 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 CDDL M 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 CDDL M 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 also adds the ability to resolve references within the deployed system, enabling remote callers to examine the state of components with it.

1.2 Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [RFC2119].

The following namespaces are used in this document:

xsd	http://www.w3.org/2000/10/XMLSchema
cdl	http://www.gridforum.org/namespaces/2005/02/cddlml/CDL-1.0

XML document structure is described in an informal syntax where

- <!-- description --> is a placeholder for elements from some "other" namespace (like ##other in XSD)
- Characters are appended to elements, attributes, and <!-- descriptions --> as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more). The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to the "?", "*", or "+" characters.

2 CDDLML-WG and the Purpose of this Document

The CDDLML working group 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 working group is to gather researchers, developers, practitioners, and theoreticians in the areas of services and application configuration, deployment, and deployment life-cycle management and to explore the community need for a broader effort in this area. The target of the CDDLML working group is to come up with the specifications for CDDLML a) language, b) component model, and c) basic services (deployment API). This document represents a CDDLML language specification based on XML.

Sections 3 and 4 are informative sections that describe concepts and models in CDDLML and CDL. Sections 5 to 10 are normative and specify the CDL.

3 Configuration Description in CDDLML

3.1 The CDDLML Framework

This subsection provides an overview of the CDDLML framework, where CDL is used.

In a Grid environment, a user executes an application (or job) with resources allocated on demand. Such resources can be hardware, software, or any other objects which are managed to make the application executable. Once required resources allocated, they need to be configured appropriately so that they are ready to use by the application. The CDDLML framework provides ways to describe configuration of resources and manage

their deployment lifecycle. In this framework, configuration of the entire set of resources for the application execution is managed as a *system*. To make an application (or job) ready to execute, a system must be made (and kept) ready. A system consists of *components* which encapsulate resources as configurable units.

- *Components*: A component is the smallest deployment unit within the CDDL M framework.
 - *Component lifecycle*: A component is treated as an atomic object that has a single state of deployment (such as initialized, running, and terminated) although it may internally consist of multiple pieces of hardware or software. The deployment lifecycle of a component is represented and managed as a state machine that defines transition among such states. CDL itself does not assume any deployment lifecycle model (or state model). Instead, Component Model specification document [**Component Model**] defines a specific lifecycle model.
 - *Properties*: A component has a set of *properties*. The user can configure the component by giving values of these properties. CDL provides ways to define properties and assign their values.
- *System*: A system is a set of components required to be configured for execution of an application.
 - *System lifecycle*: A state of a system represents the overall states of components in the system. A user can manage (e.g., operate and monitor) a single state of the system in order to control states of the components. For example, in order to make components *running* (i.e., available for the application to use), a user makes a system *running*. When all the components become *running*, the system is considered to become *running*. Relationship between a system lifecycle and component lifecycles is specified in Component Model specification.
- *Deployment Service*: A deployment service enables a user to manage the deployment lifecycle of a system. The service supports (1) lifecycle management operations that control system lifecycle (i.e., state transition) and (2) lifecycle monitoring services that let users know the deployment status. A system configuration described in CDL is passed from the user to the service. The concrete API of this service is specified in Deployment API document [**CDDL M-API**].

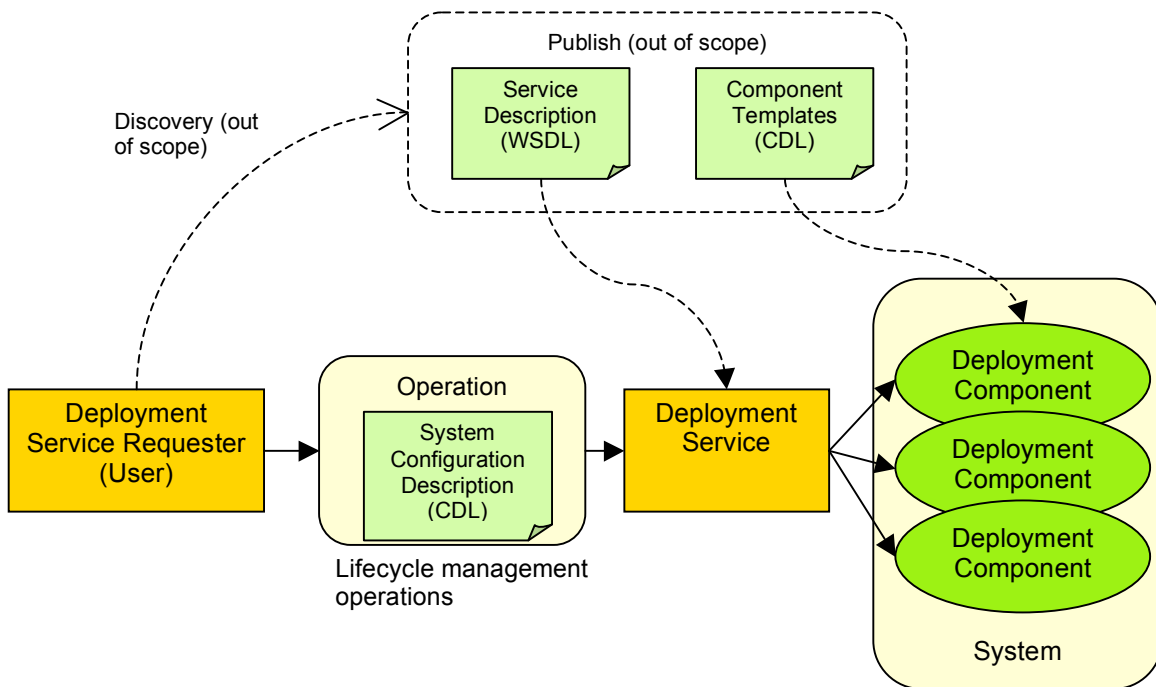


Figure 1: Use of CDL in the CDDL Framework

3.2 Use of Configuration

Figure 1 illustrates how configuration description is used in this deployment services framework.

A component provider provides a configurable component, an implementation of a CDDL deployment component that encapsulates and manages resource configuration. The user's requirement on a deployment component should be written in CDL as a language component. The component provider describes a component template in CDL so that the user can refer to these templates and describe a language component appropriately.

A component template can include the following information:

- Property names
- Property types
- Default values of properties
- Whether property values are required or optional
- Properties that are dynamically assigned in deployment time

The CDDL framework (hence CDL) does not specify:

- How to describe implementation of the deployment component (i.e. how resources are configured within a component)
- How to publish and discover component templates
- How to allocate or reserve resources

A deployment service provider provides service endpoints through which components are configured as a system. It accepts a system configuration description that refers to component templates.

A deployment service user describes a system configuration description in CDL, which is given to the deployment service. To specify how configurable components should configure resources, a system configuration description can contain:

- References to component templates
- Values of properties
- Value dependencies among properties

The user can also describe partial system configuration description as templates for convenience. Since a user typically execute an application repeatedly with different configuration in Grid environments, such templates are useful to reuse description across multiple deployments. A complete system configuration description can refer to these templates.

4 Requirements for the Language

The CDL requires the following properties to support the CDDLM framework described above.

- **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 configuration description should provide enough information for a deployment service to dynamically generate correct sequencing of operations across distributed resources for deployment and lifecycle management.
- **XML-based.** The language should be XML-based. A well-formed configuration description should be a well-formed XML document.
- **Dynamic configuration.** The language should be able to be applied to dynamic configuration use cases, where some configuration parameter values can not be determined before deployment time.
- **Referenciability.** The language should be able to specify dependencies between configuration parameter values so that the deployment service can manage consistency between parameters.
- **Composability.** The complete system configuration description should be composed by combining multiple descriptions that may be provided by multiple component vendors. The language should provide a way to define a new composed configuration description by referring to existing descriptions.
- **Security.** The security requirement in the deployment service framework should be achieved by incorporating Web Service/Grid/XML security standards into a configuration description.
- **Extensibility.** The language should allow the user to add extensibility elements in a description.

5 Configuration Data Model

5.1 Properties and Property Lists

Data required to configure a component is given as an ordered list of properties, each of which has a name and a value. A list may contain duplicate properties, i.e., properties with the same name. A property value may be a property list so that a nested structure of

properties can be constructed. The CDL does not assume any semantics on the order of properties in a list, but a list in CDL is processed in an order-sensitive way so that an application can make use of the order of properties as configuration information. Properties and property lists are represented as XML data defined with a domain specific schema. A property is an XML element whose name and content represent the name and value of the property, respectively. A property list is an XML element whose children are properties. A property and a property list may have any type of attributes. These attributes are not regarded as properties but as annotations to properties. CDL introduces such attributes (e.g., value references) inserted in property lists. The following is an example of a property list:

```
<WebServer>
  <hostname>example.com</hostname>
  <port>80</port>
  <maxClients>150</maxClients>
  <applicationServer>app1.example.com</applicationServer>
  <applicationServer>app2.example.com</applicationServer>
</WebServer>
```

5.2 Configuration Description

In order to dynamically generate property lists for components from multiple sources, which may be given from different organizations in different timing, CDL provides XML notations for the following functionalities:

- Unique naming of property lists (hence, naming of component templates)
- Inheritance of property lists (hence, inheritance of component templates)
- References that define data dependency between properties

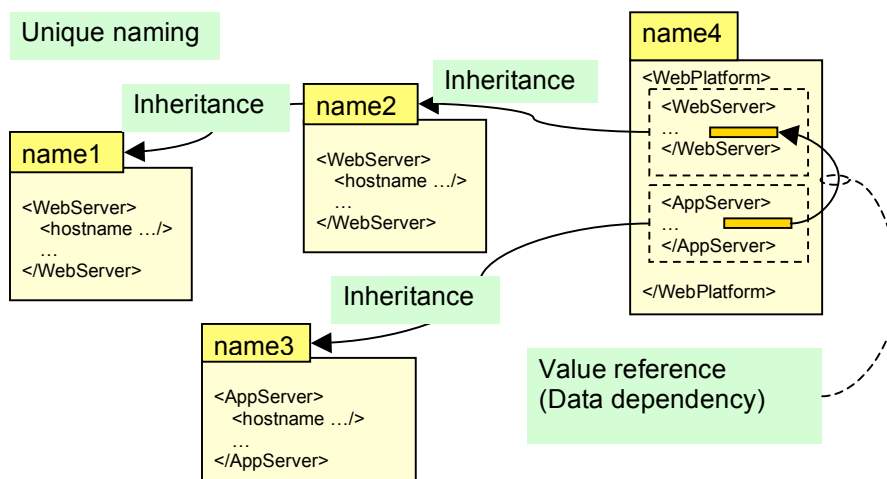


Figure 2: Functionalities of the Configuration Description Notations

This inheritance and reference structure should be resolved to make property lists available for configuration of the corresponding deployment components.

6 Document Structure

- A CDL document has the following structure:

```
<cdl:cdl targetNamespace=xsd:AnyURI?>
  <cdl:documentation .../>?
  <cdl:import .../>*
  <cdl:types>?
    <cdl:documentation .../>?
    <!-- schema definition -->*
    <!-- extensibility element -->*
  </cdl:types>
  <cdl:configuration>?
    <cdl:documentation .../>?
    <!-- PropertyList -->*
  </cdl:configuration>
  <cdl:system>?
    <cdl:documentation .../>?
    <!-- PropertyList -->+
  </cdl:system>
  <!-- extensibility element -->*
</cdl:cdl>
```

The optional attribute `cdl:cdl/@targetNamespace` specifies the namespace of (1) types defined in the `cdl:types` element, and (2) names of properties and property lists defined in `cdl:configuration` element.

6.1 Types

The `cdl:types` element encloses data type definitions using some type system (such as XSD). The configuration description MAY refer to these definitions to declare the type of a property and a property list. The namespace of data types within this element is specified at `cdl:cdl/@targetNamespace`, which is also the namespace of properties and property lists in the `cdl:configuration` element. When the configuration description refers to data types of other namespaces, the corresponding schema definition SHOULD be specified with a `cdl:import` element. Use of schema definition for validation in CDL is optional.

6.2 Configuration

The `cdl:configuration` element describes uniquely named property lists.

6.3 System

The `cdl:system` element describes a system configuration. A system configuration description has this element whereas a component template description does not.

6.4 Import

The `cdl:import` elements are used to refer to external configuration descriptions.

6.5 Documentation

The `cdl:documentation` elements are containers for human readable documentation.

7 Configuration Description

7.1 Top Level Property List

A `cdl:configuration` element contains zero or more property lists as its children. These property lists are called top level property lists and used to describe component templates. A top level property list **MUST** have a name unique within the document such that a top level element is uniquely referred to with a QName.

The following is an example of a configuration description in CDL document. The children of the `cdl:configuration` element, `WebServer` and `AppServer` are unique names of top level property lists.

```
<cdl:cdl targetNamespace="urn:tmp-uri1">
<cdl:configuration>
<WebServer>
  <hostname>www.example.com</hostname>
  <port>80</port>
</WebServer>
<AppServer>
  <WebServer .../>
  <hostname .../>
</AppServer>
</cdl:configuration>
</cdl:cdl>
```

Note that a property list which is not top level may not have a unique name.

7.2 Template Inheritance

In order to support inheritance of component templates, the CDL has a mechanism of prototype inheritance that allows the user to describe a new property list by extending an existing property list. When the user wants to specify values in an existing template, the user will describe a new property list through inheritance instead of modifying the original template.

7.2.1 Prototype Reference

The `@cdl:extends` attribute represents a prototype reference that refers to a property list to inherit.

```
<xsd:attribute name="extends" type="xsd:QName" />
```

The value of the `@cdl:extends` attribute is the QName that uniquely identifies a top level property list. The `@cdl:extends` attribute **MAY** appear at any node that is supposed to have a property list as its value. The following example shows that `@cdl:extends` can be attached not only to a top level element but also to its descendants at the same time.

```
<cdl:configuration xmlns:ext="urn:tmp-uri2">
<a cdl:extends="ext:aTemplate">
  <b cdl:extends="ext:bTemplate">
    <c>100</c>
    <d cdl:extends="ext:dTemplate"/>
  </b>
```

```
<e>200</e>
</a>
</cdl:configuration>
```

Note that, in this way, the user can describe a system structure that consists of multiple components, each of which is given as a component template by the provider.

7.2.2 Resolution

Resolution of a prototype reference consists of inheritance of elements and attributes. Let a node n have `@cdl:extends` that refers to a node n' . Resolution of the `@cdl:extends` attribute is done as follows:

- 1 If n' has `@cdl:extends`, resolve this prototype reference.
- 2 Inherit elements from n' to n .
- 3 Inherit attributes from n' to n .
- 4 Remove the `@cdl:extends` attribute from n .

In the following definition, two names are same if and only if they are the same name of the same namespace.

7.2.2.1 Inheritance of Elements

Inheritance of elements from a node n' to a node n is defined as follows:

- 1 Let N an empty node list.
- 2 For each child element e' of n' from the first element to the last element:
 - 2.1 If the node n has child elements $E = \{e_1, e_2, \dots\}$ that have the same name as e' :
 - 2.1.1 If N does not contain an element that has the same name as e' :
 - 2.1.1.1 For each element e_i in E in the document order, append e_i to the end of N . Inherit attributes from e' to e_i .
 - 2.2 Otherwise, append e' to the end of N .
- 3 For each child element e of the node n from the first element to the last element:
 - 3.1 if the element e' does not have a child element that has the same name as e , append e to the end of N .
- 4 Replace n' 's children with the nodes in N .

Note that the above procedure preserves the order of elements in n' in the resolved list N , followed by elements not appeared in n' (if any).

```
<list1>
  <a>1</a>
  <b>2</b>
</list1>
<list2 cdl:extends="list1">
  <c>3</c>
  <a>4</a>
</list2>
```

A property list `list2` in the above example is resolved as follows:

```
<list2>
  <a>4</a>
  <b>2</b>
  <c>3</c>
</list2>
```

Note also that when n' has duplicate properties (i.e., properties with the same name), they are totally replaced with properties with the same name in n (if any).

```
<list1>
  <a>1</a>
  <a>2</a>
  <a>3</a>
</list1>
<list2 cdl:extends="list1">
  <a>4</a>
  <a>5</a>
</list2>
```

A property list `list2` in the above example is resolved as follows:

```
<list2>
  <a>4</a>
  <a>5</a>
</list2>
```

7.2.2.2 Inheritance of Attributes

Inheritance of attributes from a node n' to a node n is defined as follows:

- 1 For each attribute a' of the node n' , if the node n does not have an attribute a that has the same name as a' , insert a' into the node n . If there is an attribute with the same name, discard a' .

Here, inheritance of attributes `@cdl:ref` and `@cdl:refroot` is handled as a special case. Refer to Section 7.3.4 for the detailed procedure.

7.2.3 Resolvable Prototype Reference

A prototype reference placed at node n is resolvable if and only if:

- There is one and only one top level property list (i.e., n' in the above definition) identified with the value of `@cdl:extends`. Otherwise, it is an error.
- The node that represents the name of the property list n' does not have a `@cdl:extends` attribute. Otherwise, the prototype resolution MUST be deferred until the prototype resolution of n' .

The following shows an example of a prototype reference that is not resolvable.

```
<a .../>
<b cdl:extends="a" .../>
<c cdl:extends="b" .../>
```

The prototype reference at the property list `c` is not resolvable until the prototype reference at the property list `b` is resolved.

There is no other restriction on resolution order except the above. For example, in the following description, there is only one restriction: `a2` must be resolved before `a3`.

```
<a2 cdl:extends="a1" ...>
  <b2 cdl:extends="b1" .../>
  <c2 cdl:extends="c1" .../>
</a2>
<a3 cdl:extends="a2" ...>
  <b2 cdl:extends="b4" .../>
</a3>
```

7.2.4 Example

The following descriptions include examples of inheritance:

```
<cdl:cdl targetNamespace="urn:tmp-uri1">
<cdl:configuration>
<WebServer>
  <hostname />
  <port>80</port>
</WebServer>

<Tomcat cdl:extends="WebServer">
  <port>8080</port>
  <maxThreads>200</maxThreads>
</Tomcat>
</cdl:configuration>
</cdl:cdl>
```

```
<cdl:configuration xmlns:tmpl="urn:tmp-uri1">
<Tomcat cdl:extends="tmpl:Tomcat">
  <hostname>myweb.com</hostname>
</Tomcat>
</cdl:configuration>
```

These descriptions are resolved as follows:

```
<WebServer>
  <hostname />
  <port>80</port>
</WebServer>

<Tomcat>
  <hostname />
  <port>8080</port>
  <maxThreads>200</maxThreads>
</Tomcat>
```

```
<Tomcat>
  <hostname>myweb.com</hostname>
  <port>8080</port>
  <maxThreads>200</maxThreads>
</Tomcat>
```

Note that the inheritance rule is only applied to immediate child elements (i.e., properties of the list) and is not applied to descendants of the children (i.e., property values of the list). When a child element is overridden, its value is fully replaced.

```
<AppPlatform>
  <WebServer>
    <hostname>localhost</hostname>
    <port>80</port>
  </WebServer>
```

```

<ApplicationServer>
  <hostname>localhost</hostname>
  <port>8080</port>
</ApplicationServer>
<DatabaseServer>
  <hostname>localhost</hostname>
  <port>6000</port>
</DatabaseServer>
</AppPlatform>

<MyApp cdl:extends="AppPlatform">
  <WebServer>
    <hostname>www.example.com</hostname>
  </WebServer>
  <ApplicationServer/>
</MyApp>

```

The property list "MyApp" is resolved to:

```

<MyApp>
  <WebServer>
    <hostname>www.example.com</hostname>
  </WebServer>
  <ApplicationServer/>
  <DatabaseServer>
    <hostname>localhost</hostname>
    <port>6000</port>
  </DatabaseServer>
</MyApp>

```

Note that the value of `WebServer/port` in the property list `AppPlatform` is not inherited to the property list `MyApp`. In order to allow an inheriting property list to override non-top-level properties in a hierarchy, the description SHOULD use parameterization, which is a pattern of description with combination of prototype references and value references. See Section 7.6 for parameterization.

Another way to achieve hierarchical inheritance is placing prototype references hierarchically. In the following example, because of the prototype reference `MyApp/WebServer/@cdl:extends`, the value of `WebServer/port` is inherited to the property list `MyApp`.

```

<WebServer>
  <hostname>localhost</hostname>
  <port>80</port>
</WebServer>

<AppPlatform>
  <WebServer cdl:extends="WebServer"/>
  <ApplicationServer>
    <hostname>localhost</hostname>
    <port>8080</port>
  </ApplicationServer>
  <DatabaseServer>
    <hostname>localhost</hostname>

```



```

    <port>6000</port>
  </DatabaseServer>
</AppPlatform>

<MyApp cdl:extends="AppPlatform">
  <WebServer cdl:extends="WebServer">
    <hostname>www.example.com</hostname>
  </WebServer>
</MyApp>

```

The property list MyApp is resolved as follows:

```

<MyApp>
  <WebServer>
    <hostname>www.example.com</hostname>
    <port>80</port>
  </WebServer>
  <ApplicationServer>
    <hostname>localhost</hostname>
    <port>8080</port>
  </ApplicationServer>
  <DatabaseServer>
    <hostname>localhost</hostname>
    <port>6000</port>
  </DatabaseServer>
</MyApp>

```

7.3 Value References

7.3.1 Reference Description

A reference to a particular property in a document is specified with two global attributes: @cdl:refroot and @cdl:ref.

```

<xsd:attribute name="refroot" type="xsd:QName"/>
<xsd:attribute name="ref" type="cdl:pathType"/>

```

They are placed in an element that represents a property without a value (i.e., a leaf node of an XML tree). Value references **MUST NOT** be placed in an element that has child elements whereas prototype reference **MAY** be placed in such an element to let child elements inherit from a prototype. Here is an example:

```

<hostname cdl:refroot="..." cdl:ref="..." />

```

The @cdl:refroot attribute is optional. The value of @cdl:refroot is the name of a top level property list (xsd:QName). The value of @cdl:ref is a subset of XPath expression (cdl:pathType). It **MUST** be a valid XPath expression, as defined in [XPath], and conform to the following extended BNF:

```

Path ::= ('/')? Step ('/' Step)*
Step ::= '.' | '..' | QName

```

The `@cdl:ref` attribute specifies a path to the destination properties of the value reference. The context information of XPath evaluation is given as follows:

- Let a node *n* have a `@cdl:ref` attribute. When the node *n* has `@cdl:refroot`:
 - The root node (`'/'`): the root of the property list identified with `@cdl:refroot`.
 - The context node (`'.'`): the root node.
- Otherwise:
 - The root node (`'/'`): the root of the property list that contains the node *n*.
 - The context node (`'.'`): the parent of the node *n*.

A reference whose path starts with `"/` is referred to as an absolute reference. The other type of reference is referred to as a relative reference. An absolute reference can be translated to an equivalent relative reference. Let an absolute reference *path* be placed at a node *n*. Let the depth (i.e., the number of steps to the top level element) of the node *n* be *d*.

When *n* does not have `@cdl:refroot`:

- If $d > 1$, the equivalent path is:

$$.. + \sum_{-2} + path$$

where summation represents concatenation of strings.

- If $d = 1$, the equivalent path is `."` + *path*.

When *n* has `@cdl:refroot`, the equivalent path is `."` + *path*.

The following is a configuration example for explanation of path expression. A reference is placed at a node "i".

```
<cdl:configuration>
<a>
  <c>
    <g>
      <i cdl:refroot="qname" cdl:ref="xpath"/>
      <j><k>1</k></j>
    </g>
    <h>2</h>
  </c>
  <d>3</d>
</a>
<b>
  <e>4</e>
  <f>5</f>
</b>
</cdl:configuration>
```

This XML can be visualized as a tree, as seen in Figure 3. A labeled box with an arrow pointing at a node shows path expressions to refer to the corresponding node from the node "i".

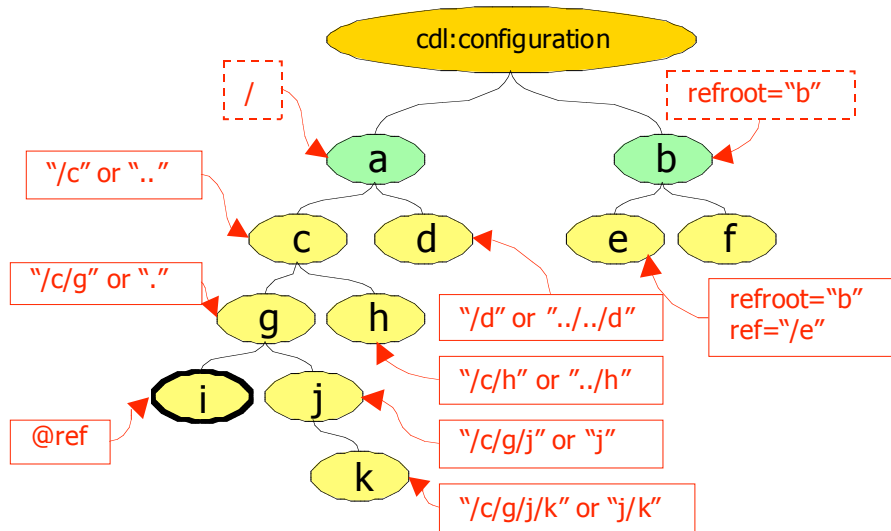


Figure 3: Example of path expression

7.3.2 Resolution

Resolution of a value reference is defined as the following transformation:

- 1 Let n be a node that has a value reference (`@cdl:ref`) to be resolved.
- 2 Let N be the node list that contains nodes identified with the reference.
- 3 Let N' be an empty node list.
- 4 For each node n_i in N in the document order:
 - 4.1 Create node n_i' with the same name as n .
 - 4.2 Copy all child nodes (and their descendants) of n_i (i.e., the value of the property n_i) to n_i' .
 - 4.3 Copy all attributes (except `@cdl:ref` and `@cdl:refroot`) of n to n_i' .
 - 4.4 Append n_i' to the end of N' .
- 5 Replace n with N' .

7.3.3 Resolvable Value Reference

A value reference at n referring to n' is resolvable if and only if:

- Prototype resolutions have been resolved: there is no `@cdl:extends` attribute at any descendants or ancestors of either n or n' .
- A `@cdl:lazy` attribute does not exist where the reference is placed. Otherwise, the resolution is deferred until the resolution of `@cdl:lazy`.
- The node n' and its descendants do not have any `@cdl:ref` and `@cdl:lazy` attribute. Otherwise, the resolution is deferred until these `@cdl:ref` and `@cdl:lazy` attributes are resolved.

See Section 7.5 for resolution of `@cdl:lazy` attributes.

7.3.4 Template Inheritance and Value References

Prototype references (`@cdl:extends`) MUST be resolved before resolution of value references.

When a prototype contains absolute references without `@cdl:refroot` attributes, they MUST be translated to equivalent relative references before prototype resolution.

In the following example, `a3` inherits `a`, which contains a reference `"/b"` at `d`.

```

<a>
  <b>100</b>
  <c>
    <d cdl:ref="/b"/>
  </c>
</a>
<a2>
  <b>200</b>
  <a3 cdl:extends="a">
    <b>300</b>
  </a3>
</a2>

```

The reference `"/b"` at `d` must be translated to the equivalent relative reference `"../b"` when it is inherited. The result of prototype resolution is as follows:

```

<a2>
  <b>200</b>
  <a3>
    <b>300</b>
    <c>
      <d cdl:ref="../b"/>
    </c>
  </a3>
</a2>

```

The value of the property `d` will be 300 after reference resolution.

Note that copying the reference to `a3` without translation yields erroneous resolution result as follows:

```

<a2>
  <b>200</b>
  <a3>
    <b>300</b>
    <c>
      <d cdl:ref="/b"/>
    </c>
  </a3>
</a2>

```

In this description, the value of the property `d` will be 200 after reference resolution.

A value reference can be inherited from a template. This fact arises two issues that must be treated specially: (1) Although a value reference is represented with a pair of attributes (i.e., `@cdl:refroot` and `@cdl:ref`), it must be handled as an atomic entity. (2) Since placing a reference (i.e., `@cdl:refroot` and `@cdl:ref`) and placing a value (i.e., child nodes) are two alternative way to specify a value of a property, a reference and a value should be handled interchangeably on inheritance. That is, one should be able to override the other. Accordingly, attributes `@cdl:ref` and `@cdl:refroot` are handled differently from other attributes (e.g., user-defined attributes) on inheritance of attributes from a node `n'` to a node `n` (see Section 7.2.2 for the base procedure):

1. A pair of `<@cdl:refroot, @cdl:ref>` is overridden by a pair of `<@cdl:refroot, @cdl:ref>` even if `@cdl:refroot` is null (missing). That is, even if the node `n` has only `@cdl:ref` and does not have `@cdl:refroot`, `@cdl:refroot` of `n'` is not copied to `n`.
2. A value reference (i.e., a pair of `<@cdl:refroot, @cdl:ref>`) is overridden by a property value (i.e., children of the node `n`). When the node `n` has children, `@cdl:refroot` and `@cdl:ref` of `n'` are not copied to `n`.

3. A property value (i.e., children of the node n) is overridden by a value reference.
When n has `@cdl:ref`, the children of n that have been copied from n are removed.

The following is an example of the case 1.

```
<cdl:configuration>
<a1>
  <c>1</c>
  <d>2</d>
</a1>
<a2>
  <b cdl:refroot="a1" cdl:ref="/c"/>
  <c>3</c>
  <d>4</d>
</a2>
<a3 cdl:extends="a2">
  <b cdl:ref="/d"/>
</a3>
</cdl:configuration>
```

On inheritance, the property list “a3” is resolved as

```
<a3>
  <b cdl:ref="/d"/>
  <c>3</c>
  <d>4</d>
</a3>
```

Note that `@cdl:refroot = "a1"` does not appear at “a3/b” – a pair of `@cdl:refroot` and `@cdl:ref` is completely replaced by a `@cdl:ref`.

The next example is in the case 2.

```
<cdl:configuration>
<a>
  <b cdl:ref="/c"/>
  <c>10</c>
</a>
<a1 cdl:extends="a">
  <b>20</b>
</a1>
</cdl:configuration>
```

On inheritance, the property list “a1” is resolved as

```
<a1>
  <b>20</b>
  <c>10</c>
</a1>
```

As for the case 3, the inheritance procedure described in Section 7.2.2 handles most cases in practice without any special treatment. Here is an example.

```
<cdl:configuration>
<a>
  <b>10</b>
  <c>20</c>
</a>
<a1>
  <b cdl:ref="/c"/>
</a1>
</cdl:configuration>
```

Without any modification, the inheritance of elements described in Section 7.2.2.1 resolves this as:

```
<a1>
  <b cdl:ref="/c"/>
    <c>20</c>
</a1>
```

The only exception is a case when a value reference is placed at a top level property list as follows.

```
<cdl:configuration>
<a cdl:refroot="b" cdl:ref="/a"/>
<b>
  <a>
    <c>10</c>
    <d>30</d>
  </a>
</b>
<a1 cdl:extends="a">
  <c>20</c>
</a1>
<a2 cdl:extends="a"/>
</cdl:configuration>
```

The top level lists “a1” and “a2” must be resolved as

```
<a1>
  <c>20</c>
</a1>
<a2 cdl:refroot="a0" cdl:ref="/b"/>
```

After value reference resolution, the top level lists “a,” “a1,” and “a2” are resolved as

```
<a>
  <c>10</c>
  <d>30</d>
</a>
<a1>
  <c>20</c>
</a1>
<a2>
  <c>10</c>
  <d>30</d>
</a2>
```

Note that, although this is a valid use case, this may not be intuitive to the user in practical situations. For instance, assume that “a” is a template provided a component provider and the user refers to it to describe language components “a1” and “a2.” To expect the resolved result above, the user needs to know that the top level list “a” is merely a reference. Otherwise the user may expect, by extending a template “a” with a value of property “c,” the property list “a1” will be resolved as

```
<a1>
  <c>20</c>
  <d>30</d>
</a1>
```

Accordingly, as a best practice, when a component provider publishes a template to users, a value reference should not be placed at a top level property list (i.e., the component provider should not use a template that is merely a reference).

7.3.5 Example

The following description includes examples of references.

```
<cdl:configuration>
<a>
  <b>test</b>
  <c>100</c>
  <d>200</d>
  <e>
    <f>abc</f>
    <g>def</g>
  </e>
</a>

<aa>
  <b cdl:refroot="a" cdl:ref="/b" />
  <c>300</c>
  <d cdl:ref="/c" />
  <e>
    <f cdl:refroot="a" cdl:ref="/e/g"/>
    <g cdl:ref="/e/f"/>
  </e>
</aa>
</cdl:configuration>
```

Here, the property list "aa" is resolved as follows:

```
<aa>
  <b>test</b>
  <c>300</c>
  <d>300</d>
  <e>
    <f>def</f>
    <g>def</g>
  </e>
</aa>
```

In the next example, a reference refers to multiple nodes:

```
<cdl:configuration>
<a>
  <portList>
    <port>80</port><port>8080</port>
  </portList>
</a>
<b>
  <portList>
    <port>8070</port>
    <port cdl:refroot="a" cdl:ref="/portList/port"/>
  </portList>
</b>
</cdl:configuration>
```

The property list b is resolved to:

```
<b>
```

```

<portList>
  <port>8070</port><port>80</port><port>8080</port>
</portList>
</b>

```

7.3.6 Expression

A `cdl:expression` element is a special type of value references and gives a property a boolean, number, or string value derived from other property values.

```

<cdl:expression value-of="xsd:string">
  <cdl:variable name="xsd:NCName" refroot="xsd:QName"?
    ref="cdl:pathType" cdl:lazy="xsd:boolean"? /*>
</cdl:expression>

```

The `@value-of` attribute is an XPath expression that is evaluated to yield a boolean, number or string value. The expression **MUST** be a valid XPath expression as specified in [XPath]. It **MUST NOT** contain any location path. It **MAY** contain a variable reference, which is defined with a `cdl:variable` element.

7.3.6.1 Resolution

As a special type of value references, `cdl:expression` is resolved as follows:

- 1 For each `cdl:variable` elements in `cdl:expression`,
 - 1.1 Identify a node *n* with `@refroot` and `@ref`.
 - 1.2 Bind the children of the node *n* (i.e., the value of the property *n*) to the name specified with `@name`.
- 2 Evaluate the XPath expression specified as the `@value-of` attribute with the set of variable bindings given above.
- 3 Replace the `cdl:expression` node with the evaluation result.

7.3.6.2 Example

The following is an example use case of a `cdl:expression` element. An XPath function is used in the `@value-of` attribute to concatenate two strings, one of which is given by a `cdl:variable` element that refers to the "hostname" property value.

```

<MyServer>
  <hostname>www.example.org</hostname>
  <url><cdl:expression value-of="concat('http://', $host, '/')">
    <cdl:variable name="host" ref="/hostname"/>
  </cdl:expression></url>
</MyServer>

```

It is resolved to:

```

<MyServer>
  <hostname>www.example.org</hostname>
  <url>http://www.example.org/</url>
</MyServer>

```


7.4 Schema Annotations

A component provider SHOULD describe well-defined data types of properties so that users of components can provide valid property values. Schema annotations defined as follows MAY be placed at properties to provide schema information to users. Use of specified annotations is optional: an implementation of CDL language processor can use this information for validation or other purposes (e.g., generation of XML Schema definition that validates the CDL document itself).

7.4.1 Property Type Declarations

A `@cdl:type` is an optional attribute that is placed at a property. The attribute specifies the data type (schema) of the property. The value of the attribute is a QName that identifies a data type.

```
<xsd:attribute name="type" type="xsd:QName"/>
```

A `@cdl:type` MAY refer to either (1) a type defined in the `cdl:types` in the current CDL document, (2) a type defined in an external namespace imported with a `cdl:import` element, or (3) a type defined in an external namespace specified with an `/cdl/@xmlns` attribute (such as the Component Model data types and primitive datatypes of XML Schema defined in [XML Schema Datatype]).

A component provider SHOULD provide data type information on properties with `@cdl:type` attributes so that users of components can provide valid values.

The following is an example of a property list with `@cdl:type` specified.

```
<WebServer>
  <hostname cdl:type="xsd:string"/>
  <port cdl:type="xsd:positiveInteger">80</port>
</WebServer>
```

7.4.2 Property Value Occurrence Constraints

A `@cdl:use` attribute MAY be placed at a property. The attribute specifies whether the property requires a value:

- required: The user of the component MUST assign values of this property.
- optional: The user of the component MAY assign values of this property.

The default value of `@cdl:use` is "optional". A component provider SHOULD place `@cdl:use` attributes with "required" at properties that require values.

```
<xsd:simpleType name="propertyUseType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="required"/>
    <xsd:enumeration value="optional"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:attribute name="use" type="cdl:propertyUseType"/>
```

The following is an example of a property list with `@cdl:use` specified.

```
<WebServer>
```

```
<hostname cdl:type="xsd:string" cdl:use="optional"/>
<port cdl:type="xsd:positiveInteger" cdl:use="required">80</port>
</WebServer>
```

At the end of all resolution phases, all properties marked as *required* within a system to be deployed **MUST** have values defined. It is an error if any such property remains un-assigned..

7.5 Laziness Annotations

7.5.1 Lazy Value Resolution

In some case, a required property value is not fixed before deployment of the system. A deployment service needs to resolve value references to such values at runtime. A `@cdl:lazy` attribute allows a deployment service to defer timing of value reference resolution.

```
<xsd:attribute name="lazy" type="xsd:boolean"/>
```

There are two use cases of the `@cdl:lazy` attribute:

- A lazy property: A property declaration with a `@cdl:lazy` attribute. Typically, a component provider specifies a lazy property in a component description.
- A lazy reference: A value reference with a `@cdl:lazy` attribute. Typically, a user of components specifies a lazy reference in a system description.

Value resolution of a reference to a lazy property or a lazy reference is deferred after the removal of the `@cdl:lazy` attribute. This removal of a `@cdl:lazy` attribute is referred to as *resolution* of the `@cdl:lazy` attribute.

A laziness resolution is defined as resolution of one or more `@cdl:lazy` attributes at the same time. After a laziness resolution, a value reference resolution **MUST** be applied to the document.

Resolution timing and selection of `@cdl:lazy` attribute to resolve is not defined in the CDL specification but defined in the component model specification, which defines constraints, or policies, on ordering of lifecycle management operations on components. A lazy property is resolved (with a property value assigned) when the component is deployed (semantics and timing of deployment are specified in the component model), and a lazy reference is resolved when the component is ready to deploy.

The CDL specification does not specify who will provide the value of a lazy property. It depends on semantics of a particular deployment component. A typical use case is that a deployment component provides a value at deployment time so that other components can refer to such a dynamic value (i.e., it is used to get values from a deployment component rather than giving values). Another use case in which a user may provide the value to a lazy property is that the deployment component gives a default value at deployment time when the user omitted it. In any case, a reference to such a lazy property should be resolved after its value is fixed at deployment time.

7.5.2 Lazy Properties

A `@cdl:lazy` attribute MAY be placed at any property that has no value. A reference to an property with a `@cdl:lazy` attribute MUST NOT be resolved before the `@cdl:lazy` attribute is resolved.

7.5.2.1 Resolution

Resolution of a lazy property is defined as the following transformation:

- 1 Let a node *n* has the `@cdl:lazy` attribute to be resolved
- 2 Insert a value into the node *n* if a value is defined
- 3 Remove the `@cdl:lazy` attribute at the node *n*

After the resolution, value reference resolution is performed, as described in section 7.3.2.

It is an error if the reference value could not be resolved at the time of lazy property resolution.

7.5.2.2 Example

In this example, a system consists of two components, `server1` and `server2`. The `server2` component requires the port number of the `server1` component as the value of a "destination" property. The "port" property value of `server1` is, however, given dynamically at deployment time. The provider of the `server1` component will place a `@cdl:lazy` attribute at the "port" property to declare that its value is assigned at run time. A value reference to this property will not be resolved before the resolution of this `@cdl:lazy` attribute.

```
<server1>
  <port cdl:lazy="true"/>
</server1>
<server2>
  <destination cdl:refroot="server1" cdl:ref="/port"/>
</server2>
```

When a "server1" component is deployed, the port number of this component is fixed. Within the CDL document, this event is seen as a laziness resolution that resolves the `@cdl:lazy` attribute at the port property as follows:

```
<server1>
  <port>8001</port>
</server1>
<server2>
  <destination cdl:refroot="server1" cdl:ref="/port"/>
</server2>
```

Value reference resolution is done immediately after the laziness resolution. The result of resolution is as follows:

```
<server1>
  <port>8001</port>
</server1>
<server2>
```

```
<destination>8001</destination>
</server2>
```

7.5.3 Lazy References

A `@cdl:lazy` attribute MAY be placed at any node that has a `@cdl:ref` attribute. The reference represented with the `@cdl:ref` attribute MUST NOT be resolved before the `@cdl:lazy` attribute is resolved.

7.5.3.1 Resolution

Resolution of a lazy reference is defined as the following transformation:

- 1 Let a node n has the `@cdl:lazy` attribute to be resolved
- 2 Remove the `@cdl:lazy` attribute at the node n

After the resolution, value reference resolution is performed, as described in section 7.3.2.

It is an error if the reference value could not be resolved at the time of lazy property resolution.

7.5.3.2 Example

System environment information is typically represented as a property list with a special name. In this example, this property list has a QName `"sys:systemProperties"`. Suppose a property `"deploymentTime"` requires a time stamp of deployment and a property `"time"` of the `"sys:systemProperties"` property list provides the current time. A `@cdl:lazy` attribute at the `"deploymentTime"` property let an implementation control the timing of value assignment.

```
<deploymentTime cdl:refroot="sys:systemProperties" cdl:ref="/time"
cdl:lazy="true"/>
```

When a component is deployed, the implementation resolves the `@cdl:lazy` attribute as follows:

```
<deploymentTime cdl:refroot="sys:systemProperties" cdl:ref="/time"/>
```

Value reference resolution is done immediately after the laziness resolution. The result of resolution is, for example, as follows:

```
<deploymentTime>2004-08-01T10:00:00Z</deploymentTime>
```

7.6 Parameterization

Parameterization is a pattern of configuration description, with which a provider of description can expose properties, which are located inside of the property list hierarchy, as top-level properties so that users can override these values with extension.

```
<server>
  <hostname>localhost</hostname>
  <port>4567</port>
</server>
```

```

<serverPair>
  <host1>localhost</host1>
  <host2>localhost</host2>
  <server1 cdl:extends="server">
    <hostname cdl:ref="/host1"/>
  </server1>
  <server2 cdl:extends="server">
    <hostname cdl:ref="/host2"/>
  </server2>
</serverPair>

<myPair cdl:extends="serverPair">
  <host1>one.example.com</host1>
  <host2>two.example.com</host2>
</myPair>

```

8 System Description

A deployable system is described with the `cdl:system` element, which contains one or more property list. The following is an example of a system description:

```

<cdl:system>
  <WebServer cdl:extends="webserver">
    ...
  </WebServer>
  <AppServer cdl:extends="appserver">
    ...
  </AppServer>
  <Database cdl:extends="database">
    ...
  </Database>
</cdl:system>

```

9 Import

The `cdl:import` is used to refer to external configuration description. The attribute `@location` specifies the absolute URI of an external CDL document to import. An external configuration description to be imported MUST NOT have a `cdl:system` element.

```

<xsd:element name="import">
  <xsd:complexType>
    <xsd:attribute name="location" type="xsd:anyURI"
      use="required"/>
  </xsd:complexType>
</xsd:element>

```

The algorithm for processing import declarations is as follows

1. All import declarations must be processed before resolution
2. All imported declarations are processed in the order of declaration within their document.

3. For each import declaration *i* with location *uri_i*:
4. If the document located by the *uri_i* has already been imported, do nothing
5. If the document has not been imported, then it should be imported by:
 1. Locating the external document
 2. Parsing it as another CDL document
 3. Resolving any import statements within that document
 4. Making the component templates within the configuration element available for resolution

The following is a use case example of the `cdl:import` element:

```
<cdl:import location="http://example.com/serverconfig.cdl"/>
...
<cdl:configuration xmlns:ex="http://example.com/serverconfig/">
  <MyServer cdl:extends="ex:genericwebserver" .../>
  ...
</cdl:configuration>
...
```

This example assumes that, in the CDL file located by `cdl:import/@location`, a template “genericwebserver” is defined in the namespace “http://example.com/serverconfig”. Unlike this example, it is valid to publish a template without specifying a namespace. In this case, the namespace of such a template will be the default namespace of the CDL document that imports the template. However, in order to avoid possible name conflicts, the component provider SHOULD specify the namespace of the name of a top level property list (i.e., the name of a template) in a CDL document to be imported by the users.

10 Documentation

The `cdl:documentation` element contains arbitrary text and elements for human or machine readable documentation.

```
<xsd:element name="documentation">
  <xsd:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any processContents="lax"/>
    </xs:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:element>
```

11 Security Considerations

The security requirements are achieved by combining Web Service/Grid/XML security standards with configuration description. For example, descriptions may be signed and encrypted. The deployment service must be allowed to decrypt configuration descriptions in order to process them. Future issues here include security setting when component providers and deployment service providers are different organizations.

12 Editor Information

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

13 Acknowledgements

The editor wishes to acknowledge the contributions from many people, including: Steve Loughran, Stuart Schaefer, Peter Toft, Dejan Milojicic, and Takashi Kojo.

Intellectual Property Statement

The OGF 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 OGF Secretariat. The OGF 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 OGF Executive Director.

Full Copyright Notice

Copyright (C) Open Grid Forum (2006-2007). 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 OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF 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 OGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE OPEN 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

- [CDDLML] *Configuration Description, Deployment, and Lifecycle Management (CDDLML) Foundation*,
http://forge.gridforum.org/projects/cddlml-wg/document/CDDLML_Foundation_Document/en/1
- [SF-CDL] *Configuration Description, Deployment, and Lifecycle Management (CDDLML) SmartFrog-based Language Specification*,
- [Component Model] CDDLML Component Model, CDDLML Working Group Draft, 2005.
- [CDDLML-API] CDDLML Deployment API, CDDLML Working Group Draft, 2005.
- [RFC2119] Brander, S. Key words for use in RFCs to Indicate Requirement Levels. IETF RFC 2119, March 1997.
- [XPath] *XML Path Language*, James Clark and Steve DeRose, eds., W3C, 16 November 1999. <http://www.w3.org/TR/1999/REC-xpath-19991116>
- [XML Schema Datatypes] *XML Schema Part 2: Datatypes*, Paul V. Biron and Ashok Malhotra, eds., W3C, 2 May 2001.
<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

Appendix A: XML Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.gridforum.org/namespaces/2005/02/cddlml/CDL-1.0"
  xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddlml/CDL-1.0"
  elementFormDefault="qualified">

<documentation>
Copyright (C) Open Grid Forum (2007). All Rights Reserved.
</documentation>

<simpleType name="propertyUseType">
  <restriction base="string">
    <enumeration value="required"/>
    <enumeration value="optional"/>
  </restriction>
</simpleType>

<simpleType name="pathType">
  <restriction base="string">
    <pattern
value="/(/)?((\i\c*:?)(\i\c*)|\.\|\.\.)(((\i\c*:?)(\i\c*)|\.\|\.\.))*">
    </pattern>
  </restriction>
</simpleType>

<attribute name="refroot" type="QName"/>
<attribute name="ref" type="cdl:pathType"/>
<attribute name="extends" type="QName"/>
```



```

<attribute name="type" type="QName"/>
<attribute name="use" type="cdl:propertyUseType"/>
<attribute name="lazy" type="boolean"/>

<element name="ref">
  <complexType>
    <attribute name="refroot" type="QName" use="optional"/>
    <attribute name="ref" type="cdl:pathType" use="required"/>
    <attribute name="lazy" use="optional"/>
  </complexType>
</element>

<complexType name="variableType">
  <attribute name="name" type="NCName" use="required"/>
  <attribute name="refroot" type="QName" use="optional"/>
  <attribute name="ref" type="cdl:pathType" use="required"/>
  <attribute name="lazy" use="optional"/>
</complexType>

<element name="expression">
  <complexType>
    <sequence>
      <element name="variable" type="cdl:variableType"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="value-of" type="string" use="required"/>
  </complexType>
</element>

<xsd:element name="documentation">
  <xsd:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any processContents="lax"/>
    </xs:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:element>

<complexType name="anyAttr" abstract="true">
  <sequence>
    <element ref="cdl:documentation" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<element name="import">
  <complexType>
    <complexContent>
      <extension base="cdl:anyAttr">
        <attribute name="location" type="anyURI"
use="required"/>
      </extension>
    </complexContent>
  </complexType>
</element>

```

```

<element name="types">
  <complexType>
    <complexContent>
      <extension base="cdl:anyAttr">
        <sequence>
          <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>

<complexType name="propertyListType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="cdl:documentation"
      minOccurs="0" maxOccurs="1"/>
    <any namespace="##other" processContents="lax"
      minOccurs="0" maxOccurs="unbounded">
      <annotation>
        <documentation>
          This element represents a property (cdl:propertyType).
        </documentation>
      </annotation>
    </any>
  </choice>
  <attribute ref="cdl:extends"/>
  <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<complexType name="propertyType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="cdl:documentation"
      minOccurs="0" maxOccurs="1"/>
    <any namespace="##any" processContents="lax"
      minOccurs="0" maxOccurs="unbounded">
      <annotation>
        <documentation>
          This element represents a property value.
        </documentation>
      </annotation>
    </any>
  </choice>
  <attribute ref="cdl:refroot"/>
  <attribute ref="cdl:ref"/>
  <attribute ref="cdl:extends"/>
  <attribute ref="cdl:type"/>
  <attribute ref="cdl:use"/>
  <attribute ref="cdl:lazy"/>
  <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<element name="configuration">
  <complexType>
    <complexContent>
      <extension base="cdl:anyAttr">
        <sequence>

```

```

    <any namespace="##other" processContents="lax"
        maxOccurs="unbounded">
        <annotation>
            <documentation>
                This element represents a top level property list
                (cdl:propertyListType).
            </documentation>
        </annotation>
    </any>
</sequence>
</extension>
</complexContent>
</complexType>
</element>

<element name="system">
    <complexType>
        <complexContent>
            <extension base="cdl:anyAttr">
                <sequence>
                    <any namespace="##other" processContents="lax"
                        maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                This element represents a top level property list
                                (cdl:propertyListType).
                            </documentation>
                        </annotation>
                    </any>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>

<element name="cdl">
    <complexType>
        <complexContent>
            <extension base="cdl:anyAttr">
                <sequence>
                    <element ref="cdl:import"
minOccurs="0" maxOccurs="unbounded"/>
                    <element ref="cdl:types"
minOccurs="0" maxOccurs="1"/>
                    <element ref="cdl:configuration"
minOccurs="0" maxOccurs="1"/>
                    <element ref="cdl:system"
minOccurs="0" maxOccurs="1"/>
                    <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="targetNamespace"
type="anyURI" use="optional"/>
            </extension>
        </complexContent>
    </complexType>

```

```
</element>
</schema>
```

Appendix B: Example

This section provides an example of component configuration description and system configuration description in CDL.

CDL itself does not assume any component model (i.e., required properties, lifecycle models, etc). This non-normative example here is only for explanation of CDL functionalities, and it does not meant to specify how the component model is represented in CDL. The normative component model is defined in the CDDL M Component Model specification [**Component Model**].

```
<cdl:cdl
  targetNamespace="http://cddl.org/component-model-example"
  xmlns="http://cddl.org/component-model-example"
  xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddl/CDL-1.0"
  xmlns:cmp="http://www.gridforum.org/cddl/components/2005/02"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <cdl:configuration>
  <Component>
    <cmp:CodeBase cdl:type="xsd:anyURI"/>
    <cmp:CommandPath cdl:type="cmp:commandPathType"/>
  </Component>

  <Compound cdl:extends="Component">
    <cmp:CommandPath>com.example.cddl.Compound</cmp:CommandPath>
    <cmp:Delegate/>
  </Compound>
  </cdl:configuration>
</cdl:cdl>
```

The element `Component` is the base template all components will inherit. The component has two properties: `CommandPath` and `CodeBase`. `CommandPath` is an identifier to specify what to instantiate in order to realize this deployment component. It can be seen as a “class” of the component. When a component provider publishes a configurable component, the description can inherit this `Component` and override `CommandPath` to specify the class of the configurable component. `CodeBase` is an identifier to specify a file (any content required for the application) to be given to this configurable component. For example, it refers to an application archive (such as a jar file for Java applications) installed on the component.

The element `Compound` defines a special utility component that manages other components as its children. Its implementation will be identified with the value of `CommandPath` property (i.e., “com.example.cddl.Compound”). The `Delegate` property means that this component will manage a set of child components on behalf of the system. Based on the above component templates, a component provider publishes a set of components that are used to run web applications.

```
<cdl:cdl
  targetNamespace="http://example.org/webapp-template"
  xmlns="http://example.org/webapp-template"
  xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddl/CDL-1.0"
```

```

xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://cddlm.org/component-model-example">
<cdl:configuration>
  <DBConnection>
    <JNDIName/>
    <hostname/>
    <port/>
    <username/>
    <password/>
  </DBConnection>

  <WebServer cdl:extends="c:Component">
    <cmp:CommandPath>com.example.cddlm.WebServer</cmp:CommandPath>
    <application cdl:type="xsd:anyURI"/>
    <applicationPath cdl:type="xsd:string"/>
    <hostname cdl:lazy="true"/>
    <port>8080</port>
    <dbconnection cdl:extends="DBConnection"/>
  </WebServer>

  <DBServer cdl:extends="c:Component">
    <cmp:CommandPath>com.example.cddlm.DBServer</cmp:CommandPath>
    <data cdl:type="xsd:anyURI"/>
    <hostname cdl:lazy="true"/>
    <port>3306</port>
    <username/>
    <password/>
  </DBServer>

  <WebApp cdl:extends="c:Compound">
    <cmp:sequence lifecycle="initialization">true</cmp:sequence>
    <cmp:sequence lifecycle="execution">true</cmp:sequence>
    <cmp:reverse lifecycle="termination">true</cmp:reverse>
    <application/>
    <applicationPath/>
    <dbname/>
    <data/>
    <dbuser/>
    <dbpassword/>

    <DB cdl:extends="DBServer">
      <cmp:CodeBase cdl:ref="/data"/>
      <username cdl:ref="/dbuser"/>
      <password cdl:ref="/dbpassword"/>
    </DB>

    <Web cdl:extends="WebServer">
      <cmp:CodeBase cdl:ref="/application"/>
      <applicationPath cdl:ref="/applicationPath"/>
      <dbconnection cdl:extends="DBConnection">
        <JNDIName cdl:ref="/dbname"/>
        <hostname cdl:ref="/DB/hostname"/>
        <port cdl:ref="/DB/port"/>
        <username cdl:ref="/DB/username"/>
        <password cdl:ref="/DB/password"/>
      </dbconnection>
    </Web>
  </WebApp>

```

```
</cdl:configuration>
</cdl:cdl>
```

Two components, `WebServer` and `DBServer`, extend `Component` and define additional properties such as `hostname` and `port`. The component user will define component description that extends these components and provide appropriate values by overriding properties.

`DBConnection` is not a component but merely a composite data structure since it does not inherit `Component` (hence, it does not have properties required to be a component). Such data structures are defined and extended for convenience.

`WebApp` is a compound component that consists of two components that extend `WebServer` and `DBServer`. References are specified so that an application on `WebServer` can connect to a database on `DBServer`. Parameterization pattern is effectively used in this definition so that the user of this template only needs to override parameters such as `application` and `applicationPath` by extension. Given those parameter values, properties of sub-components are appropriately assigned through reference resolution.

By referring to the above component description, a deployment service requester requests deployment of a web application as follows:

```
<cdl:cdl
targetNamespace="http://example.org/webapp-deploy"
xmlns="http://example.org/webapp-deploy"
xmlns:t="http://example.org/webapp-template"
xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0">
<cdl:system>
  <WebApplication cdl:extends="t:WebApp">
    <t:application>http://repository.org/test.war</t:application>
    <t:applicationPath>/test</t:applicationPath>
    <t:dbname>jdbc/Test</t:dbname>
    <t:data>http://repository.org/db.zip</t:data>
    <t:dbuser>myapp</t:dbuser>
    <t:dbpassword>pass</t:dbpassword>
  </WebApplication>
</cdl:system>
</cdl:cdl>
```

The above system description is statically resolved as follows:

```
<WebApplication>
  <cmp:CodeBase/>
  <cmp:CommandPath cdl:type="cmp:commandPathType">
    com.example.cddlm.Compound</cmp:CommandPath>
  <cmp:Delegate/>
  <cmp:sequence lifecycle="initialization">true</cmp:sequence>
  <cmp:sequence lifecycle="execution">true</cmp:sequence>
  <cmp:reverse lifecycle="termination">true</cmp:reverse>
  <t:application>http://repository.org/test.war</t:application>
  <t:applicationPath>/test</t:applicationPath>
  <t:dbname>jdbc/Test</t:dbname>
  <t:data>http://repository.org/db.zip</t:data>
  <t:dbuser>myapp</t:dbuser>
  <t:dbpassword>pass</t:dbpassword>
  <t:DB>
    <cmp:CodeBase cdl:type="xsd:anyURI">
      http://repository.org/db.zip</cmp:CodeBase>
```

```
<cmp:CommandPath cdl:type="cmp:commandPathType">
  com.example.cddl.DBServer</cmp:CommandPath>
<t:hostname cdl:lazy="true"/>
<t:port>3306</t:port>
<t:username>myapp</t:username>
<t:password>pass</t:password>
</t:DB>
<t:Web>
  <cmp:CodeBase cdl:type="xsd:anyURI">
    http://repository.org/test.war</cmp:CodeBase>
  <cmp:CommandPath cdl:type="cmp:commandPathType">
    com.example.cddl.WebServer</cmp:CommandPath>
  <t:applicationPath cdl:type="xsd:string"/>test</t:applicationPath>
  <t:hostname cdl:lazy="true"/>
  <t:port>8080</t:port>
  <t:dbconnection>
    <t:JNDIName>jdbc/Test</t:JNDIName>
    <t:hostname cdl:ref="../../t:DB/t:hostname"/>
    <t:port>3306</t:port>
    <t:username>myapp</t:username>
    <t:password>pass</t:password>
  </t:dbconnection>
</t:Web>
</WebApplication>
```

Note that the reference at `WebApplication/t:Web/t:dbconnection/t:hostname` has not been resolved since it refers to a lazy property, `WebApplication/t:DB/t:hostname`. A runtime system is supposed to resolve this reference in deployment time.