dcml Data Center Markup Language

Data Center Markup Language Framework Specification

Draft
Version 0.11
May 5, 2004, 2004

Change History

Version	Date	Notes
version 0.1	November 9, 2003	Initial draft
version 0.2	November 14, 2003	Wording and example edits
version 0.3	November 17, 2003	Added copyright
version 0.4	March 31, 2004	Incorporated feedback from 3/2004 working group meeting.
version 0.7	April 14, 2004	Incorporated feedback from reviewers.
version 0.8	April 22, 2004	More feedback
version 0.9	April 23, 2004	Move processor implementation to appendix
version 0.10	April 28, 2004	Added language around blueprint to make it more experiemental.
version 0.11	May 5, 2004	Incorporated OWL and RDF related feedback from Michael K. Smith

Copyright

About This Document

Data Center Markup Language (DCML) is a data oriented approach to solve 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.

Required Reading

This document assumes background knowledge in various standard technologies and at times makes direct reference to these technologies. In order for this specification to be effective, the reader should have a good understanding of the technologies described by the following specifications:

- "Resource Description Framework (RDF): Concepts and Abstract Syntax", W3C Recommendation, Feb. 2004, http://www.w3.org/TR/rdf-concepts

- "OWL Web Ontology Language Guide", W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-guide

- “XML Encryption Syntax and Processing”, W3C Recommendation 10 December 2002, http://www.w3.org/TR/xmlenc-core

Optional Reading

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

- "OMG Unified Modeling Language Specification", An Adopted Formal Specification of the Object Management Group, Inc., Version 1.5, March 2003

Table of Contents

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 recipe that can be used by an automation system to construct and manage an environment. The third is the set of rules, policies, best practices and standards that should be used in managing the environment. DCML defines a standard XML-based format for describing each type of information.

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.

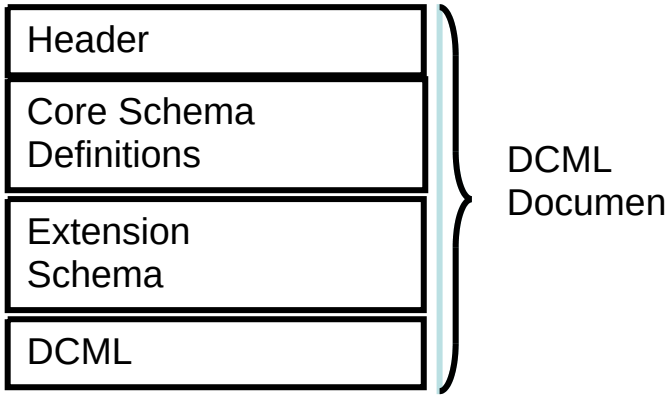
- Extension schema is the DCML mechanism to describe new types in the changing management environment. As DCML grows, the number of extension schemas also grows. Extension schemas consist of entities that capture domain-specific knowledge such as configuration parameters for a hardware load-balancer. 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:



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.

2.4 DCML Processing

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

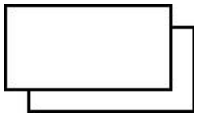
The DCML conceptual model consists of two layers, a core schema and extension schemas. The core schema described in this section sets the foundation for extension schemas. The core schema and extension schemas are described using the semantics of the DCML meta-model. The following sections describe the 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

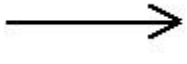
The DCML conceptual model is vocabulary of W3C Web Ontology Language [OWL], specifically OWL Full. OWL is a Resource Description Framework Schema [RDFS] vocabulary for describing properties and classes.

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.

A 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.

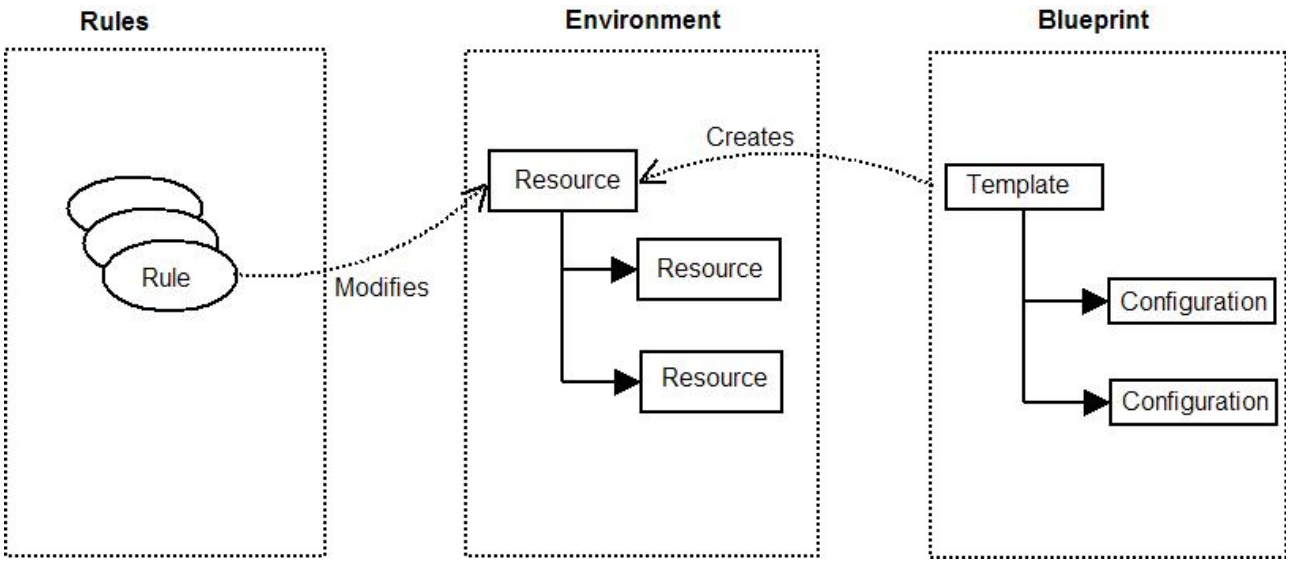
5.2 DCML Core Schema

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:

```
<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. 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

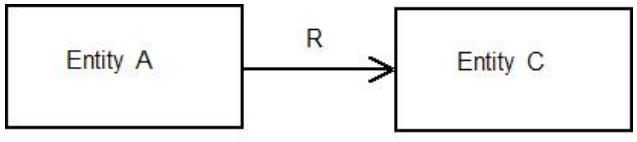
The NonDCMLEntity class represents a wrapper of non-RDF encoded information. The information wrapped by a NonDCMLEntity instance is opaque to DCML and DCML processors. Non-DCML entities range from large binary software packages to information encoded in non-DCML standards such as DMTF CIM objects and 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:



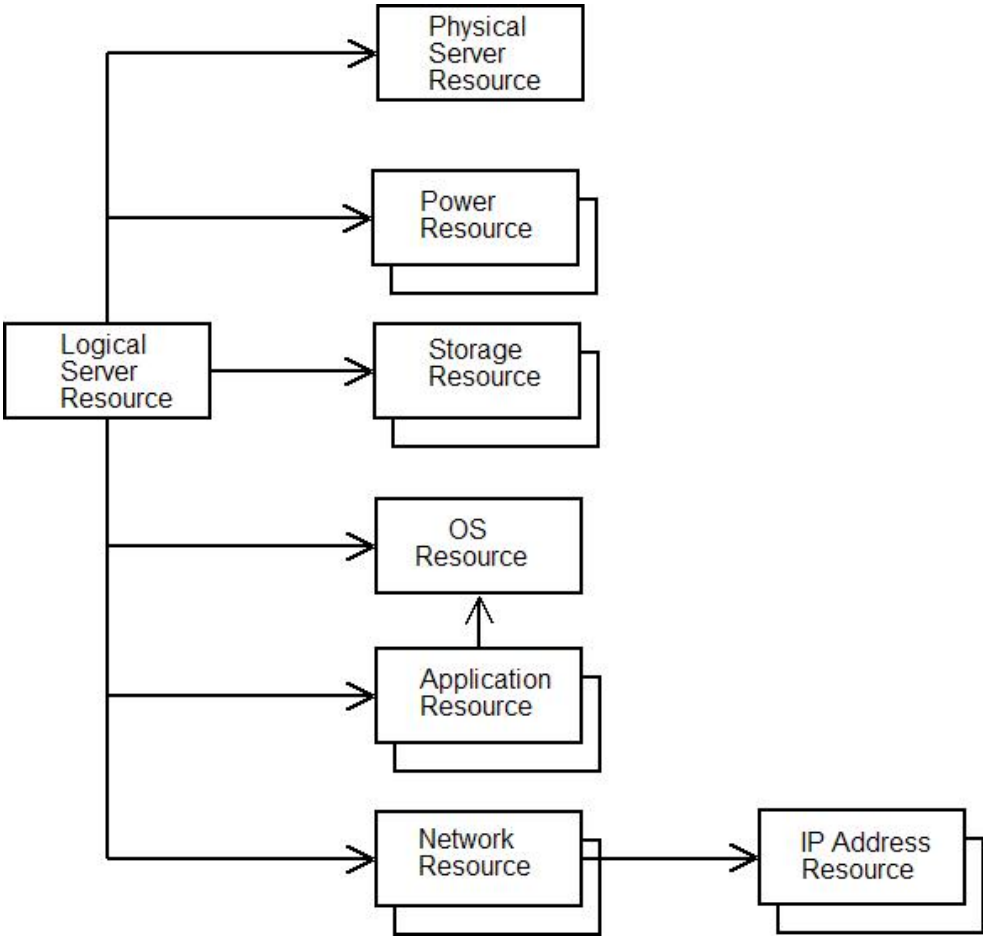
Here, entity-A contains a relationship-R to entity-C. This means the entity class A has an OWL object property that refers to a class C, similar to a class member in an OO language.

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.

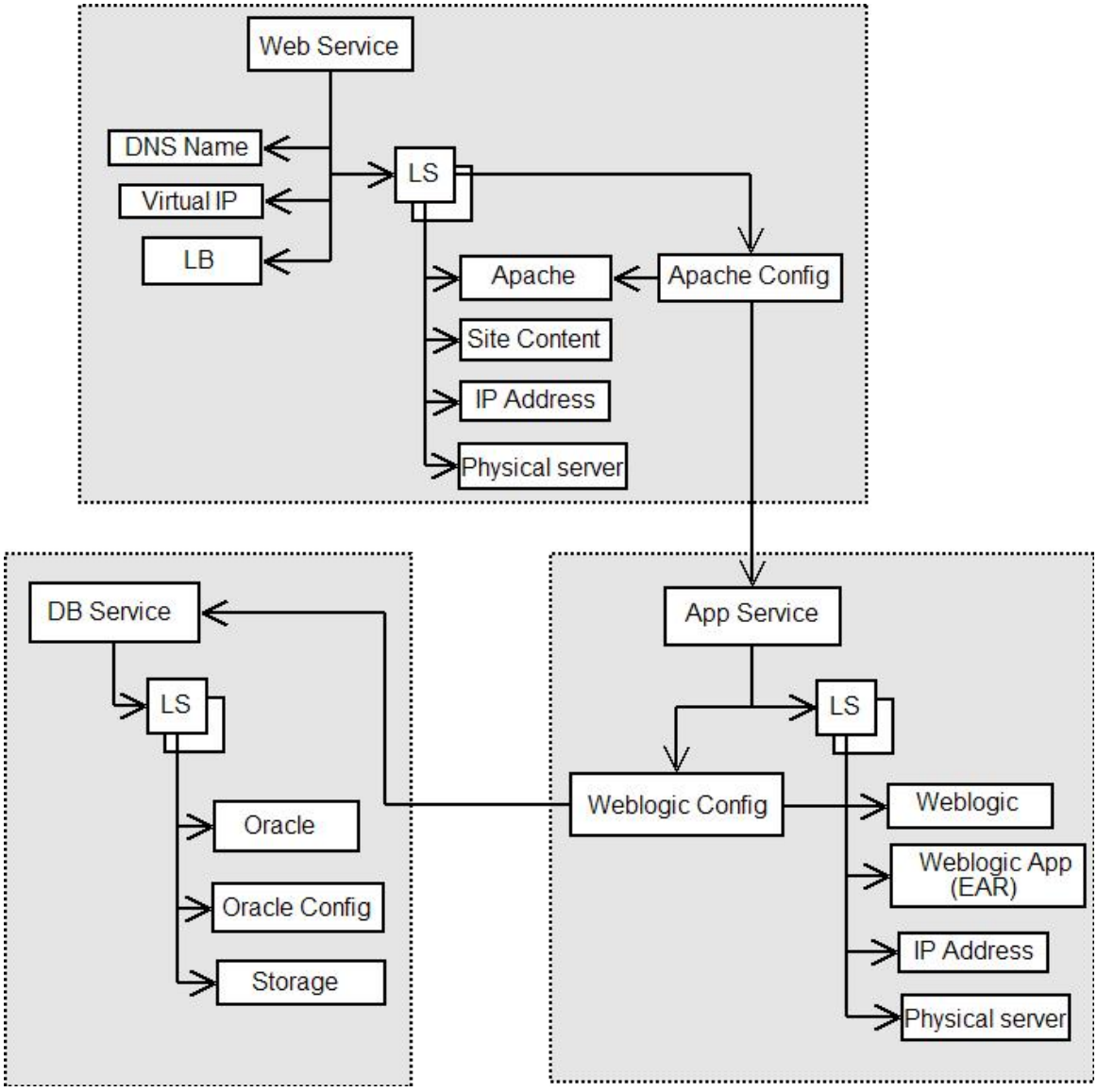
Using the resource and relationship constructs, extension schemas can create specific definitions. The following is a simple static diagram example that shows resource and relationship definitions for a logical server resource. The formal definition of the entities and relationships in the example below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

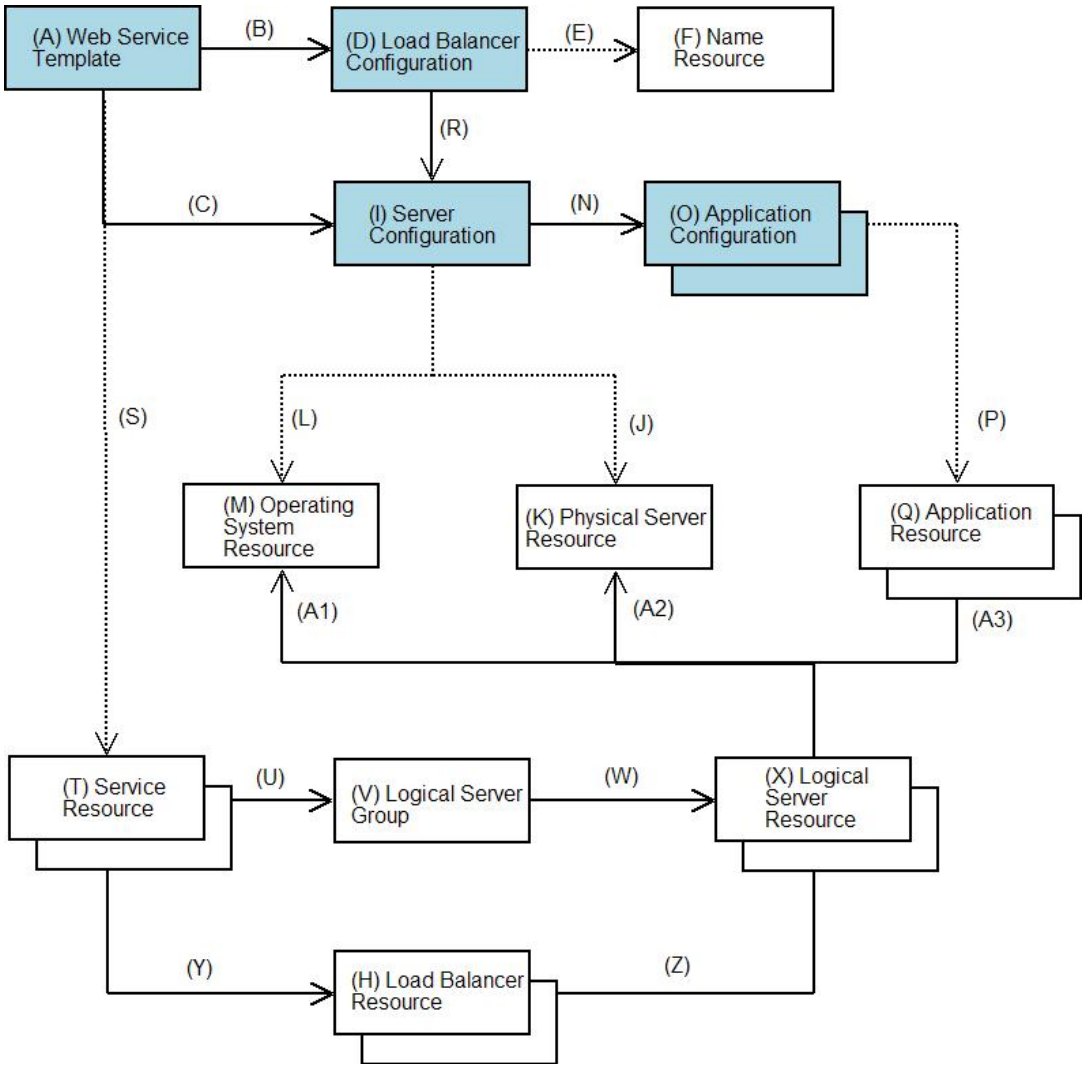
The blueprint consists of templates, relationships, and configurations that describe how applications and services should be instantiated in an environment. A template consists of attributes and relationships to configurations and is used to describe the configuration of services. (See diagram at end 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.

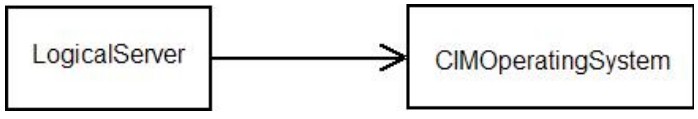
5.3 Core Rules Entities

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 definined 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.




```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!DOCTYPE rdf:RDF [
 3   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
 4   <!ENTITY owl "http://www.w3.org/2002/07/owl#">
 5   <!ENTITY dcml "http://www.dcml.org/ns/dcml/1/core#">
 6 ]>


 7 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 8          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 9         xmlns:owl="http://www.w3.org/2002/07/owl#"
10         xmlns:dcml="http://www.dcml.org/ns/dcml/1/core#"
11         xmlns="http://www.dcml.org/ex/dcml/1/exampleSchema#"
12          xml:base="http://www.dcml.org/ex/dcml/1/exampleSchema">


13   <dcml:Header rdf:about="">
14     <rdfs:comment>Example DCML extension schema</rdfs:comment>
15     <rdfs:label>Example</rdfs:label>
16     <dcml:author>DCML Framework Working Group</dcml:author>
17     <dcml:createDate>2004-04-22T01:08:30.711Z</dcml:createDate>
18   </dcml:Header>


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

```
22      <rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>
23    </owl:Class>


24    <owl:Class rdf:ID="LogicalServer">
25      <rdfs:comment>A DCML logical server.</rdfs:comment>
26      <rdfs:subClassOf rdf:resource="&dcml;Resource"/>


27      <rdfs:subClassOf>
28        <owl:Restriction>
29          <owl:onProperty rdf:resource="#os"/>
30          <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
31        </owl:Restriction>
32      </rdfs:subClassOf>
33    </owl:Class>


34    <owl:ObjectProperty rdf:ID="os">
35      <rdfs:comment>A logical server to OS relationship.</rdfs:comment>
36      <rdfs:domain rdf:resource="#LogicalServer"/>
37      <rdfs:range rdf:resource="#CIMOperatingSystemEntity"/>
38    </owl:ObjectProperty>
39
40  </rdf:RDF>
```

- 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.

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


 2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 3           xmlns:dcml="http://www.dcml.org/ns/dcml/1/core#"
 4           xmlns="http://www.dcml.org/ex/dcml/1/exampleSchema#">


 5    <dcml:Header rdf:about="">
 6      <rdfs:comment>Example DCML instance document.</rdfs:comment>
 7      <rdfs:label>Example</rdfs:label>
 8      <dcml:author>DCML Framework Working Group</dcml:author>
 9      <dcml:createDate>2004-04-22T01:08:30.711Z</dcml:createDate>
10    </dcml:Header>


11    <LogicalServer rdf:ID="myServer">
12      <dcml:name>MyServer</dcml:name>
13      <os rdf:resource="#myOS"/>
14    </LogicalServer>
15
16    <CIMOperatingSystem rdf:ID="myOS">
17      <dcml:xmlData>
18        <CIM CIMVERSION="2.0" DTDVERSION="2.0">
19          <INSTANCE CLASSNAME="OperatingSystem">
20            <PROPERTY NAME="MaxProcessesPerUser">
21              <VALUE>32</VALUE>
22            </PROPERTY>
23          </INSTANCE>
24        </CIM>
25      </dcml:xmlData>
26    </CIMOperatingSystem>


27  </rdf:RDF>
```

- Lines 5-10 describe the document header.
- Lines 11-14 describe a LogicalServer instance that contains an os property to a CIMOperatingSystem instance.
- Lines 16-26 describe a CIMOperatingSystem instance and embeds a CIM OperatingSystem instance encoded XML CIM.

7 Instance Naming

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.

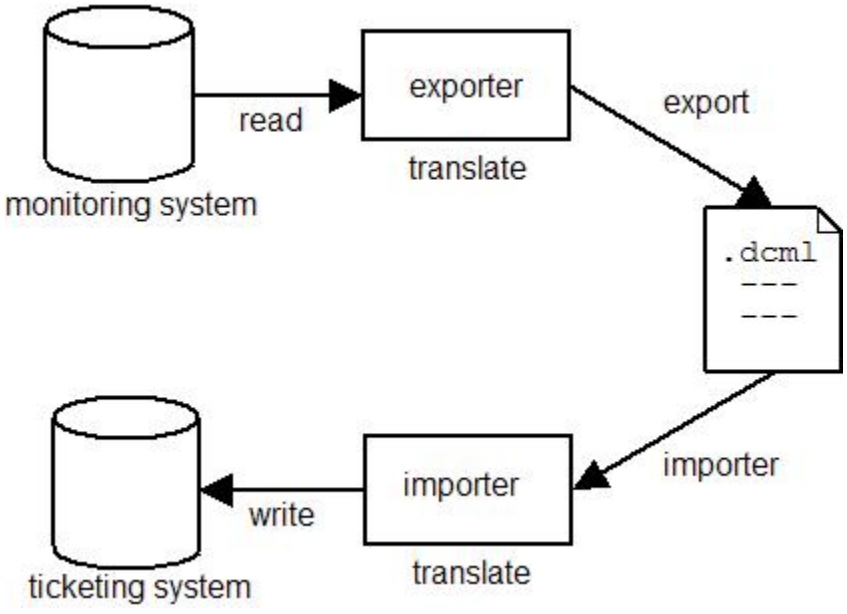
A management system exporting DCML is responsible for creating unique names (within its assigned namespace) that map to DCML entity instances it exports. This name must be consistent (i.e. a name must always match the same entity within the namespace) through the deployed life of the management system.

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 for all server records where the primary IP address is 10.2.1.2. 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

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!DOCTYPE rdf:RDF [
 3   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
 4   <!ENTITY owl "http://www.w3.org/2002/07/owl#">
 5   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 6 ]>


 7 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 8          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 9          xmlns:owl="http://www.w3.org/2002/07/owl#"
10          xmlns="http://www.dcml.org/ns/dcml/1/core#"
11          xml:base="http://www.dcml.org/ns/dcml/1/core">


12   <Header rdf:about="">
13     <rdfs:comment>DCML core schema definition.</rdfs:comment>
14     <rdfs:label>DCML</rdfs:label>
15     <author>DCML Framework Working Group</author>
16     <createDate>2004-04-22T01:08:30.711Z</createDate>
17   </Header>


18
19   <!-- ########################################################## -->
20   <!-- #                                                        # -->
21   <!-- #                Header Definition                       # -->
22   <!-- #                                                        # -->
23   <!-- ########################################################## -->


24   <owl:Class rdf:ID="Header">
25     <rdfs:comment>The Header class captures information about a DCML extension
schema and instance document.</rdfs:comment>
26     <rdfs:subClassOf rdf:resource="&owl;Ontology"/>
27   </owl:Class>


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


33   <owl:DataProperty rdf:ID="createDate">
34     <rdfs:comment>The creation date of the schema or instance
document.</rdfs:comment>
```

```
    <rdfs:domain rdf:resource="#Header"/>
    <rdfs:range rdf:resource="&xsd;dateTime"/>
  </owl:DataProperty>



  <!-- ########################################################## -->
  <!-- #                                                        # -->
  <!-- #                Entity Definition                       # -->
  <!-- #                                                        # -->
  <!-- ########################################################## -->

  <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>



  <!-- ########################################################## -->
  <!-- #                                                        # -->
  <!-- #            NonDCMLEntity Definition                    # -->
  <!-- #                                                        # -->
  <!-- ########################################################## -->

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


  <owl:DataProperty rdf:ID="encoding">
    <rdfs:comment>The encoding format of the wrapped information.  This can be
"ASN.1", "XML", etc.  This propery 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:DataProperty 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>
```

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


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




    82    <!-- ########################################################## -->
    83    <!-- #                                                        # -->
    84    <!-- #               CIMEntity Definition                     # -->
    85    <!-- #                                                        # -->
    86    <!-- ########################################################## -->
    87
    88    <owl:Class rdf:ID="CIMEntity">
    89      <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>
    90      <encoding>XML</encoding>
    91    </owl:Class>




    92    <!-- ########################################################## -->
    93    <!-- #                                                        # -->
    94    <!-- #              Environment Definiton                     # -->
    95    <!-- #                                                        # -->
    96    <!-- ########################################################## -->


    97    <owl:Class rdf:ID="Environment">
    98      <rdfs:comment>Top level environment classification.</rdfs:comment>
    99    </owl:Class>


   100
   101    <!-- ########################################################## -->
   102    <!-- #                                                        # -->
   103    <!-- #                 Resource Definiton                     # -->
   104    <!-- #                                                        # -->
   105    <!-- ########################################################## -->
```

```
106    <owl:Class rdf:ID="Resource">
107      <rdfs:comment>A managable entity.</rdfs:comment>
108      <rdfs:subClassOf rdf:resource="#Environment"/>
109      <rdfs:subClassOf rdf:resource="#Entity"/>
110      <rdfs:subClassOf>
111        <owl:Restriction>
112          <owl:onProperty rdf:resource="#name"/>
113          <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
114        </owl:Restriction>
115      </rdfs:subClassOf>
116    </owl:Class>


117
118    <!-- ////////////////////////////////////////////////////// -->
119    <!-- //                                                    // -->
120    <!-- //               Blueprint Definition                 // -->
121    <!-- //                                                    // -->
122    <!-- ////////////////////////////////////////////////////// -->


123    <owl:Class rdf:ID="Blueprint">
124      <rdfs:comment>Top level blueprint classification.</rdfs:comment>
125    </owl:Class>
126


127    <!-- ////////////////////////////////////////////////////// -->
128    <!-- //                                                    // -->
129    <!-- //               Template Definition                  // -->
130    <!-- //                                                    // -->
131    <!-- ////////////////////////////////////////////////////// -->


132    <owl:Class rdf:ID="Template">
133      <rdfs:comment>Configuration template.</rdfs:comment>
134
135      <rdfs:subClassOf rdf:resource="#Blueprint"/>
136      <rdfs:subClassOf rdf:resource="#Entity"/>
137    </owl:Class>




138    <!-- ////////////////////////////////////////////////////// -->
139    <!-- //                                                    // -->
140    <!-- //             Configuration Definition               // -->
141    <!-- //                                                    // -->
142    <!-- ////////////////////////////////////////////////////// -->


143    <owl:Class rdf:ID="Configuration">
144      <rdfs:comment>Configuration entity.</rdfs:comment>
145      <rdfs:subClassOf rdf:resource="#Blueprint"/>
```

```
146      <rdfs:subClassOf rdf:resource="#Entity"/>
147    </owl:Class>


148
149    <!-- ////////////////////////////////////////////////////// -->
150    <!-- //                                                    // -->
151    <!-- //                 Rules Definition                   // -->
152    <!-- //                                                    // -->
153    <!-- ////////////////////////////////////////////////////// -->


154    <owl:Class rdf:ID="Rules">
155      <rdfs:comment>Top level rules classification.</rdfs:comment>
156    </owl:Class>



157  </rdf:RDF>
```

11 References

[OMD] “Ontology Definition MetaModel – Initial Submission”, DTSC, OMG Inc., 18 August 2003, (pg 13).

[XML] "Extensible Markup Language (XML) 1.0", W3C Recommendation, Feb. 1998, http://www.w3.org/TR/REC-xml

[XMLS2] “XML Schema Part 2: Datatypes”, W3C Recommendation, 02 May 2001, http://www.w3.org/TR/xmlschema-2

[RDF] "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, Feb. 1999, http://www.w3.org/TR/REC-rdf-syntax

[RDFS] "Resource Description Framework (RDF) Schema Specification", W3C Recommendation, March 2000, http://www.w3.org/TR/rdf-schema

[UML] "OMG Unified Modeling Language Specification", An Adopted Formal Specification of the Object Management Group, Inc., Version 1.5, March 2003

[OWL] "OWL Web Ontology Language Guide", W3C Candidate Recommendation 18 August 2003, http://www.w3.org/TR/owl-guide

[CIM] “Common Information Model Specification,” Distributed Management Task Force, Inc., version 2.2, 14 June 1999.

[RDQL] “RDQL – A Query Language for RDF,” W3C Member Submission 9 January 2004, http://www.w3.org/Submission/RDQL

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.