Data Center Markup Language
Framework Specification

Draft
Version 0.1112
May 5August 28, 2004
Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>version 0.1</td>
<td>November 9, 2003</td>
<td>Initial draft</td>
</tr>
<tr>
<td>version 0.2</td>
<td>November 14, 2003</td>
<td>Wording and example edits</td>
</tr>
<tr>
<td>version 0.3</td>
<td>November 17, 2003</td>
<td>Added copyright</td>
</tr>
<tr>
<td>version 0.4</td>
<td>March 31, 2004</td>
<td>Incorporated feedback from 3/2004 working group meeting.</td>
</tr>
<tr>
<td>version 0.7</td>
<td>April 14, 2004</td>
<td>Incorporated feedback from reviewers.</td>
</tr>
<tr>
<td>version 0.8</td>
<td>April 22, 2004</td>
<td>More feedback</td>
</tr>
<tr>
<td>version 0.9</td>
<td>April 23, 2004</td>
<td>Move processor implementation to appendix</td>
</tr>
<tr>
<td>version 0.10</td>
<td>April 28, 2004</td>
<td>Added language around blueprint to make it more experimental.</td>
</tr>
<tr>
<td>version 0.11</td>
<td>May 5, 2004</td>
<td>Incorporated OWL and RDF related feedback from Michael K. Smith</td>
</tr>
<tr>
<td>version 0.12</td>
<td>August 28, 2004</td>
<td>Extended meta-model to describe common DCML constructs. Added dependsOn property to the core schema. Added server provisioning use-case.</td>
</tr>
</tbody>
</table>

Copyright

Copyright 2003, 2004 dcml Inc.. All rights reserved.
About This Document

Data Center Markup Language (DCML) is a data oriented approach to solving the problem of large scale systems management, particularly in a data center environment. DCML stitches together multiple management systems and tools to form a unified management view of the environment. In this unified view, management systems can exchange domain knowledge about the environment with other management systems in the same environment. A common data oriented approach is the first step toward a unified management view of the environment. With a data oriented approach, systems communicate by importing and exporting data in vocabularies of a well known language - DCML.

This approach requires a definition of key concepts and elements in forms of DCML vocabularies, semantics for DCML vocabulary definitions, encoding of DCML vocabularies, and processing of DCML instance documents.

This document defines the DCML data oriented framework for use by all DCML sub-specifications and DCML compliant management systems and tools.

Optional Reading

The following background knowledge is helpful, but not essential, in making effective use of this specification:


Table of Contents

1  Introduction ........................................................................................................................................4  
2  Framework Overview ......................................................................................................................5  
   2.1  ..............................................................................................................................................5  
   2.2  ...........................................................................................................................................6  
   2.3  ...........................................................................................................................................6  
   2.4  ...........................................................................................................................................6  
3  Design Goals .....................................................................................................................................7  
   3.1  ................................................................................................................................................7  
   3.2  ..............................................................................................................................................7  
   3.3  ............................................................................................................................................7  
   3.4  ............................................................................................................................................8  
   3.5  ............................................................................................................................................8  
   3.6  ............................................................................................................................................8  
   3.7  ............................................................................................................................................8  
   3.8  ............................................................................................................................................8  
4  Design Approach ..............................................................................................................................9  
   4.1  ................................................................................................................................................9  
   4.2  ..............................................................................................................................................10  
   4.3  ..............................................................................................................................................10  
5  Conceptual Model ............................................................................................................................10  
   5.1  ................................................................................................................................................11  
   5.1.1 ........................................................................................................................................11
1 Introduction

Managing a large-scale data center environment is a daunting task that has gotten markedly more complex recently. The proliferation of the Internet and the World Wide Web has made it dramatically easier to develop and deploy applications. The introduction of multi-tiered Web-based application architectures has caused a substantial shift of computing power from client back to
server. The continuing trend toward smaller, cheaper servers has resulted in a dramatic increase in the number of servers. The result of these trends has been an explosion of complexity and scale in the data center. Today’s data centers often contain thousands or tens of thousands of servers, networking devices, storage devices and other special-purpose equipment, running an even greater array of operating systems, software, configurations and data.

Traditional management systems and standards have not kept pace with these changes. Today’s data center management relies on making manual changes to the environment and reacting to problems after they occur. Traditional management systems and standards are aimed at making this manual, reactive approach to data center management easier, but there has not been a fundamental advance in management for a decade.

This situation has led to a new approach to data center management employing automation to replace the traditional manual approach. This new approach does not replace the conventional monitoring, ticketing, and other operational systems. Rather, it typically integrates with and builds on those systems, making them more effective in the new data center environment.

This new approach has brought with it new requirements for vendor-independence and interoperability. Just as traditional management approaches led to the development of standards such as SNMP and CIM for interoperability between management systems and the technologies they manage (e.g., network monitoring tools and the network elements monitored), today’s new automated management approach requires new standards to support interoperability among management solutions. The data center markup language (DCML) is a proposal for one such standard.

DCML provides the ability to describe three broad types of information required in an automated data center environment. The first is the state of the environment itself, similar to and compatible with what would be described by a traditional management system. The second is the blueprint or
DCML is designed to be used in a number of scenarios. First is by automation systems to codify descriptions of environments and the formerly manually interpreted rules governing the management of those environments. Second is by traditional and automated management systems alike to describe and exchange their views on and representations of managed environments.

DCML is a data format and corresponding data model. It is not a protocol or application program interface. Therefore, it specifies how to represent information, not how to transfer, access, create or modify information. Future work in these areas is possible, but for now, DCML defines the first step in modern management system information sharing.

## 2 Framework Overview

This section provides an overview of the DCML framework and its typical use. It is the intention of this section to give the reader the background required to navigate through rest of the specification.

DCML is a data format and corresponding data model for exchanging information among management systems. Systems exchange information in a well known vocabulary and a well defined format. The DCML framework specification defines the conceptual data model in which data center elements are described, how this data model is extended to represent specific elements, how the conceptual model is encoded into DCML document instances, and processing rules for interpreting DCML document instances.

### 2.1 Conceptual Model

The DCML conceptual model must be flexible enough to capture knowledge of both current and future managed environments and the policies and procedures governing the management of those environments. Therefore, the conceptual model consists of two layers – a core schema and extension schemas.

- The core schema consists of definitions of common classifications: an environment section for capturing information about the managed environment (instances of managed things); a blueprint section for capturing information about abstract idealized managed environments (classes of managed things); and a rules section for capturing policies that govern change in the management environment.

- Extension schema is the DCML mechanism to describe new types in the changing management environment. As DCML grows, the number of extension schemas also grows. Extension schemas consist of entities that capture domain-specific knowledge such as configuration parameters for a hardware load-balancer. Entity types in
extension schemas cut across one or more common classifications in the core model. In addition, extension schemas can introduce additional classifications to capture shared knowledge across sub-domains of knowledge.

2.2 DCML Encoding Scheme

A DCML model of a data center environment must be encoded into a concrete representation before it can be exchanged with another management system, written to a file, or transported by a protocol. DCML schemas are Web Ontology Language (OWL) vocabularies. OWL provides an extensible and flexible language to describe large and complex environments as a knowledge representation. The choice of OWL is obvious when one considers the uses to which DCML will be put. DCML documents do not merely describe static information about the simple state of an environment. DCML documents often describe complex entities and relationships and the rules governing the management of those entities. DCML must do this in a way that facilitates automated reasoning about the data. This is exactly the purpose for which OWL was designed.

2.3 DCML Instance Document

DCML core and extension schema definitions and instances of these schemas are encoded in RDF/XML documents. RDF/XML provides an extensible and flexible structure to capture large data populations. DCML instance documents may contain encoded DCML elements and non-DCML elements. Non-DCML elements allow proprietary or vendor-specific information intended for use by bi-lateral agreement to be transported alongside standard information. At a high level, a DCML instance document looks like this:

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Schema Definitions</td>
</tr>
<tr>
<td>Extension Schema</td>
</tr>
<tr>
<td>DCML Processing</td>
</tr>
</tbody>
</table>

The header refers to meta-document information such as document author and creation date. Core schema definitions refer to optional inclusion (using the OWL import mechanism) of core DCML entity class definitions. Extension schema definitions refer to optional inclusion of DCML extension entity class definitions. DCML instance data refer to optional instance information in a non-schema document.

Management systems participating in DCML information exchange can export, import, or export and import DCML instance documents. The exchanged DCML instance document can contain either part or all of the DCML knowledge known by the exporting system. Both the exporting system and the importing system must process the DCML instance document according to the semantics of the conceptual model and additional processing rules specified in this specification and DCML extension specifications.
3 Design Goals

This section enumerates the high level goals of DCML.

3.1 Interoperability

In a complex environment such as a data center, change is a constant. Organizations cope with this complexity through a variety of management systems and tools. Use of multiple management system is born out of necessity rather than desire. It’s difficult for individual management systems to keep up with this constant change, especially in the area of configuration, so organizations are constantly searching for management solutions along with new supported entities. One obvious undesirable effect of having multiple deployed management systems is inconsistent, including overlapping and/or missing, information about the managed environment. For example, monitoring systems have a limited view of what they can monitor but not necessarily the same view as an asset tracking system. As a result, some IT organizations purchase or build tools to feed asset information to monitoring systems. This type of information sharing between management systems grows exponentially as the number of management systems grows. This leads to one of the goals of DCML – interoperability. DCML provides a common language that can be used by the management systems to express their knowledge of the managed environment. In an environment where management systems speak the same language, integration between the management systems is simplified.

3.2 Visibility

Lack of consistent information about a managed environment can be a hindrance to business decision making. A common operations task is to reconcile the number of deployed and number of requested servers for a customer. In order to get this information, operations personnel might create data collection tools to gather information from provisioning systems and asset management systems. Once the data is collected, data manipulation tools are used to uniform the data and generate a report. Similar to the integration problem, data center operations create multiple data collection and manipulation tools to gather data about and make sense of the managed environment. This leads to another goal of DCML – visibility. DCML improves the quality of information and the means to acquire information by providing a common data format.

3.3 Enable Automation

In an environment where management systems speak the same language, higher level automation can be achieved. An automation system can aggregate knowledge from various management systems and make configuration decisions based on the aggregated knowledge. For example, based on information from a configuration system, an automation system can decide how to program a monitoring system to monitor a configured entity, or how to respond to changing load. Another example is disaster recovery. Based on collected configuration information, an automation system can determine how entities should be rebuilt following a...
catastrophic failure. This leads to another goal of DCML – *enable automation*. A DCML vocabulary can describe the managed environment in a way that enables tools to automate cross system management tasks such as rebuild after catastrophic failure.

3.4 Extensibility
The managed environment is constantly changing. New applications, system software, and hardware types are routinely added to a managed environment. With each new managed entity introduced, new knowledge about how the entity is to be managed is also introduced. The DCML framework must define how new schemas are created to describe the information necessary to manage these new entities.

3.5 Flexibility
One primary feature all DCML compliant management systems need to implement is to translate native knowledge representations to and from DCML representations. Both DCML syntax and semantics need to be flexible in describing both logical and physical concepts. For example, logical concept describes “B2C server rack,” and physical concept describes “Catalyst 6500 Switch”. In addition, DCML processors need to be flexible in supporting multiple platforms.

3.6 Scalability
A managed environment can be very large, encompassing thousands of physical entities. In addition to each physical entity there are numerous policies, configurations, and software that need to be tracked. As a result the amount of knowledge about a managed environment encoded in DCML can be quite large. DCML accommodates large data populations by allowing partial knowledge exports/imports. This means a management system can export a subset of its knowledge, typically about a limited number of DCML types, or about a limited number of managed entities.

3.7 Security
Management systems often contain sensitive information about a management environment. One obvious example is security credentials such as passwords. DCML provides facilities to allow portions of the information represented to be encrypted and/or signed.

3.8 Installed Base
A final important design goal of DCML is to be able to work with the installed base of technologies, management systems, and other management standards that make up today’s data centers. DCML must not assume new infrastructure or technology upgrades to be useful. It should not duplicate information already in use in an environment via existing management standards, such as CIM or SNMP. Instead, it must be designed to incorporate the installed base of technologies and standards and build on them to introduce new functionality.
4 Design Approach

DCML is a data oriented framework that ties together a complex management environment. This framework consists of a conceptual model, syntax for the conceptual model and its instances, and a set of processing rules for instance documents. The following discusses the approach DCML takes on these parts of the framework.

4.1 Practical Ontology

DCML takes an ontology approach toward modeling and capturing information about the managed environment. Practically, ontology is a way to represent concepts and relationships which collectively describe a domain. For example, concepts of software installation, asset tracking, service monitoring, etc. characterize the ‘systems management’ domain. Once such ontology is defined, various applications can use this information to perform tasks in the domain. For instance a business analytics system can utilize information from server provisioning, application configuring, and ticketing to determine relative support cost among application platforms.

A useful analogy can be made with object oriented models to enhance the understanding of ontology, particularly in context of DCML. Ontology is similar to object oriented models in they both capture part of the real world for machine processing. They both build on the idea of classes and relationships with data types and aggregations. They differ however in their motivation:

O-O models are abstractions; they eliminate or simplify concepts and relationship down to those specifically relevant to the task at hand (usually the design of some IT system), whereas ontologies are intended for knowledge representation in which the goal is “if something is known, it should be able to be recorded in a machine-interpretable manner” …[OMD]

Another distinction is object oriented models do not distinguish between the general case and the full range of possibilities. For example, how often will the optional “description” attribute of a class have a value? Ontology, on the other hand, describes the full range of possibilities.

Ontology languages such as the Web Ontology Language [OWL] provide rich language constructs. In OWL, classes can be formed based on set constructs. For example, a class can be the union of two or more classes. Ontology languages also support the notion of evolvability, where a data set conforming to one schema can be transformed to conform to another schema. In general, ontology languages provide inference abilities to late-bind data populations to schemas. This capability allows more sophisticated data manipulation at the target system. These ontology constructs provide DCML with rich expressiveness.

Practically, these ontology features simplify the job of DCML processors in translating DCML into native representations. For example, a monitoring system importing a DCML document in
search of applications to monitor may internally represent a DCML Application class with a similar but differently named class. The monitoring system can tell the processor its native class is equivalent to the DCML Application class. The processor can subsequently return instances of the DCML Application Class as instances of the native class. In essence, the ontology language can be leveraged for transformation.

4.2 The Relation to Semantic Web
The DCML ontology represents shared knowledge of a managed environment where a multitude of management systems and tools participate in sharing their domain knowledge. While there is not necessarily an aggregated physical store of knowledge, one can view all management systems and tools that participate in this information sharing as a logical repository of knowledge. That is, the management systems are parts of a loosely distributed knowledge base. Loosely distributed in the sense there is no direct knowledge of peer existence by a managed system.

The W3C Semantic Web views the World Wide Web as a vast knowledge store and created a framework to describe resources in a machine process-able form at the scale of the World Wide Web. The framework is centered on Resource Description Framework [RDF]. RDF provides a simple and general assertion model for data representation and encoding rules for externalizing data. A schema layer, Resource Description Framework Schema [RDFS], adds constructs for inheritance and constraints and adds inference rules for interpreting RDF data populations. An ontology layer, Web Ontology Language [OWL], adds tighter constraints and more powerful inference rules. Together, the three layers form a knowledge representation based framework to describe structured data on the web.

The DCML view of the environment is very similar, not by coincidence, to W3C Semantic Web’s view of knowledge on the World Wide Web. In a sense, DCML is a microcosm of the Semantic Web.

4.3 Note on Encoding
DCML knowledge is encoded in XML instance documents for exchange between management systems. The structure of the instance document is described by Resource Description Framework [RDF] RDF/XML encoding rules. This is due primarily to DCML’s adoption of the Semantic Web efforts (OWL and RDF) and XML schema’s limited semantics. Early prototypes of DCML showed that the nested nature of XML schema does not lend well to a non-hierarchical relational model. In order to encode non-hierarchical references, a DCML processor would have to add reference semantics and encoding. RDF/XML provides the semantics (via RDF) for encoding relational models which reduce the requirement on DCML.

5 Conceptual Model
The DCML conceptual model consists of two layers, a core schema and extension schemas. The core schema described in this section sets the foundation for extension schemas. The core schema and extension schemas are described using the semantics of the DCML meta-model. The following sections describe the DCML meta-model, the core schema, and other general issues surrounding the conceptual model such as schema naming and entity references into non-DCML schemas.

5.1 DCML Meta-Model
The DCML conceptual model is a vocabulary of W3C Web Ontology Language [OWL], specifically OWL Full. OWL is a Resource Description Framework Schema [RDFS] vocabulary extension for describing properties and classes. The following sections are informal descriptions of OWL constructs commonly used in a DCML schema. These constructs are a subset of structure and basic elements of OWL Lite that provide common class and property definitions. See [OWL] for a complete element reference.

5.1.1 DCML Document Structure
The DCML conceptual model is represented in a DCML document, to be more specific, a DCML schema document. A schema document defines the DCML vocabulary in which DCML instance documents exchange information. Both schema and instance documents share a common format and structure.

A DCML document is an electronic file containing XML elements. A DCML document structure consists of a top level rdf:RDF element that describes element namespace references. The rdf:RDF element optionally contains a dcml:Header element followed by zero or more DCML elements or meta-model elements.

The following is a simple example of a DCML schema document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [ 
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"> 
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#"> 
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#"> 
<!ENTITY owl "http://www.w3.org/2002/07/owl#"> 
<!ENTITY dcml "http://www.dcml.org/ns/dcml/1/core#"> ]>

<rdf:RDF xmlns:rdf = "&rdf;"
  xmlns:rdfs = "&rdfs;"
  xmlns:owl = "&owl;"
  xmlns:dcml = "&dcml;"
  xmlns = "http://www.dcml.org/ex/1/extension#"
  xml:base = "http://www.dcml.org/ex/1/extension">
  <dcml:Header rdf:about=""/>
  <rdfs:comment>Example DCML extension schema</rdfs:comment>
</rdf:RDF>
```
<rdfs:label>Example</rdfs:label>
<dcml:author>DCML Framework Working Group</dcml:author>
<dcml:createDate>2004-04-22T01:08:30.711Z</dcml:createDate>
<owl:imports rdf:resource="http://www.dcml.org/ns/dcml/1/core"/>
</dcml:Header>

<owl:Class rdf:ID="CIMOperatingSystem">
<rdfs:comment>An Operating System resource defined by CIM.</rdfs:comment>
<rdfs:subClassOf rdf:resource="&dcml;Environment"/>
<rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>
</owl:Class>

<owl:Class rdf:ID="LogicalServer">
<rdfs:comment>A DCML logical server.</rdfs:comment>
<rdfs:subClassOf rdf:resource="&dcml;Resource"/>
<owl:Restriction rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
</owl:Class>

<owl:ObjectProperty rdf:ID="os">
<rdfs:comment>A logical server to OS relationship.</rdfs:comment>
<rdfs:domain rdf:resource="#LogicalServer"/>
<rdfs:range rdf:resource="#CIMOperatingSystem"/>
</owl:ObjectProperty>
</rdf:RDF>

In the example above, lines 2-8 define a set of XML entities to be referenced by other elements in the document.

Lines 9-14 show a top level element providing namespace reference information. All DCML documents top level element is rdf:RDF.

Lines 15-21 show a DCML header element providing information about the document. The header element optionally embed owl:imports sub-elements to accumulate information defined in other DCML documents. The owl:imports mechanism is a document inclusion mechanism for DCML documents.

Lines 23-27 define a DCML class – CIMOperatingSystem. The CIMOperatingSystem class inherits two classes described in the DCML core schema document – Environment and CIMEntity classes.
Lines 29-38 define the LogicalServer class and constrain the cardinality of an os property to 1. This indicates there can be no more than 1 os property for any instance of LogicalServer.

Lines 39-43 define the os ObjectProperty. An ObjectProperty joins an instance of a class to another. The rdfs:domain element specifies the os property’s “source” class type must be a LogicalServer and the rdfs:range element specifies the os property’s “destination” class type must be a CIMOperatingSystem class instance.

5.1.2 Common DCML Document Elements
Every DCML document share a set of common element types. These include the top level rdf:RDF element and the dcml:Header elements.

The top level element of any DCML document is the rdf:RDF element. The rdf:RDF element describe namespaces that are used by terms in the document. In addition, it can also describe what namespace the document is to be bound.

```
<rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns = "http://www.dcml.org/ex/1/extension#"
    xml:base = "http://www.dcml.org/ex/1/exaension">
```

xmlns:rdf says the all element prefix with rdf: refers to elements defined in the http://www.w3.org/1999/02/22-rdf-syntax-ns# namespace. The xmlns line says for all elements that is not prefixed with a namespace, use http://www.dcml.org/ex/1/extension# as the default namespace. Finally, xml:base says map the document to the http://www.dcml.org/ex/1/exaension namespace.

All namespace attributes are optional. If not present, and an element being used is defined in a different document, then the absolute name of the element (<namespace uri>#<name>) must be used to denote the element.

A header element in DCML documents provides information about the document itself such as its creator and when it was created. In addition it also contains import statements for the inclusion of other DCML documents. If a document refers to elements defined in a second document then it must import the contents of the second document. Import is done in addition to mapping elements from a document to a namespace as described above. An example would be an instance document in reference to a schema document.

```
<dcml:Header rdf:about="">
    <rdfs:comment>Example DCML extension schema</rdfs:comment>
    <rdfs:label>Example</rdfs:label>
    <dcml:author>DCML Framework Working Group</dcml:author>
```
The header element itself is defined in the DCML core schema (see section 5.2). The `rdfs:comment` element provides a human readable description of the document and `rdfs:label` provides a human readable name for the document. `Dcml:author` provides the author of the document. `Dcml:createDate` provides the document’s creation date. Finally, `owl:imports` element says to import the contents of http://www.dcml.org/ns/dcml/1/core into the document. There can be multiple import elements importing elements from multiple sources.

### 5.1.3 DCML Schema Elements

A DCML schema document describes the DCML instance document vocabulary. This means most schema documents contain classes and property definitions. This section describes class and property elements and how they’re used to describe the DCML vocabulary.

#### 5.1.3.1 Class

A class in DCML is similar to a class in an object oriented language in it represents a type. Since DCML is a data language, a class in DCML only has data members but no methods.

```xml
<owl:Class rdf:ID="CIMOperatingSystem">
  <rdfs:comment>An Operating System resource defined by CIM.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="&dcml;Environment"/>
  <rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>
</owl:Class>
```

The `owl:Class` element defines a class. `Rdf:ID` attribute specifies the class name in the namespace of the schema document. A class can contain a human readable comment. More specific classes can be formed through subclassing. A class can be a subclass of zero or more classes. `Rdfs:subClassOf` element specifies a subclass. `Rdf:resource` attribute reference a parent class by its URI. The string “&dcml;Environment” is a short hand for http://www.dcml.org/ns/dcml/1/core#Environment.

An instance of a class is an element in a DCML instance document:

```xml
<CIMOperatingSystem rdf:ID="os"/>
```

#### 5.1.3.2 Property

Properties represent facts about a class. A property is similar to a class data member in object oriented languages. A property is a binary relation and two types of properties are distinguished.
• Datatype properties, relations between instance of classes and literals.

```xml
<owl:DatatypeProperty rdf:ID="name">
   <rdfs:domain rdf:resource="#CIMOperatingSystem"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

• Object properties, relations between instances of two classes.

```xml
<owl:ObjectProperty rdf:ID="os">
   <rdfs:domain rdf:resource="#LogicalServer"/>
   <rdfs:range rdf:resource="#CIMOperatingSystem"/>
</owl:ObjectProperty>
```

There are a number of ways to restrict a relation. The most common and discussed here are the rdfs:domain and rdfs:range. Rdfs:domain specifies the class in which the property applies. Rdfs:range specifies the type of the property. A property may have many rdf:domains which means the property refers to an element that is an instance of the union of all rdf:domains.

In addition to providing type checking, rdfs:domain and rdfs:range can also allow one to infer a type from a property.

```xml
<owl:Thing rdf:ID="m0067">
   <os rdf:resource="#os1"/>
</owl:Thing>
```

It can be inferred that m0067 is a LogicalServer because the domain of os is a LogicalServer.

Datatype properties may range over simple types defined in accordance with XML Schema datatypes. The following simple XML Schema datatypes are recommended for use.

```xml
xsd:string, xsd:normalizedString, xsd:Boolean, xsd:decimal, xsd:float, xsd:double
xsd:integer, xsd:nonNegativeInteger, xsd:positiveInteger, xsd:nonPositiveInteger,
xsd:negativeInteger xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong,
xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:hexBinary,
xsdbase64Binary, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear,
xsqMonthDay, xsd:day, xsd:Month xsd:anyURI, xsd:token, xsd:language,
xsd:NMTOKEN, xsd:Name, xsd:NCName
```

5.1.3.2.1 Cardinality Restriction

The number of times a property is applied to a class instance can be restricted through the use of cardinality restrictions. If no cardinality restriction is specified a class may have zero or more instances of a property. Owl:minCardinality specifies that there can be minimally some number of instance of a property. Conversely, owl:maxCardinality specifies that
there can be no more than some number of instances of a property. *Owl:Cardinality* specifies there must be exactly some number of instances of a property. In combination, the cardinality restrictions permit a schema to constrain to “at least x”, “no more than x”, and “exactly x” where x is the number of instances.

```xml
<owl:Class rdf:ID="LogicalServer">
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#os"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The above example states the class LogicalServer is a subclass of an anonymous subclass that has a restriction on the property os. The restriction is a cardinality restriction that restricts the number of instances of the property to 1.

### 5.1.4 Graphical Representation

An abbreviated informal graphical notation is used in this document to help describe modeling concepts and how entities and relationships can be applied. The following is a description of the graphical constructs for this notation.

- A rectangle represents an entity class.

- Overlapping rectangles represents many instances of an entity class.

- A directional arrow represents a relationship between two entity classes. The direction of the arrow indicates a source entity class contains a relationship to a destination entity class.

*Note on visual formalism:* There are various proposals under way to extend the Unified Modeling Language [UML] to provide graphical notation for ontology. OMG is entertaining several RFPs for the Ontology Definition MetaModel (ODM). Once OMG formalizes ODM, DCML will decide on the adoption of a graphical notation for both core and extension schema specifications.
5.2 DCML Core Schema

The DCML core schema describes core entities and classifications that are commonly useful to all schemas. Core entities consist of extensible classes that capture common concepts used by extension schemas. These extensible classes include entity, relationship, and group. Core entities details are described below.

Classifications captures classifies types of information described in DCML. DCML classifies information into three classifications: environment, blueprint, and rules. The three classifications are represented by top level OWL class elements:

```xml
<owl:Class rdf:ID="Environment"/>
<owl:Class rdf:ID="Blueprint"/>
<owl:Class rdf:ID="Rules"/>
```

In general, the environment classification consists of entities in a managed environment and their relationships to each other. These entities include, but are not restricted to, servers, services, applications, network devices, and storage devices. The blueprint classification consists of entities that describe how entities should be instantiated in a managed environment. For example, a blueprint for a web-service may specify the applications, servers, load-balancers, and initial configuration for each component that makes up a web-service. This blueprint can be used by a provisioning system to then instantiate the service in a managed environment. The rules classification consists of set of rules that describe how changes in the physical environment affect the configuration of resources in the managed environment. For example, a rule states if the number of connections to a web-server resource, add a web-server instance into the cluster. All three classifications are explained in detail in the following sections.

This diagram shows the usage relationship between the three classifications. The environment classification describes resources and their configuration in the managed environment.
environment. The blueprint classification describes templates that contain best-practice policies in form of default resource configurations. Templates are used to instantiate resources in the environment. Once a resource is created from a template, subsequent changes to the template are not reflected on the resource. Configuration changes are made directly against a resource in the environment. The changes can be initiated by humans or automated rules defined in the rules classification. The rules classification describes rules to alter configuration of resources. A rule contains a condition and an action. A condition is an observable event in the managed environment such as disk utilization exceeding 95%. An action is a sequence of resource configuration changes. Rules can be used to describe service level agreements (SLA) and management policies.

Extension schemas should be used to introduce additional classifications in DCML only if the two top level classifications do not adequately cover the new entity types. An extension schema classifies an entity within one of the three core classifications by explicitly subclassing entities beneath one of the three classifications or by using ontology mapping rules to describe mapping of schema entities to one of the classifications.

The DCML core schema is bound to the http://www.dcml.org/ns/dcml/1/core namespace.

In addition to the three classifications, the core schema defines entities in each of the classifications. The following sub-sections describe class and property definitions in each of the classifications. A complete reference to the core schema can be found in section 10.

The initial version of DCML will concentrate on the elaboration of environment classification with blueprint and rules classifications to follow in later releases.

5.2.1 Core Entities
Core entities capture common building blocks used by extension schemas. These building blocks describe key concepts of DCML. The following sub-sections describe individual core classes and other sections in the section provide examples on how they may be used.

5.2.1.1 Header Class
The header class is an extension of the OWL Ontology class and is designed to provide creation information about a schema or instance document. The two properties in the header class are author, the name of the document author, and creationDate, the creation date and time of the document.

5.2.1.2 Entity Class
An entity is a base DCML type. Entities can have complex relationships to each other (described below). The Entity OWL class has no default properties and it’s meant to be extended by DCML classes.
5.2.1.3 NonDCMLEntity Class
The NonDCMLEntity class represents a wrapper of non-RDF encoded information. The information wrapped by a NonDCMLEntity instance is opaque to DCML and DCML processors. Non-DCML entities range from large binary software packages to information encoded in non-DCML standards such as DMTF CIM objects and SNMP MIBs. DCML can either embed or refer to these entities. For example, a NonDCMLEntity can refer to an application package via a file URI or a NonDCMLEntity can embed to an XML stanza in a DCML instance document.

5.2.1.4 CIMEntity Class
The CIMEntity class is a NonDCMLEntity class that maps to a CIM object class. An instance of the CIMEntity embeds a CIM object instance encoded in CIM/XML. The primary purpose of this class is to facilitate information bridging between DCML and DMTF CIM. Extension schemas can define specific class mappings. For example a schema can define a CIMOperatingSystemEntity to map a CIM OperatingSystem class.

5.2.1.5 Relationship
Relationships between entities are represented as OWL properties in DCML. For example:

![Diagram of entity relationship]

Here, entity-A contains a relationship-R to entity-C. This means the entity class A has an OWL object property that refers to a class C, similar to a class member in an OO language.

5.2.2 Core Environment Entities
The DCML environment classification consists of resources and relationships between resources in a managed environment.

A resource is an entity comprised of primitive attributes and relationships to other resource(s). For example, a resource may be a CPU, NIC, Disk, OS, or an application. Resources can be extended (i.e. sub-classed) to form more complex entities.

Using the resource and relationship constructs, extension schemas can create specific definitions. The following is a simple static diagram example that shows resource and relationship definitions for a logical server resource. The formal definition of the entities
and relationships in the example below is provided by DCML extension schemas. Namely, the following example is provided by the DCML server extension.

5.2.2.1 Example: Three tier system model

The following is a more complicated example of how environment primitives can be used to model a common three tiered system.

Note the model below is one of many ways services may be modeled in DCML. The purpose of the diagram is to illustrate how the framework can be utilized to describe entities relevant to DCML but not the formal description of the DCML entities.
The web, app, and db tiers are modeled as individual services. Each service contains one or more logical servers which host applications in aggregate to provide a service. Services are tied together through application configuration entities. For example, the Apache Config entity has a relationship to the App Service. In the diagram above, relationships, in the general, are used to create a hierarchical service structure. A relationship also implies a configuration (described as attributes of the entity) exists between the source and the destination entity. In case where finer grain relationship is required, such as relationship between services, specific relationship types can be used.
5.2.3 Core Blueprint Entities

The blueprint consists of templates, relationships, and configurations that describe how applications and services should be instantiated in an environment. A template consists of attributes and relationships to configurations and is used to describe the configuration of services. (See diagram at end of this section for an example.) Configurations include both default settings and requirements for a resource in context of a template. For example, a load-balancer configuration for a template may contain the DNS name of a service and the requirement that it must support hot backups. Finally, relationships tie templates, configuration, and resources together.

The goal of the blueprint is to allow systems or humans to define what components a service consists of and how it should be configured to provide the service. Once a service is instantiated from a blueprint subsequent configuration changes in the blueprint do not affect the instantiated service. Configuration changes to an instantiated service should be made on the corresponding environment entities.

The following is an example of a static template diagram that describes a simple web content service and instantiation of the template in the environment. The web content service consists of one or more load-balancers fronting the public DNS name of the service and a cluster of web-servers that host the actual contents.

The blue colored boxes indicate entities classified as blueprint entities and white boxes indicate entities classified as environment entities. Solid lines indicate relationships within a classification and dotted lines indicate relationships between classifications.

The formal definition of the following entities and relationships is provided by DCML extension schemas. Namely, they are provided by the DCML service and application schema extension, the DCML server schema extension, and the DCML network schema extension.
(A) A web service template ties together the service definition.

(B) A relationship binds a service template to a load-balancer configuration. This relationship indicates the service has a load-balancer.

(C) A relationship that indicates the service consists of a webTier with one or more web servers.

(D) A load-balancer configuration that defines the number of load-balancers (an attribute of the configuration) in the cluster.

(E, F) A relationship binds load balancer configuration to a name resource. The name
resource is an assignable resource defined in the environment.

(I) A server configuration that defines the number of servers in the cluster (a server configuration attribute) as well as its requirements indicated by J, L, and N.

(J, K) A configuration relationship that says, the server must have a physical server type resource defined by (K). The physical server resource is valid physical server resource defined in the environment.

(L, M) Similarly, this configuration relationship states that the server must have an OS defined by the OS resource (M).

(N, O) This relationship states, the server requires applications configured with (O). Application configuration describes how an application should be configured, for example, on Unix systems, what base directory should be application be installed in. Or, what listening port the application should use.

(O, P, Q) Each application configuration ties to an application resource definition in the environment.

(R) A load balancer configuration that indicates a communication relationship between service load balancers and its servers.

(S) A relationship that says services indicated by (T) are instantiated from template rule (A).

(T) An instantiated service resource in the environment. A service resource is a container of all resources in unison provides a logical service.

(U, V) A relationship that indicates the service contains a logical server group.

(V) A logical server group resource that contains one or more logical server resources.

(W, X) A relationship binds a set of logical server resources to a server group resource.

(X) A logical server resource represents a server abstraction.

(Y, H) A relationship that indicates the service T is fronted by a load-balancer resource.

(Z, X) A relationship that indicates the load-balancer makes network connections to the logical server X.

(A1, M) A relationship that indicates logical server X contains the operating system M.

(A2, K) A relationship that indicates the logical server X is overlaid on the physical server K.
(A3, X) A relationship that indicates the application Q is installed on the logical server X.

5.3 Core Rules Entities
Core rule entities will be defined in a future release of the framework specification. In this release of the framework specification, the core rule classification is reserved for future use.

5.4 Extending the Schema
Extending the DCML core schema means creating an OWL schema and extending classes defined in the DCML core schema. The following is an example of simple extension schema. The schema describes a LogicalServer with a relationship to a CIMOperatingSystem. The core schema definition can be found in section 10.

The following diagram is a graphical representation of the schema defined below.

![Diagram of LogicalServer and CIMOperatingSystem relationship]
Lines 2-6 provide XML entity definitions.
• Lines 7-12 provide a RDF header that includes namespace definitions for various dependent schemas. Line 5 is a reference to the DCML core schema version 1. Line 6 binds this schema to the http://www.dcml.org/ex/dcml/1/exampleSchema namespace.
• Lines 13-18 is a DCML header describing the author and creation date of the schema.
• Lines 19-23 describe the CIMOperatingSystem, a CIM OperatingSystem class wrapper entity.
• Lines 24-33 describe the LogicalServer with a cardinality constraint of 1 on a os property.
• Lines 34-39 describe an os property that refers to the CIMOperatingSystem class.

5.5 Schema Naming and Versioning
Schemas inevitably change due to numerous factors which include errors, completeness, and changes in the world they represent. As schemas change there needs to be a way to easily identify the change and the rules for operating in an environment where multiple versions may co-exist. The following describes a simple DCML naming and versioning scheme.

DCML schemas, like XML, use XML namespaces for versioning and naming. Each DCML schema is bound to an XMLBASE represented by a valid URI. All DCML schemas must be bound to http://www.dcml.org/ns/dcml/<version>/<schema>, where <version> is a DCML version string with a numeric version number, and where <schema> is the name of the extension schema. Extension schemas are part of a single version of the DCML schema set. There are no subversions for individual DCML schema versions. Further, http://www.dcml.org/ns/dcml namespace is reserved for only DCML schemas. Non-schema instance documents must not use this namespace. The following is an example of a valid DCML schema URI: http://www.dcml.org/ns/dcml/1/core.

6 DCML Instance Document
DCML processing involves the import and export of the DCML instance document. A DCML instance document is an encoding of some knowledge about the managed environment. Knowledge is described with the vocabulary defined in the DCML conceptual model and encoded in RDF/XML [RDF].

An instance document has the following properties.

- An instance document contain both instance and/or schema data – per RDF.
- An instance document must be an instance of a DCML schema – this provides a minimal level of recognized knowledge by the importing processor.
- An instance document can contain non-DCML schema data for system specific payload. Payload data described by an extended schema must refer to a valid reachable (by the importer) schema definition or embedded schema definition.

- An instance document can contain partial knowledge of a DCML schema. That is, while an instance document must be a valid DCML document, it need not reference all entities defined in the schema.

- An instance document can contain a subset of all knowledge known by the exporting system. Exporters may provide methods for exporting partial knowledge and importers can process partial knowledge according the import processing rules discussed in the next section.

- An instance document must be encoded in RDF/XML and not other RDF formats. This reduces the number of formats (e.g. N3, N-Triples, XML-ABBREV) the import/export processor would have to deal with.

- All instance documents must be valid DCML documents (i.e. they must be successfully validated against a DCML schema). RDF does not perform systematic validation by default; this is a requirement specific to DCML.

- XML stanzas in an instance document may be encrypted according to encryption rules specified by the XML Encryption Syntax and Processing specification [reference?].

- In RDF, and by extension DCML, instance and schema documents are both encoded as RDF/XML. In DCML, instance and schema differ by the namespace where the document is bound. DCML schema documents must be bound to a namespace specified in the Conceptual Model section. Non-schema instance document must not be bound to any namespace under http://www.dcml.org/ns/dcml/.

The following is an example of a DCML instance document. The instance document is an instance of the exampleSchema described in section 5.4.

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dcml=rdfs="http://www.dcmlw3.org/ns/dcml/1/core2000/01/rdf-schema#"
         xmlns:owl="http://www.dcmlw3.org/ex/dcml/1/exampleSchema#">2002/07/owl#"

<dcml:Header rdf:about=""
            xmlns:dcml = "http://www.dcml.org/ns/dcml/1/core#"
            xmlns = "http://www.dcml.org/ex/1/extension#">
8  <dcml:Header rdf:about="">  
 9   <rdfs:comment>Example DCML instance document.</rdfs:comment> 
10   <dcml:author>DCML Framework Working Group</dcml:author> 
11   <dcml:createDate>2004-04-22T01:08:30.711Z</dcml:createDate> 
12   <owl:imports rdf:resource="http://www.dcml.org/ex/1/extension"/> 
13 </dcml:Header> 
14 
15  <LogicalServer rdf:ID="myServer">  
16    <dcml:name>MyServer</dcml:name> 
17    <os rdf:resource="#myOS"/> 
18  </LogicalServer> 
19 
20  <CIMOperatingSystemLogicalServer rdf:ID="myOSmyServer">  
21    <dcml:xmlData rdf:parseType="Literal">  
22      <PROPERTY NAME="MaxProcessesPerUser">  
23        <VALUE>32</VALUE>  
24        <PROPERTY NAME="MaxProcessesPerUser">  
25          <VALUE>32</VALUE>  
26          </PROPERTY>  
27        </CIMOperatingSystem>  
28      </PROPERTY>  
29      </CIM>  
30      </dcml:xmlData>  
31  </CIMOperatingSystem>  
32 </rdf:RDF>

- Lines 5-108-14 describe the document header.
- Lines 11-1416-19 describe a LogicalServer instance that contains an os property to a CIMOperatingSystem instance.
- Lines 16-2621-30 describe a CIMOperatingSystem instance and embeds a CIM OperatingSystem instance encoded XML CIM.

7 **Instance Naming**

DCML Framework Specification
29
DCML provides a naming mechanism that addresses the requirement to address an entity in a managed environment and to allow sharing of entity data between management systems.

This section uses the term entity instance to refer to a record that describes some entity in the managed environment. It is the name of the record we are concerned with.

All entity instances in DCML use the following OWL/RDF name scheme:

\[ http://<namespace>/<name> \]

Namespace is a unique identifier (within the management domain of an organization) and can be a well known hostname of a management system. The uniqueness of the namespace is maintained by a 3rd party – either human administrators or a namespace management system. The <name> is a unique name of the entity instance. Uniqueness is within the <name>’s associated namespace. For example, \[ http://foo.com/bar#myServer \] is different from \[ http://baz.com/bar#myServer \]. The combination of \[ <namespace>/<name> \] allows for a unique DCML entity instance name within a management domain. The protocol portion of the instance name is fixed to “http” and the instance name URI may or may not resolve to a valid DCML instance document representing the entity.

Namespace assignment is a configuration task of an administrator. An administrator can choose any strategy for carving out namespaces as long as it does not prevent an entity instance name from being unique. A common strategy will be to use the well known DNS name of the exporting management system as the DCML namespace.

A management system exporting DCML is responsible for creating unique names (within its assigned namespace) that map to DCML entity instances it exports. This name must be consistent (i.e. a name must always match the same entity within the namespace) through the deployed life of the management system.

If an exporting system and a separate importing system both describe the same entity with its own entity instances then the entity will have two different referring entity names. This means duplicate records exist for an entity in the managed environment. It is the responsibility of the importing system to detect duplicates. This can typically be detected by matching required properties of the imported and existing records.

8 DCML Processing

Management systems in the DCML framework exchange information in the form of DCML instance documents. An exporting system translates its knowledge about the managed environment in its native vocabulary into DCML and stores the translated knowledge in a DCML instance document. The DCML instance document is then transported to an importing system that translates DCML to its native knowledge representation. This exchange results in knowledge sharing between the two systems.
The importer and exporter processes (they may actually be components of the systems but are shown as separate processes for illustrative purposes) are translators and DCML processors that handle native knowledge representation to DCML transformation and vice versa. DCML instance documents can be transported to importing systems using any file transfer mechanisms ranging from scp to http and is not dictated by the DCML framework.

### 8.1 Import and Export

DCML compliant management systems must implement DCML import and/or export features to exchange knowledge with each other. (The term import used in this context refers to the act of importing a DCML instance document as opposed to the OWL usage of the term for ontology inclusion). An importer is responsible for translating DCML instance documents from one or more sources into native representation. An exporter is responsible for translating native representation into DCML instance documents.

From an implementation standpoint, the primary responsibility of importers and exporters is translation. Importers and exporters are responsible for implementing vendor specific knowledge translation to and from DCML.

In addition to translation, importers and exporters may handle the case of partial knowledge transfer. A common case calls for an exporter to export a subset of known information. It is not the common case to ask a system to export gigabytes of DCML. Usually a client knows what type of information it is interested in, so it can ask for relevant information.

In order to support this partial knowledge transfer, an exporter needs to support the ability to export a subset of knowledge based on specified criteria. An exporter must support instance fetching, where a client requests all information on a DCML class by a known property value.
For example, a client can ask for10.2.1.2. Also an exporter may support export by type. For example, a client can ask for all known server instances.

An importer handles partial knowledge transfer similar to normal transfers. Importers read a DCML instance document and translate its contents into a native representation. However, a DCML instance document containing partial knowledge may contain dangling references. An instance document with complete knowledge only contains dangling references as error conditions. An importer needs to detect dangling references and perform translation based on local policy. Dangling references may be fine and expected for certain entities but not for others, so it’s the importer’s responsibility to make the translation decision.

8.2 Security
DCML extension schemas may require management systems expose sensitive information about the managed environment. For example, credentials to log onto a server may be part of an instance document. This sensitive information must be protected from an unintended audience.

DCML supports two mechanisms for securing the instance document. A complete instance document can be encrypted by a 3rd party encryption system or XML stanzas in an instance document can be secured using techniques described in XML Encryption [XMLENC].

9 Mapping Existing Schema into DCML

There exists a large amount of standardized knowledge and schemas which describe some aspects of the managed environment. DCML extension schemas should attempt to re-use these standards where appropriate. In addition to the core schema NonDCMLEntity mapping capability, we anticipate a portion of existing schemas such as the Common Information Model [CIM] from Distributed Management Task Force Inc., can be translated into DCML.

Mapping an RDF based schema is nearly a direct translation from RDFS or OWL into DCML. This can be done by using the OWL class and property mapping constructs to move schemas in a non-DCML namespace to a DCML namespace.

Mapping a non-RDF based schema is done on two levels: the schema level and the instance data level. Both levels are required in order to validate information and to enable inference in automation systems. Schema level translation involves translating an existing schema into a DCML schema. Most schemas are either relational or object oriented – both map well into DCML.

10 Core Schema
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [>
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
<!ENTITY owl "http://www.w3.org/2002/07/owl#">
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
]>
<!ENTITY owl "http://www.w3.org/2002/07/owl#">
]

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <rdfs:subClassOf rdf:resource="&owl;Ontology"/>
    <rdfs:Class rdf:ID="Header">
        <rdfs:comment>The Header class captures information about a DCML extension schema and instance document.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="&owl;Ontology"/>
    </rdfs:Class>

    <owl:DataProperty rdf:ID="author">
        <rdfs:comment>The schema author.</rdfs:comment>
        <rdfs:domain rdf:resource="#Header"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DataProperty>

    <owl:DataProperty rdf:ID="createDate">
    </owl:DataProperty>

    </rdf:RDF>

</Header>
</rdf:RDF>

><!-- # Header Definition -->
<!-- # -->
<!-- # -->
</owl:Class>
</owl:DataProperty>
</owl:DataProperty>
<rdfs:comment>The creation date of the schema or instance document.</rdfs:comment>

<rdf:Property rdf:ID="author">
    <rdfs:comment>The schema author.</rdfs:comment>
    <rdfs:domain rdf:resource="#Header"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="createDate">
    <!-- # -->
    <rdfs:domain rdf:resource="#Header"/>
    <rdfs:range rdf:resource="&xsd;dateTime"/>
</rdf:Property>

<owl:Class rdf:ID="Entity">
    <rdfs:comment>Base DCML class.</rdfs:comment>
</owl:Class>

<owl:DatatypeProperty rdf:ID="name">
    <rdfs:comment>A non-unique human readable entity name.</rdfs:comment>
    <rdfs:domain rdf:resource="#Entity"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="NonDCMLEntity">
    <rdfs:comment>An entity class that wraps data encoded in a non-DCML format.</rdfs:comment>
</owl:Class>

<owl:ClassDatatypeProperty rdf:ID="NonDCMLEntityencoding">
    <rdfs:comment>An entity class that wraps data encoded in a non-DCML format.</rdfs:comment>
</owl:ClassDatatypeProperty>

<owl:DataProperty rdf:ID="encoding"/>
<rdfs:comment>The encoding format of the wrapped information. This can be "ASN.1", "XML", etc. This property should be applied to subclass of NonDCMLEntity to specify encoding for the subtype. See CIMEntity definition for an example of this.</rdfs:comment>

<rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>

<owl:DataPropertyDatatypeProperty rdf:ID="binaryData">
<rdfs:comment>Base64 encoded non-DCML information used for embedding binary information.</rdfs:comment>
<rdfs:domain rdf:resource="#NonDCMLEntity"/>
<rdfs:range rdf:resource="&xsd;base64Binary"/>
</owl:DataProperty>

<owl:DataPropertyDatatypeProperty rdf:ID="xmlData">
<rdfs:comment>String encoded non-DCML information used for embedding XML information.</rdfs:comment>
<rdfs:domain rdf:resource="#NonDCMLEntity"/>
<rdfs:range rdf:resource="&rdf;XMLLiteral"/>
</owl:DataProperty>

<owl:DataPropertyDatatypeProperty rdf:ID="reference">
<rdfs:comment>A URI reference to information referenced by DCML.</rdfs:comment>
<rdfs:domain rdf:resource="#NonDCMLEntity"/>
<rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DataProperty>

<!-- # -->
<owl:Class rdf:ID="CIMEntity"/>
<!-- # -->
<rdfs:subClassOf rdf:resource="#NonDCMLEntity"/>
<!-- # -->
<rdfs:comment>A CIM entity encoded in XML. The data or reference property is an XML document that describe a corresponding CIM object or object cluster.</rdfs:comment>
<!-- # -->
<owl:Class rdf:ID="CIMEntity">Environment</owl:Class>
<!-- # -->
<rdfs:comment>A CIM entity encoded in XML. The data or reference property is an XML document that describe a corresponding CIM object or object cluster.</rdfs:comment>
<encoding>XML</encoding>
</owl:Class>
92   <!-- -------------------------------------------------------- -->
93   <!-- # -->
94   # -->owl:Class
95   rdf:ID="Resource">
96   <!-- Environment Definition # -->
97   <rdfs:comment>A manageable entity.</rdfs:comment>
98   <!-- # -->
99   # -->
100  <rdfs:subClassOf rdf:resource="#Environment"/>
101  <!-- -------------------------------------------------------- -->
102  # -->owl:Class
103  rdf:ID="Environment">
104  <rdfs:comment>Top level environment classification.</rdfs:comment>
105  <rdfs:subClassOf>
106  <owl:Restriction>
107  </owl:Restriction>
108  # --></owl:Class>
109  <!-- -------------------------------------------------------- -->
110  # -->owl:ClassObjectProperty rdf:ID="ResourcedependsOn">
111  <rdfs:comment>A manageable entity. generic relationship for forming resource dependencies such as installation ordering</rdfs:comment>
112  <rdfs:subClassOf:type
113  rdf:resource="#Environment="#owl;TransitiveProperty"/>
114  <rdfs:subClassOf>range rdf:resource="#EntityResource"/>
115  <owl:RestrictionObjectProperty>
116  <owl:onProperty rdf:resource="#name"/>
117  </owl:RestrictionObjectProperty>
118  # -->owl:cardinality
119  rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
120  # --></owl:Class>
121  <!-- -------------------------------------------------------- -->
122  # -->owl:Class
123  rdf:ID="Blueprint">
124  <rdfs:comment>Top level blueprint classification.</rdfs:comment>
125  </owl:Class>
126  # -->owl:Class
127  rdf:ID="Template"
<!-- # Blueprint Definition # -->
<rdfs:comment>Configuration template.</rdfs:comment>
<!-- # -->
<!-- # -->
<rdfs:subClassOf rdf:resource="#Blueprint"/>

<owl:Class rdf:ID="Blueprint">
  <rdfs:subClassOf rdf:resource="#Entity"/>
  <rdfs:comment>Top level blueprint classification.</rdfs:comment>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Configuration">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Template">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Configuration">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Rules">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:comment>Top level rules classification.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Entity"></owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Configuration">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Rules">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:comment>Top level rules classification.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Entity"></owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Configuration">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<!-- ################################################################# -->
<owl:Class rdf:ID="Rules">
  <rdfs:subClassOf rdf:resource="#Blueprint"/>
  <rdfs:comment>Top level rules classification.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Entity"></owl:Class>

DCML Framework Specification
37
11 References


Appendix A  Implementation Considerations

DCML importers and exporters may share a common DCML processor library. This library handles common tasks such as serializing and deserializing RDF/XML, and provides encryption and decryption of XML stanzas. The following lists the set of features provided by a DCML processor library.

Note: The following sections provide a coarse description of DCML processor features importer and exporter libraries can rely on, and detail design is left to the reference processor implementation. A reference implementation will likely be based on the Jena2 Java RDF library.

Validation
DCML processor, by default, validates a DCML instance document on import and export. This validation can be turned off on demand.
**Process Large Instance Documents**
DCML processor may provide APIs to allow large documents to be stored in a relational database for processing. Storing an instance document in a database allows processing of instance documents which normally would not fit in virtual memory.

**Search**
The DCML processor may provide an API to search for resources in a DCML instance document or across multiple instance documents. The search API may use one of the RDF query languages such as RDF Data Query Language [RDQL], a SQL like language, to search the population data set of instance documents.

**Merging**
The DCML processor may provide an API to merge data sets from multiple DCML instance documents.

**Encryption and Decryption**
DCML processor provides automatic encryption and decryption as described by the XML Encryption Syntax and Processing specification.

**Handling Large Payload**
A management system may have vast knowledge regarding a managed environment. Full exports of this knowledge may result in very large DCML instance documents. In this case, the exporter will have few problems provided it streams the DCML data onto disk. However, importing such a large document can be problematic, especially if an importer wants to perform simple inference or validation on the content. To support cases where the instance document may be larger than available virtual memory, the DCML processor allows processing of DCML instance document in a relational database. The DCML processor can store a DCML instance document in a simple relational database to perform validation and inference.

**Appendix B  A Server Provisioning Use Case Example**

**Background**
There are many processes that can lead to server provisioning - adding capacity, disaster recovery, service provisioning, etc. This use-case concentrates on the creation of a new server as part of service provisioning.

**Problem**
A service is being deployed and part of the deployment requires a web-server to be created. A service architect creates a service blueprint that specifies how the service is to be created - what servers it needs to construct the service and what software installed on these servers. This blueprint is codified in DCML. An automation system or tool orchestrates the construction of the service by passing along
the blueprint (either in whole or in part) to various provisioning systems. A web-server blueprint is passed to our server provisioning system.

Outcome

Our server provisioning system interprets the web-server blueprint, instantiates a server to specification (of the blueprint), and returns a DCML document describing the instantiated server.

Approach

The provisioning transaction involves an input (server blueprint) into the server provisioning system and an output (server instance) from the same system. We will define DCML schema to describe both the server blueprint and the server instance. The network transport can range from simple HTTP transactions (i.e. input and output messages are bodies of HTTP-POST and response messages respectively) to asynchronous Web-Services based messaging to paper request handed to a system administrator. The definition of the network protocol, although important, but not essential to this use-case, will not be discussed further here.

Once the server provisioning system receives the create request, it may go through a series of steps, some of which may involve humans, to satisfy the request. The server blueprint will provide specification that can be satisfied many different ways. In this use-case, a web-server needs to be created. What makes a server a web-server is the application that a server runs. So, the server blueprint only specifies the application it needs to have installed and perhaps an OS (since an application may run on many OSs). It is up the server provisioning system to find the proper hardware (which maybe shared) to host the application. This is where humans may get involved to select the server or intelligent server provisioning system may select the server based on a pre-programmed policy. Once the proper hardware has been selected and provisioned, the specified application is installed. Upon completion of the provisioning process, the provisioning system replies the request with information about the server resource and its components. The returned information should include what’s necessary for down-stream systems to perform its task (e.g. monitoring, patching).
1. Define DCML extension schema to describe a server.

1.1 Define the conceptual server model.

Since the use-case is software centric (i.e. provision server and install web-server), the model below reflects this fact by making software centric field first level attributes. In contrast, a hardware centric use-case would have been to add more physical ram or add more physical storage to the system. Since hardware attributes are not required to satisfy this use-case, hardware related attributes are delegated to CIM definitions. The software attributes reflect what’s necessary to accomplish software installation using known (proprietary or otherwise) software packaging technology.

This is a simple conceptual model of the server that includes only the necessary and essential information to satisfy this use-case and to illustrate relationship between DCML and CIM. As other use-cases are tackled, the following elements are enhanced.

Note: [] is used to indicate cardinality of the attribute. Relationships aren’t explicit in this section. See the OWL schema for explicit relationships.

1.1.1 LogicalServerBlueprint

Description:
This class represents a server without resource constraints. This class is typically used to model the server portion of a service. Modeling is typically performed by a service architect using some design tool. The design process can be done in absence of actual resources. Actual hardware resources are assigned during provision time.

Some characteristics of the logical server blueprint that differentiate it to the logical server:

1. Logical server blueprint is created by a site architect that has minimal insight into the running environment.
2. Logical server blueprint can be used across management domains where the managed environment looks very different.
3. Logical server blueprint specifies only information important to the service.

Attributes:
- subclass of dcml:Configuration
A configuration in the DCML core specification represents is part of the blueprint and it ties configuration to the blueprint classification. This subclassing is purely for organizational purposes.

- os [1]
- package [0 or more]

### 1.1.2 LogicalServer

**Description:**
A logical server represents the current state of a server. In addition, a local server maybe mapped onto 1 or more physical servers.

**Attributes:**
- subclass of dcml:Resource
A resource is a base entity in the environment that can be managed.

- cimComputerSystem [1]
- cimFileSystem [1 or more]
- os [1]
- osPatch [0 or more]
- application [0 or more]
- networkConfig [1]
- managementStatus
  [available, offline, etc.]

### 1.1.3 Application

**Description:**
An application represents a configured instance of a software package. The configuration information is to facilitate monitoring systems or application configuration systems to do their jobs.

**Note:** application overlaps with work being done in the services/applications group. It’s here for completeness of the example.

**Attributes:**
- subclass of dcml:Resource
- package [1 or more]
- protocol [0 or more]
- listenPort [0 or more]
- installDir [1]
- installedOn inverseOf LogicalServer.application

### 1.1.4 SoftwarePackage

**Description:**
A software package describes information about a software package and its location in the managed environment.

**Note:** software overlaps with work being done in the services/application group. It’s here for completeness of the example.
Attributes:
- subclass of dcml:Resource
- vendor [0 or 1]
- version [0 or 1]
- location [1]
- type [1]
- size [1]
- checksum [1]
- requiredDiskSpace [0 or 1]
- os [0 or more]

1.1.5 NetworkConfiguration

Description:
A network configuration describes the IP network configuration of a device network interface.

Attributes:
- subclass of dcml:Entity
- interface [1]
- ip [1]
- dnsName [0 or 1]
- configuredTo inverseOf LogicalServer.networkConfig

1.1.6 OperatingSystem

Description:
The OperatingSystem class represents an installable server operating system.

Attributes:
- subclass of dcml:Resource
- supportedHardwareArch [1 or more]
- packages [1 or more package(s)]

1.1.7 OS Patch

Description:
The OS Patch class represents an installable OS patch package.

Attributes:
- subclass of SoftwarePackage
- patchId [1]
- os [1 or more]

1.1.8 CIM ComputerSystem

Description:
The CIM ComputerSystem class is a DCML class that wraps a CIM ComputerSystem class.
Attributes:
- subclass of dcml:CIMEntity

1.1.9 CIMFileSystem

Description:
The CIMStorage class is a DCML class that wraps a CIM_FileSystem class.

Attributes:
- subclass of dcml:CIMEntity

1.2 Encode the conceptual model in DCML schema (OWL).

This is a first crack at the schema, so limited normalization or strict typing has been enforced.
<owl:Class rdf:ID="LogicalServer">
    <rdfs:subClassOf rdf:resource="#Resource"/>
    <owl:ObjectProperty rdf:ID="cimComputerSystem">
        <rdfs:domain rdf:resource="#LogicalServer"/>
        <rdfs:range rdf:resource="#CIMComputerSystem"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="cimFileSystem">
        <rdfs:domain rdf:resource="#LogicalServer"/>
        <rdfs:range rdf:resource="#CIMFileSystem"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="osPatch">
        <rdfs:domain rdf:resource="#LogicalServer"/>
        <rdfs:range rdf:resource="#OSPatch"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="package">
        <rdfs:range rdf:resource="#SoftwarePackage"/>
    </owl:ObjectProperty>

    <!-- LogicalServer definition -->
    <owl:Class rdf:ID="LogicalServer">
        <rdfs:subClassOf rdf:resource="#Resource"/>
        <owl:ObjectProperty rdf:ID="os">
            <rdfs:range rdf:resource="#OperatingSystem"/>
        </owl:ObjectProperty>
        <owl:ObjectProperty rdf:ID="package">
            <rdfs:range rdf:resource="#SoftwarePackage"/>
        </owl:ObjectProperty>
        <owl:ObjectProperty rdf:ID="osPatch">
            <rdfs:range rdf:resource="#OSPatch"/>
        </owl:ObjectProperty>

        <!-- Restrict cimComputerSystem to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#cimComputerSystem" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>

        <!-- Restrict os to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#os" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>

        <!-- Restrict managementStatus to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#managementStatus" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>
    </owl:Class>

    <!-- LogicalServer definition -->
    <owl:Class rdf:ID="LogicalServer">
        <rdfs:subClassOf rdf:resource="#Resource"/>
        <owl:ObjectProperty rdf:ID="os">
            <rdfs:range rdf:resource="#OperatingSystem"/>
        </owl:ObjectProperty>
        <owl:ObjectProperty rdf:ID="package">
            <rdfs:range rdf:resource="#SoftwarePackage"/>
        </owl:ObjectProperty>
        <owl:ObjectProperty rdf:ID="osPatch">
            <rdfs:range rdf:resource="#OSPatch"/>
        </owl:ObjectProperty>

        <!-- Restrict cimComputerSystem to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#cimComputerSystem" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>

        <!-- Restrict os to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#os" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>

        <!-- Restrict managementStatus to cardinality of 1 -->
        <rdfs:subClassOf><owl:Restriction>
            <owl:onProperty rdf:resource="#managementStatus" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction></rdfs:subClassOf>
    </owl:Class>

    <!-- LogicalServer definition -->
    <owl:Class rdf:ID="LogicalServer">
        <rdfs:subClassOf rdf:resource="#Resource"/>
        <owl:ObjectProperty rdf:ID="os">
            <rdfs:range rdf:resource="#OperatingSystem"/>
<owl:ObjectProperty rdf:ID="application">
    <rdfs:domain rdf:resource="#LogicalServer"/>
    <rdfs:range rdf:resource="#Application"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="networkConfig">
    <rdfs:domain rdf:resource="#LogicalServer"/>
    <rdfs:range rdf:resource="#NetworkConfiguration"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="managementStatus">
    <rdfs:domain rdf:resource="#LogicalServer"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- Application definition -->
<owl:Class rdf:ID="Application">
    <rdfs:subClassOf rdf:resource="&dcml;Resource"/>
    <!-- Restrict package mincardinality of 1 -->
    <rdfs:subClassOf><owl:Restriction>
        <owl:onProperty rdf:resource="#package"/>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction></rdfs:subClassOf>
    <!-- Restrict installDir 1 -->
    <rdfs:subClassOf><owl:Restriction>
        <owl:onProperty rdf:resource="#installDir"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="protocol">
    <rdfs:domain rdf:resource="#Application"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="listenPort">
    <rdfs:domain rdf:resource="#Application"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="installDir">
    <rdfs:domain rdf:resource="#Application"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="installedOn">
    <rdfs:domain rdf:resource="#Application"/>
</owl:ObjectProperty>
<rdfs:range rdf:resource="#LogicalServer"/>
<owl:inverseOf rdf:resource="#application"/>
</owl:ObjectProperty>

<!-- SoftwarePackage definition -->
<owl:Class rdf:ID="SoftwarePackage">
<rdfs:subClassOf rdf:resource="&dcml;Resource"/>

<!-- Restrict vendor maxcardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#vendor"/>
<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict version maxcardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#version"/>
<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict location cardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#location"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict type cardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#type"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict size cardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#size"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict size cardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#checksum"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict requiredDiskSpace maxcardinality of 1 -->
<owl:subClassOf><owl:Restriction>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

</owl:Class>
<owl:onProperty rdf:resource="#requiredDiskSpace"/>
<owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="vendor">
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="version">
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="location">
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="type">
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="size">
<rdfs:comment>Quantity in bytes.</rdfs:comment>
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;long"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="checksum">
<rdfs:comment>Base64 checksum in form of alg:checksum. Where
alg can be crc32 or md4 and checksum is the base64
encoded value of the checksum.</rdfs:comment>
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="requiredDiskSpace">
<rdfs:comment>Quantity in bytes.</rdfs:comment>
<rdfs:domain rdf:resource="#SoftwarePackage"/>
<rdfs:range rdf:resource="&xsd;long"/>
</owl:DatatypeProperty>

<!-- NetworkConfiguration definition -->

<owl:Class rdf:ID="NetworkConfiguration">
<rdfs:subClassOf rdf:resource="&dcml;Resource"/>
</owl:Class>
<!-- Restrict interface cardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#interface"/
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict ip cardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#ip"/
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict dnsName maxcardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#dnsName"/
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction></rdfs:subClassOf>

<owl:DatatypeProperty rdf:ID="interface">
  <rdfs:domain rdf:resource="#NetworkConfiguration"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ip">
  <rdfs:comment>IP address in dotted notation: aa.bb.cc.dd</rdfs:comment>
  <rdfs:domain rdf:resource="#NetworkConfiguration"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="dnsName">
  <rdfs:comment>FQDN</rdfs:comment>
  <rdfs:domain rdf:resource="#NetworkConfiguration"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="configuredTo">
  <rdfs:domain rdf:resource="#NetworkConfiguration"/>
  <rdfs:range rdf:resource="#LogicalServer"/>
  <owl:inverseOf rdf:resource="#networkConfig"/>
</owl:ObjectProperty>

<!-- OperatingSystem definition -->
<owl:Class rdf:ID="OperatingSystem">
  <rdfs:subClassOf rdf:resource="#dcml1;Resource"/>
</owl:Class>

<!-- Restrict supportedHardwareArch mincardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#supportedHardwareArch"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict package mincardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#package"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="supportedHardwareArch">
<rdfs:domain rdf:resource="#OperatingSystem"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- OSPatch definition -->
<owl:Class rdf:ID="OSPatch">
<rdfs:subClassOf rdf:resource="#SoftwarePackage"/>

<!-- Restrict patchId cardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#patchId"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>

<!-- Restrict os mincardinality of 1 -->
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#os"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="patchId">
<rdfs:domain rdf:resource="#OSPatch"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- CIMComputerSystem definition -->
<owl:Class rdf:ID="CIMComputerSystem">
<rdfs:subClassOf rdf:resource="#dcml:Environment"/>
<rdfs:subClassOf rdf:resource="#dcml:CIMEntity"/>
</owl:Class>
© 2005 DCML.org

<!-- CIMFileSystem definition -->
<owl:Class rdf:ID="CIMFileSystem">
   <rdfs:subClassOf rdf:resource="&dcml;Environment"/>
   <rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>
</owl:Class>

1.3 Post extension to DCML.org

Formal process is to be discussed. In the mean time, ask Sara to post the schema document at the appropriate location on www.dcml.org. Also, post this document and related design documents in a document repository on DCML.org. The location of the document repository is TBD.

2. Example use the extension specification

Now the schema is defined, we can illustrate an actual server provisioning transaction. The client provides the server provisioning system with a dcml document that contains an element of LogicalServerBlueprint. The LogicalServerBlueprint explicitly specify the operating system (e.g. Solaris 5.9) and a software package to install (e.g. Apache 1.3.1). The server provisioning system provisions the requested server and application and returns a dcml document that contains a LogicalServer and its related information.

2.3 serverblueprint.dcml

Serverblueprint.dcml codifies a creation request corresponding to the example above.

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY dcml "http://www.dcml.org/ns/dcml/1/core#" >
]>
<rdf:RDF xmlns:rdf = "&rdf;"
xmlns:rdfs = "&rdfs;"
xmlns:owl = "&owl;"
xmlns:dcml = "&dcml;"
xmlns = "http://www.dcml.org/ex/1/extension#"
xml:base = "http://www.dcml.org/ex/1/exaension">
  <dcml:Header rdf:about=""
   <rdfs:comment>Example DCML extension schema</rdfs:comment>
   <rdfs:label>Example</rdfs:label>
<owl:Class rdf:ID="CIMOperatingSystem">
  <rdfs:comment>An Operating System resource defined by CIM.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="&dcml;Environment"/>
  <rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>
</owl:Class>

<owl:Class rdf:ID="LogicalServer">
  <rdfs:comment>A DCML logical server.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="&dcml;Resource"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#os"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</owl:Class>

<owl:ObjectProperty rdf:ID="os">
  <rdfs:comment>A logical server to OS relationship.</rdfs:comment>
  <rdfs:domain rdf:resource="#LogicalServer"/>
  <rdfs:range rdf:resource="#CIMOperatingSystem"/>
</owl:ObjectProperty>

2.4 webserver.dcml

Webserver.dcml codifies the result web-server instance corresponding to the example above.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf = "$rdf;" xmlns:rdfs = "$rdfs;" xmlns:owl = "$owl;"
```
<Pre><code>xmlns:dcml = "&dcml;"
xmlns:srvr = "&srvr;"
xmlns = "http://www.dcml.org/ns/ex/1/webserver.dcml">

<Header rdf:about="">
  <rdfs:label>Apache Web-Server instance</rdfs:label>
  <rdfs:comment>DCML web-server instance document.</rdfs:comment>
  <dcml:author>DCML Framework Working Group</dcml:author>
  <dcml:createDate>2004-07-20T01:08:30.711Z</dcml:createDate>
  <owl:imports rdf:resource="http://www.dcml.org/ns/ex/1/server.dcml"/>
</Header>

<LogicalServer rdf:ID="webserver1">
  <dcml:name>ws1</dcml:name>
  <cimComputerSystem rdf:resource="#cimcs1"/>
  <cimFileSystem rdf:resource="#cimfs1"/>
  <os rdf:resource="#solaris59"/>
  <application rdf:resource="#apacheapp"/>
  <networkConfig rdf:resource="#netconfig1"/>
  <managementStatus>available</managementStatus>
</LogicalServer>

<OperatingSystem rdf:ID="solaris59">
  <dcml:name>Solaris 5.9</dcml:name>
  <supportedHardwareArch>UltraSparc</supportedHardwareArch>
  <package rdf:resource="#sol59pkg"/>
</OperatingSystem>

<Application rdf:ID="apacheapp">
  <dcml:name>Apache application</dcml:name>
  <package rdf:resource="#apache131"/>
  <protocol>http</protocol>
  <listenPort>8080</listenPort>
  <installDir>/cust/apache</installDir>
</Application>

<NetworkConfiguration rdf:ID="netconfig1">
  <dcml:name>le0</dcml:name>
  <interface>le0</interface>
  <ip>192.168.2.3</ip>
  <dnsName>apache.foo.bar.cc</dnsName>
</NetworkConfiguration>

<SoftwarePackage rdf:ID="apache131">
  <dcml:name>Apache 1.3.1</dcml:name>
  <location>http://package.server.com/APACH131.pkg</location>
  <type>Solaris_Package</type>
  <size>204900</size>
  <checksum>md5:abcdefghijklmnopq==</checksum>
</SoftwarePackage>

<SoftwarePackage rdf:ID="sol59pkg">
</SoftwarePackage>
</code></Pre>
<dcml:name>Solaris 5.9 OS package</dcml:name>
<srvr:location>http://package.server.com/SOL59.pkg</srvr:location>
<srvr:type>Solaris_Package</srvr:type>
<srvr:size>1204900</srvr:size>
<srvr:checksum>md5:abcdefghijklmnopa==</srvr:checksum>
</srvr:SoftwarePackage>

<srvr:CIMComputerSystem rdf:ID="cimcs1">
<!-- No CIM_ComputerSystem information provided -->
<dcml:xmlData rdf:parseType="Literal"/>
</srvr:CIMComputerSystem>

<srvr:CIMFileSystem rdf:ID="cimfs1">
<!-- No CIM_FileSystem information provided -->
<dcml:xmlData rdf:parseType="Literal"/>
</srvr:CIMFileSystem>

</rdf:RDF>
Appendix C  Transporting Proprietary Data Example

Background

At times there are proprietary data that needs to be transported between management systems that are specific to a task. This data is not part of DCML due to any of the following:

- The data transport is outside the realm of problems DCML is designed to solve.
- The data schema has yet to make it into DCML.
- The management system is not up to date on the latest version of DCML, so new DCML information appears to be proprietary.

In these situations, there needs to be a way to transport this proprietary data as part of a DCML document exchange between two systems.

Problem

A vendor would like to use DCML to export its configuration database from one deployment of its product and import it into another deployment. This export/import ability is similar to DCML import/export functionality and it helps them reduce initial setup time for a new deployment. However, a majority of the vendors’ dataset is proprietary and specific to the vendor thus not part of the DCML schema.

Outcome

The vendor is able to use the DCML import/export mechanism to transport non-DCML data.

Approach

Since DCML is RDF and RDF can contain any data, DCML can do the same. Our vendor simply creates a DCML instance document and adds elements which are proprietary. These proprietary elements are ignored by the DCML processor and passed thru to the vendor program. For example, webserver.dcml in Appendix B can be considered to carry proprietary information since its schema is not yet part of DCML.