GWD-R (draft-ggf-cddlm-xml-cdl-03.doc)
Configuration Description, Deployment and Lifecycle Management
XML-Based Configuration Description Language
http://forge.gridforum.org/projects/cddlm-wg

Editors:
J. Tatemura, NEC

8/9/2004

Configuration Description, Deployment, and Lifecycle Management

XML Configuration Description Language Specification Revision 0.3 (in progress)

Status of this Memo

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

Abstract

Successful realization of the Grid vision of a broadly applicable and adopted framework for distributed system integration, virtualization, and management requires the support for configuring Grid services, their deployment, and 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 CDDLM working group within the Global Grid Forum (GGF), provides a definition of the XML-based configuration description language and its requirements.

GLOBAL GRID FORUM
office@ggf.org
www.ggf.org

Table of Contents

List of Figures

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 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.

The following namespaces are used in this document:

xsd	http://www.w3.org/2000/10/XMLSchema
cdl	This specification (uri to be defined)

2 CDDLM-WG and the Purpose of this Document

The CDDLM WG addresses how to: describe configuration of services; deploy them on the Grid; and manage their deployment lifecycle (instantiate, initiate, start, stop, restart, etc.). The intent of the WG is to gather researchers, developers, practitioners, and theoreticians in the areas of services and application configuration, deployment, and deployment life-cycle management and to explore the community need for a broader effort in this area. The target of the CDDLM WG is to come up with the specifications for CDDML a) language, b) component model, and c) basic services. This document represents one of the two CDDLM language specifications. This specification is based on XML, which provides interoperability with other XML-based Grid specifications, and other specification is based on SmartFrog language developed at HP Labs, which provides user-friendly syntax and functionalities. The two languages will be compatible.

3 Configuration Description in CDDLM

3.1 Configuration Description in Deployment Services Framework

This section describes overview of deployment services, through which users can manage the deployment lifecycle of their services on the Grid infrastructure. Configuration description, specified in this document, is used in this deployment services framework.

- Configurable Components

  A configurable component is a resource with which a service can be deployed. Examples include a Linux server, an application server, and a database server.

    - Configuration Properties: A set of properties is defined for each component. The user can configure the component by giving values of these properties.
    - Configurable Component Providers: A provider of configurable components publishes description on these components so that users can configure these components properly. This document provides specification on this description. This document, however, does not specify how the description is published and discovered by users.
    - Component Lifecycle: A component has status in terms of deployment lifecycle such as deployed and running. The configuration description language does not assume any specific lifecycle model.
- Systems (to be deployed)
  A system is a set of configurable components required to deploy a service.
    - System Lifecycle: A system has status in terms of deployment lifecycle, which is defined based on status of configurable components within the system. For example, a system is running when all the components in the system are running.
- Deployment Services
  A deployment service handles deployment and lifecycle management of a system from deployment to un-deployment.
    - Deployment Service Providers: A provider of deployment services may or may not be a provider of configurable components.
    - Lifecycle Management Operations: The deployment service provides operations to control system lifecycle.
    - Lifecycle Monitoring Services: The deployment service provides ways to monitor the lifecycle status of the system.

Figure 1 illustrates how configuration description is used in this deployment services framework. Figure 1 shows two use cases of configuration description in this framework:

- Description of configurable components, which is used in the publish and discovery process
- Description of systems, which is used in the lifecycle management operations

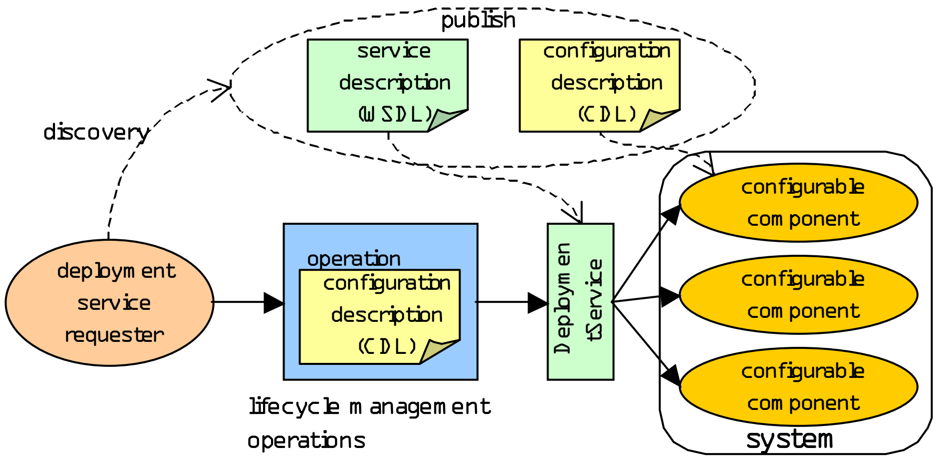
publish
service description (WSDL)
configuration description (CDL)
discovery
configurable component
configurable component
configurable component
deployment service requester
operation
configuration description (CDL)
DeploymentService
lifecycle management operations
system

Figure 1: The Deployment Service Framework

3.2 Use of Configuration

3.2.1 Configurable Components

With the configuration description language, a configurable component provider describes configurable components that are to be published. The description MAY include the following information:

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

The description MAY include extra information using other specifications. Examples of such information include:

- Restrictions and negotiability on property values
- Security information
- Other policy information
- Service information such as endpoint references.

Specification on such information is out of scope of this document.

3.2.2 System Configuration

A system configuration is described and used in order to deploy a system that consists of configurable components.
The configuration MAY include

- References to configurable components
- Values of properties
- Value dependencies among properties

Systems described in the language are categorized into the following two types:

- Deployable System Configuration
  A deployable system configuration is a complete system configuration that is ready to be used in deployment.
- Partial System Configuration
  A system configuration MAY be incomplete, which means that the description includes properties without values required to deploy the system. Such configurations are used as modules with which a deployable (complete) system description is composed.

3.3 Roles of Two Configuration Description Languages

XML-based Configuration Description Language provides composability with other specifications and interoperability between different implementations:

- Composability with Web Services and Grid components (such as security)
- Interoperability among different service implementations
- Interoperability among different front-end languages and tools

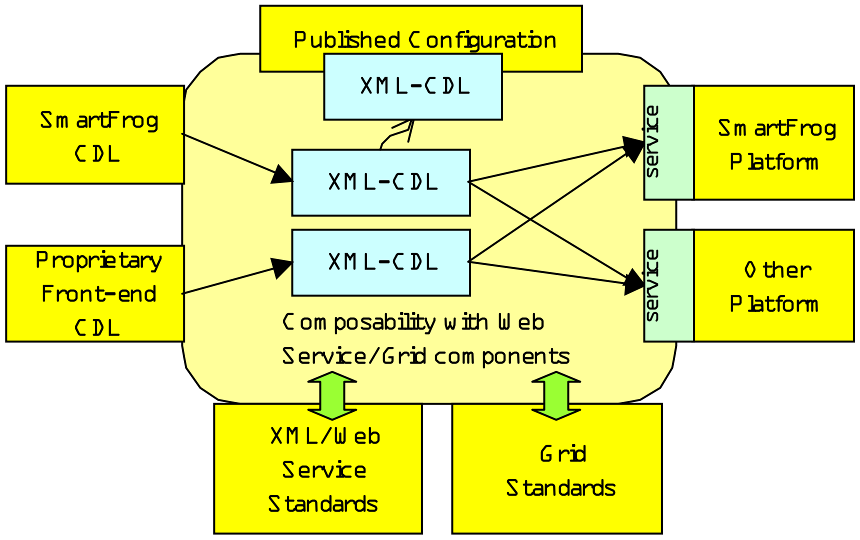

Figure 2: Role of the XML-based Configuration Description Language

SmartFrog-based Configuration Description Language provides user-friendly syntax of description, used as a front-end language. When SmartFrog is used in the deployment service framework, it is preprocessed and translated into XML-CDL.

4 Requirements for the Language

- Declarative Description. The configuration description should be declarative: it should not be a sequence of operations but a set of declarations that describes dependency between resources. The declarative description should provide enough information for 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 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.
- Consistency. 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 configuration vendors. The language should provide a way to define a new composed 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 Property Lists

Data required to configure a component is given as a list of properties, each of which has a name and a value. The name of a property is unique in a property list, that is, a property with a particular name MUST NOT appear more than once in a property list. A property MAY have a property list as its value so that a nested structure of properties can be constructed.
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. An attribute of the element MUST NOT be used to define a property value. A property list is an XML element whose children are properties. A property and a property list MUST allow insertion of any type of attributes. A property list MUST allow insertion of any type of elements as its children.
The following is an example of a property list:

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

5.2 Configuration Description

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

- Unique naming of property lists

- Inheritance of property lists
- References that define data dependency between properties

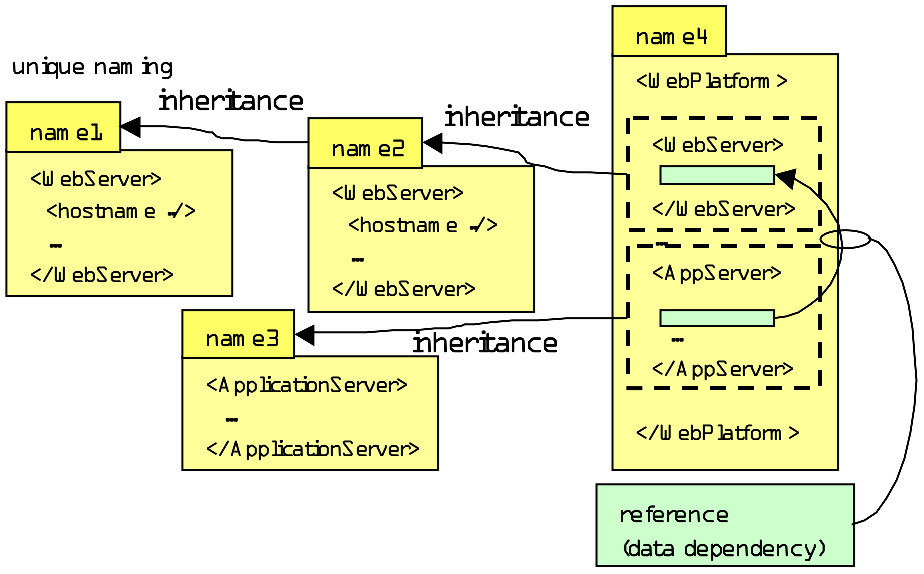

Figure 3: Functionalities of the Configuration Description Notations

The language processor resolves this inheritance and reference structure and makes property lists available for configuration of the corresponding components.

6 Document Structure

A configuration description document has the following structure:

```
<cdl:cdl targetNameSpace=xsd:AnyURI?>
  <cdl:documentation …/>?
  <cdl:import …/>*
  <cdl:types>?
    <cdl:documentation …/>?
    <xsd:schema …/>*
    <-- extensibility element -->*
  </cdl:types>
  <cdl:configuration>?
    (<cdl:documentation …/>?
     <-- PropertyList -->)*
  </cdl:configuration>
  <cdl:system>?
    (<cdl:documentation …/>?
     <-- PropertyList -->)*
  </cdl:system>
  <-- extensibility element -->*
</cdl:cdl>
```

6.1 Types

A data type (schema) MAY be defined in the cdl:types element. The configuration description in this document MAY declare the type of a property by referring to the definition in the cdl:types element.

6.2 Configuration

The cdl:configuration element describes uniquely named property lists.

6.3 System

The cdl:system element describes a system configuration.

6.4 Import

The cdl:import elements are used to refer to external configuration descriptions or schema definitions.

6.5 Documentation

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

7 Configuration Description

7.1 Property List Name

A property list is called a top level property list when it is a child of a cdl:configuration element. A top level property list MUST has a name unique within the document. Combined with the namespace name specified with cdl:cdl/@targetNamespace, the name is uniquely referred to with QName.

The following is an example of a 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 Prototype References

7.2.1 Reference Model

The @cdl:extends attribute is used in a property list to inherit an existing property list.

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

The value of the @cdl:extends attribute is the QName of a property list that is to be inherited. Only a top level property list, which has a unique name, MAY be the destination of a prototype reference. 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 the root node but also to its descendants at the same time.

```
<cdl:configuration xmlns:ext="…">
<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>
```

7.2.2 Resolution

Resolution of a prototype reference consists of the following three data transformations:

- Removal of @cdl:extends attribute
- Inheritance of elements
- Inheritance of attributes

7.2.2.1 Inheritance of Elements

Inheritance of elements is defined as follows:

1 Let a node *n* has @cdl:extends which refers to the node n_0, and let *N* an empty node list.
2 For each child element *e* of the node n_0 from the first element to the last element:
   2.1 If the node *n* has a child element *e*, append this element to the last of the node list *N*.
   2.2 Otherwise, append the child element *e* of the node n_0 to the last of the node list *N*.
3 For each child element *e* of the node *n* from the first element to the last element: if the node list *N* does not contain an element *e*, append *e* to the last of the node list.
4 Replace the children of the node *n* with the nodes in the node list *N*.

7.2.2.2 Inheritance of Attributes

Inheritance of attributes is defined as follows:

1 Let a node *n* has @cdl:extends which refers to the node n_0.
2 For each attribute @a of the node n_0, if the node *n* does not have an attribute @a, insert @a into the node *n*.

7.2.3 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 the following example, on the other hand, 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 Model

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 a tree).

```
<hostname cdl:refroot=”…” cdl:ref=”…”/>
```

They MUST NOT be placed in an element that has child elements.

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 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.

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 4. A labeled box with an arrow pointing at a node shows path expressions to refer to the corresponding node from the node "i".

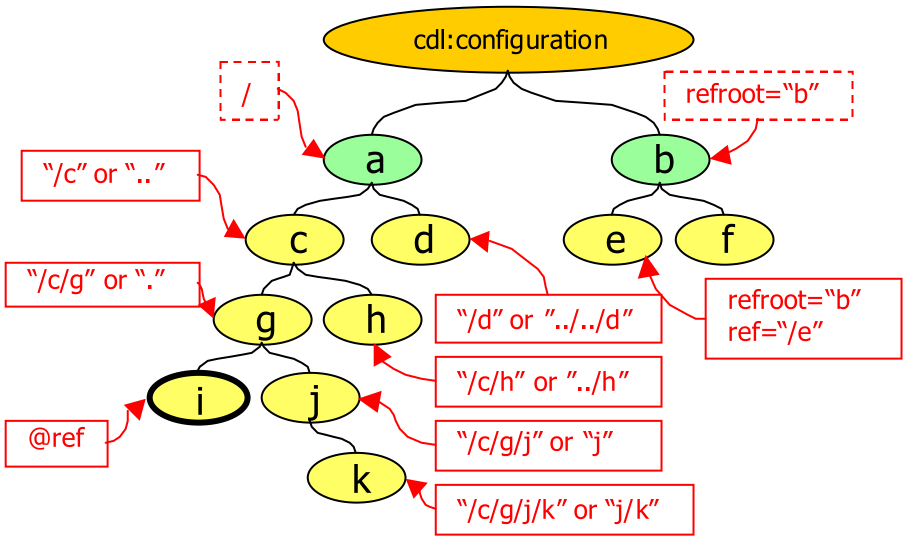

Figure 4: Example of path expression

7.3.2 Resolution

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

1 Let a node n_1 the node identified with @cdl:ref, which is attached to the node *n*.
2 Let a node list *N* be the children of the node n_1 (i.e., the value of the property n_1).
3 Insert nodes in the list *N* into the node *n* with their order preserved.
4 Remove @cdl:ref and @cdl:refroot (if it exists) at the node *n*.

7.3.3 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>
```

7.3.4 Value Insertion

A cdl:ref element is a special type of value references and inserts a value into a structured data.

```
<cdl:ref refroot=”xsd:QName”? ref=”cdl:pathType”
       cdl:lazy=”xsd:boolean”?/>
```

This element is typically used when a property has a value of structured data type. By placing a cdl:ref element within a structured data, external data can be imported as a part of the structured data.

7.3.4.1 Resolution

As a special type of value reference, cdl:ref is resolved as follows:

1 Let a node n_1 the node identified with cdl:ref/@ref.
2 Let a node list N be the children of the node n_1 (i.e., the value of the property n_1).
3 Replace the node cdl:ref with nodes in the list N with their order preserved.

Note that the following @cdl:ref attribute and cdl:ref element are equivalent and resolved to the same result:

```
<a cdl:ref=”/b”>
```

```
<a><cdl:ref ref=”/b”/></a>
```

7.3.4.2 Example

The following is an example use case of a cdl:ref element. A property list “a” has a property “portList” whose value is a list of “port” elements. The cdl:ref element is used in a property list “b” in order to import this list and add a new “port” element.

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

The property list b is resolved to:

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

7.3.5 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.

7.3.5.1 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.

7.3.5.2 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 configurable component provider SHOULD describe well-defined schema of components so that users of components can provide valid configuration values. Schema annotations defined as follows MAY be placed at properties to provide schema information to users. An implementation of CDL language processor MAY use this information to validate configuration data after resolution.

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

The default value of @cdl:use is “optional”. A configurable component provider SHOULD place @cdl:use attributes 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:use=”optional”/>
 <port cdl:use=”required”>80</port>
</WebServer>
```

7.4.2 Property Type Declarations

A @cdl:type attribute MAY be 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 a type defined in the cdl:types in the current CDL document, a type defined in an external namespace imported with a cdl:import element, or primitive datatypes of XML Schema defined in [XML Schema Datatype].
A configurable component provider SHOULD provide data type information on properties with @cdl:type attributes so that users of configurable 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>
```

7.5 Laziness Annotations

7.5.1 Lazy Value Resolution

In some case, a value required to property 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 let a deployment service control 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.
Specification on resolution timing and selection of @cdl:lazy attribute to resolve is out of this specification's scope.

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 done.

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 done.

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 propery "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"/>
  </serer2>
</serverPair>
```

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

8 System Description

A system can be described with the cdl:system element, which contains a property list. After resolution, a deployment service implementation observes the content of the cdl:system element to process deployment of multiple components.

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 or schema information specified with external namespaces.

```
<xsd:element name=”import”>
 <xsd:complexType>
  <xsd:attribute name=”namespace” type=”xsd:anyURI”
             use=”optional”/>
  <xsd:attribute name=”location” type=”xsd:anyURI”
             use=”required”/>
 </xsd:complexType>
</xsd:element>
```

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

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

10 Documentation

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

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

11 Resolution: Operational Aspects

11.1 Language Processing Model

Figure 5 illustrates a model of CDL language processing. Note that it does not specify anything on implementation.

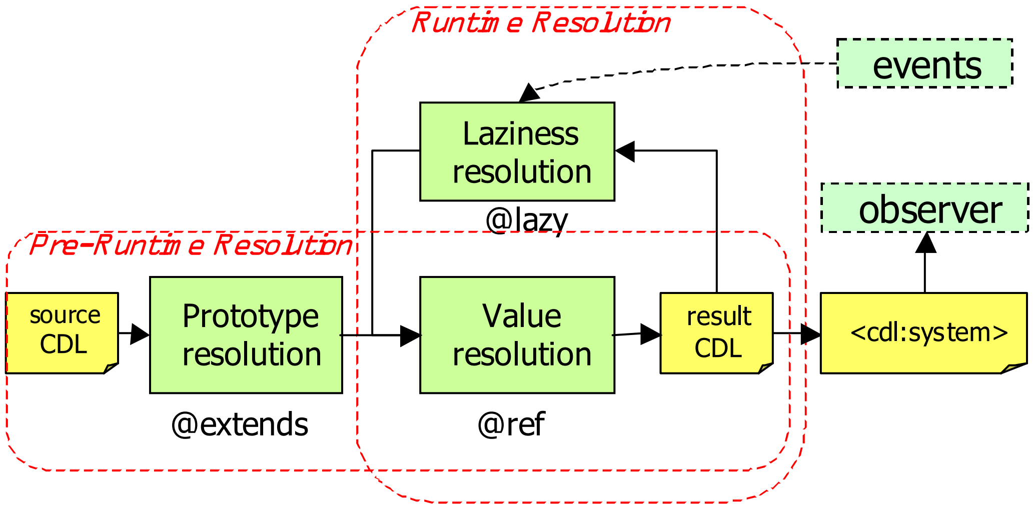

Figure 5: CDL Language Processing Model

11.2 Pre-Runtime Resolution

Resolution is done in the following order:

1 Prototype reference resolution
  Repeat resolution of a resolvable prototype reference until there is no resolvable prototype reference
2 Value reference resolution
  Repeat resolution of a resolvable value reference until there is no resolvable value reference

When the configuration document includes laziness annotations (@cdl:lazy), some value references may not be resolved until deployment time (runtime).

11.2.1 Resolvable Prototype Reference

A prototype reference is resolvable if and only if:

- There is one and only one top level property list (let it be *n*) identified with the value of @cdl:extends
- The node that represents the name of the property list *n* does not have a @cdl:extends attribute.

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.

11.2.2 Resolvable Value Reference

A value reference is resolvable if and only if:

- A @cdl:lazy attribute does not exist where the reference is placed.
- Evaluation of the location path returns one and only one node (let it be *n*).
- The node *n* and its descendants do not have any @cdl:ref and @cdl:lazy attribute.

11.3 Runtime Resolution

A runtime resolution is invoked by an event during deployment time.

1 Laziness resolution
Resolution of one or more laziness annotations
2 Value reference resolution
Repeat resolution of a resolvable value reference until there is no resolvable value reference

11.4 Resolved Configuration Data

An implementation that uses results of CDL processing MAY observe the content of the cdl:system element after each resolution. The semantics of data structure in the cdl:system element depends on the implementation.

12 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 configurable component providers and deployment service providers are different organizations.

13 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

14 Acknowledgements

The editors wish to acknowledge the contributions from many people, including: Steve Loughram, Stuart Schaefer, Peter Toft, Dejan Milojicic, and Takashi Kojo.

References

[CDDLM] Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation, http://forge.gridforum.org/projects/cddlm-wg/document/CDDLM_Foundation_Document/en/1
[SF-CDL] Configuration Description, Deployment, and Lifecycle Management (CDDLM) SmartFrog-based Language Specification,
[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/2000/10/XMLSchema”
   xmlns:cdl=”http://ggf.org/cddlm/xml-cdl/1.0”
   targetNameSpace=”http://ggf.org/cddlm/xml-cdl/1.0”
   elementFormDefault=”qualified”>

<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 ref=”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 ref=”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>


<element name=”documentation”>
 <complexType mixed=”true”>
  <choice minOccurs=”0” maxOccurs=”unbounded”>
    <any minOccurs=”0” maxOccurs=”unbounded”/>
  </choice>
  <anyAttribute/>
 </complexType>
</element>

<complexType name=”anyAttr” abstruct=”true”>
 <sequence>
   <element ref=”cdl:documentation” minOccurs=”0” maxOccurs=”1”/>
 </sequence>
 <anyAttribute namespace=”##other”/>
</complexType>

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

```
<element name=”types”>
 <complexType>
  <complexContent>
   <extension base=”cdl:anyAttr”>
    <sequence>
     <any namespace=”##other”
          minOccurs=”0” maxOccurs=”unbounded”/>
    </sequence>
   </extention>
  </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”/>
 </choice>
 <anyAttribute namespace=”##other”/>
 </complexContent>
</complexType>


<element name=”configuration” type=”cdl:propertyListType”/>

<element name=”system” type=”cdl:propertyListType”/>

<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”
            minOccurs=”0” maxOccurs=”unbounded”/>
    </sequence>
    <attribute name=”targetNamespace”
            type=”anyURI” use=”optional”/>
   </extension>
  </complexContent>
 </complexType>
</element>
</schema>
```

Appendix B: Example