



Copyright © 2003-2006 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

DEN LDAP Mapping

The DMTF Technical Committee

DEN-support@dmf.org

This technical note discusses some of the issues that arise when implementing an LDAP mapping of the CIM Schema. It reflects work in the DEN (Directory Enabled Networks) Working Group of the Distributed Management Task Force (DMTF).

Introduction

CIM is defined to be technology and implementation-neutral, but to be useful, must be mapped into existing repositories and accessed via established transports and interfaces. One of the possibilities is to map CIM to a directory that supports the LDAP protocol and an X.500 model. However, there are several items to consider when doing this. This technical note discusses these issues.

Appropriate Use of a Directory

Since LDAP directory servers are tuned for very fast reads and searches, they are usually poor performers when it comes to writes (edits), adds (new objects), and deletes. The CIM object lifecycle includes creation (adds), modification (writes), and deletion. Therefore, most aspects of the lifecycle are burdensome for the LDAP server, and LDAP servers generally should not be used for CIM object manager repositories. However, there are many CIM applications that spend a lot of time reading and searching for CIM objects and associations. When the ratio of reads and searches to CIM lifecycle operations is reasonably high, LDAP directories make excellent repositories, especially for CIM namespaces that span many hosts and devices. So what types of CIM classes and associations make good candidates for CIM to LDAP mapping? As a rule, those whose lifetimes are long relative to their number and those whose properties are not highly volatile. As it turns out, many areas of the CIM Core and Common Models fall into this category.

Users, computers, services, operating systems and software, service access points, policies, configurations and settings, physical elements, and others are examples of CIM managed elements that once created, have long lifetimes, and whose properties are pseudo-static. In the case of users and computers, the number of distinct objects and associations to those objects is generally quite high in an enterprise or service provider environment. In the case of configurations and policies, their modeled properties may change more rapidly than those of users or computer systems, but in general, that is balanced by there being relatively fewer of them. On the other hand, CIM_StatisticalData is an example of a class whose properties are highly volatile, and in general not a good candidate for LDAP mappings.

DMTF STANDARDS AND TERMINOLOGY

- 1 Introduction
- 1 Appropriate Use of a Directory
- 2 Naming Considerations
- 2 Definition of the CIM "Object" Tree
- 2 Lack of Associations in the Directory
- 3 Versioning
- 3 Closing Remarks

Naming Considerations

Although the rules for uniqueness of CIM objects are strict (basically, their key properties must have correct values), the rules for the Relative Distinguished Names (RDNs) of mapped CIM objects are flexible and vary by implementation. When choosing the RDN, one must consider whether the directory implementation restricts the choice of RDN attributes to "well-known" values (such as O, OU, etc.). DEN applications desiring interoperability should consider the DN of an entry to be a set of opaque attributes, without any particular semantic assigned to them. Specific semantics and searchable fields should be restricted to named class attributes, and not implied in a DN. An attribute lookup is preferred over the possibility that a particular attribute might be used as the RDN.

If it is desired to use the specified CIM "key" properties for naming, there are some items to consider. In combination with a deep namespace design, the use of CIM keys for directory RDNs can result in very long Distinguished Names (DNs). Depending on the directory server implementation, long DN's can have a negative impact on search speed and indexing. In general, short RDNs that are reasonably unique (at least unique with respect to their siblings in the namespace) are preferable.

Definition of the CIM "Object" Tree

All CIM classes are mapped as abstract classes, and subclassed as defined in the CIM Schema. (There is a minor exception to this rule in the case of ManagedElement, which subclasses from the directory's "top".) For each concrete class, there are two subclasses – the first is an auxiliary class and the second is a structural one. To understand this, consider the following example - a CIM_FooService class is defined that subclasses from CIM_Service. In the DEN mapping, it would be represented as follows (where inheritance is indicated by indenting):

```
dn: dlmName=foobar, ou=corporate,
dc=acmenetworks,dc=net
dlmName: foobar
objectClass: top
objectClass: dlmlManagedElement (A=Abstract)
objectClass: dlmlManagedSystemElement (A)
objectClass: dlmlLogicalElement (A)
objectClass: dlmlEnabledLogicalElement (A)
objectClass: dlmlService (A)
objectClass: dlmlFooService (A)
objectClass: dlmlFooServiceAuxClass
objectClass: dlmlFooServiceInstance
```

(Note that the prefix, "dlm" is an acronym for "DEN LDAP Mapping".)

The abstract classes are included in the hierarchy for two reasons:

- (1) they provide a natural hierarchy for modeling CIM inheritance
- (2) they provide a convenient object class for searches

By searching for the abstract class, an application can find both instance and auxiliary entries and therefore interoperate with other directory applications. The structural classes are used in cases where an entry does not pre-exist in the directory, and a new structural object is needed for instances to be defined. The auxiliary class form is used to overlay a new DEN semantic over an already existing directory entry – and add the CIM attributes to that entry.

Lack of Associations in the Directory

In many cases, associations are relatively static. This is not the main concern regarding their implementation in a directory. Because association classes define relationships between objects, directory namespace issues start to come into play. There are a few different ways that CIM associations can be mapped into LDAP directories, and namespace issues are a factor in all of these ways.

First, let us consider the alternatives for mapping associations into a directory. Associations can be handled by three approaches. The first uses auxiliary classes to attach a reference to the "other" endpoint of the association. The second approach uses a structural entry for the association and "helper" references to point to the association entry, from each endpoint of the association. Finally, some associations are not explicitly mapped, but are captured through DIT containment. Examples of the latter are the weak associations from the Physical Model, that logically fit into the directory concept of DIT containment.

Returning to the namespace issue, every environment and application will have different requirements for their directory namespaces. Therefore, each implementation must carefully consider the implications of which CIM objects and associations will be mapped, where they will be placed in the namespace, and how the namespace will be partitioned. For a non-partitioned namespace, some of these problems may go

away. However, in the general case, directory services will span physical servers, and in the absence of multi-master replication capability, the nature and namespace location of objects that are replicated must be carefully considered.

For example, consider what will happen if an association relates objects from partitioned portions of a directory namespace. Following the association may cause the directory server to generate LDAP referrals to the client, forcing the client to connect and (usually) authenticate to the server partition holding the associated object. Chasing referrals is a relatively inefficient operation for most application clients. Given CIM's heavy usage of associations, the number of referrals required for common association traversals must be minimized. This can be accomplished by a namespace design that is sensitive to the association considerations of a CIM mapping.

For example, a HostedService association and a ServiceToServiceDependency are two examples where it behooves the namespace designer to place the associated objects within portions of the namespace that will not be partitioned. ElementSetting may be another example since finding a CIM_Setting for a given element may be done frequently. However, ServiceAccessBySAP may be less of a problem, unless the connection span for accessing the service by a client is very short, requiring many association traversals over short time intervals.

Versioning

Class names in the directory include a version number. This is necessary because the mapping of OID to object class name to object class description is 1:1:1. That is, once the object class definition changes, it is required to assign both a new name and OID. And, a change to the definition results in a new branch of the object class tree - as all children classes must also have their version information incremented. This makes finding a specific class very cumbersome.

To support multiple versions, an auxiliary "indicator" class is defined in the DEN mappings. This takes the form, "dln<class name>IndicatorAuxClass". This auxiliary class can be attached to any object class to permit locating a particular class, regardless of version number. For example, consider the CIM_FooService class from the

previous example. This class may be modified to add a property. So, two possible class names may be found in the directory - dlm1FooService or dlm2FooService. But, either version could be found by searching on dlmFooIndicatorAuxClass!

Closing Remarks

The DEN initiative has evolved to describe how to use CIM and a directory to locate management information, and to access management data. DEN specifies the LDAP mappings for the DMTF's CIM releases. This technical note provides an overview of the issues that were considered in defining that mapping.

More detailed technical information on DEN can be found in the standards area (www.dmtf.org/standards/standard_den.php) of the DMTF web site.