CIM Concepts White Paper

CIM Versions 2.4+

Document Version 0.9 June 18, 2003

Abstract

The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment. A fundamental taxonomy of objects is defined — both with respect to classification and association, and with respect to a basic set of classes intended to establish a common framework.

The basic concepts and constructs of the management model are described in this paper.

Notice

DSP110 ***Status: Work in progress***

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

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents for uses consistent with this purpose, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

Table of Contents

Abstract 1
Notice 2
Table of Contents 3
1 Introduction 4
1.1 Overview 4
1.2 Background Reference Material 5
1.3 Terminology 6
2 CIM Concepts 7
2.1 Background and Assumptions 7
2.2 MOF representation of the Model 7
2.3 UML Representation of the Model 8
2.3.1 Boxes 8
2.3.2 Lines 10
2.4 Subclassing in CIM 12
2.4.1 Extending CIM's Enumerations 13
2.5 Naming in CIM 14
3 Do's and Don'ts in CIM Design 16
Appendix A – Change History 19
Appendix B – References 20

1 Introduction

1.1 Overview

CIM is an information model, a conceptual view of the managed environment, that attempts to unify and extend the existing instrumentation and management standards (SNMP, DMI, CMIP, etc.) using object-oriented constructs and design. Note that the word, "unify," is used in the preceding sentence, not the word, "replace." CIM does not require any particular instrumentation or repository format. It is only an information model – unifying the data, using an object-oriented format, made available from any number of sources.

The value of CIM stems from its object orientation. Object design provides support for the following capabilities, that other "flat" data formats do not allow:

- Abstraction and classification – To reduce the complexity of the problem domain, high level and fundamental concepts (the "objects" of the management domain) are defined. These objects are then grouped into types ("classes") by identifying common characteristics and features (properties), relationships (associations) and behavior (methods).
- Object inheritance – Subclassing from the high level and fundamental objects, additional detail can be provided. A subclass "inherits" all the information (properties, methods and associations) defined for its higher level objects. Subclasses are created to put the right level of detail and complexity at the right level in the model. This can be visualized as a triangle – where the top of the triangle is a "fundamental" object, and more detail and more classes are defined as you move closer to the base.
- Ability to depict dependencies, component and connection associations – Relationships between objects are extremely powerful concepts. Before CIM, management standards captured relationships in multi-dimensional arrays or cross-referenced data tables. The object paradigm offers a more elegant approach in that relationships and associations are directly modeled. In addition, the way that relationships are named and defined describe the semantics of the object associations. Further semantics and information can be provided in properties (specifying common characteristics and features) of the associations.
- Standard, inheritable methods – The ability to define standard object behavior (methods) is another form of abstraction. Bundling standard methods with an object's data is encapsulation. Imagine the flexibility and possibilities of a standard able to invoke a "Reset" method against a hung device, or a "Reboot" method against a hung computer system regardless of the hardware, operating system or device.

In addition, CIM's goal is to model all the various aspects of the managed environment, not just a single problem space. To this end, various "Common Models" have been created to address System, Device, Network, User, Application, and other problem spaces. All of the problem domains are interrelated via associations and subclassing.

They all derive from the same fundamental objects and concepts - as defined in the Core Model.

In order to understand why a unifying model is important, consider the following scenario. A payroll application generates an alert, due to multiple timeouts, when communicating with a database server. The alert is forwarded to a management application and network administrator who is monitoring alerts and events within a particular network domain, using the management application. The administrator performs a network route analysis to display all the "managed objects" and their percent utilization, in the path from the client application to the database server.

The administrator discovers that one card in a network device is at 98% utilization. In investigating further (by traversing CIM associations and exploring additional subclasses and data objects), the network administrator finds that one port of that card has a very high traffic rate, and its traffic is not related to the payroll application. Again following associations, the administrator can determine the "owner" of the System currently attached to the network port. If the owner cannot be reached, the administrator can use a standard method to "disable" the port, thereby returning the hub to normal bandwidth levels and allowing the payroll application to proceed.

In the preceding example, Network, Device, System and User Models were all exercised. It was required that the models' interrelationships and interactions be consistently defined, as well as the correct basic objects.

1.2 Background Reference Material

CIM Core and Common Models - Versions 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, and 2.7 - Downloadable from http://www.dmtf.org/standards/standard_cim.php

Common Information Model (CIM) Specification, V2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

DMTF Specifications - Approved Errata - Downloadable from http://www.dmtf.org/standards/standard_cim.php

Unified Modeling Language (UML) from the Open Management Group (OMG) - Downloadable from http://www.omg.org/uml/

1.3 Terminology

Term	Definition
Data Model	A concrete representation of an information model in terms appropriate to a specific data store and access technology
Information Model	An abstraction and representation of the entities in a managed environment - their properties, operations, and relationships. It is independent of any specific repository, application, protocol, or platform.
MOF	Managed Object Format, the "language" defining the CIM object classes, properties, methods and associations
Object-Oriented Classification	Grouping or typing of objects (into "classes") by identifying common characteristics and features (properties), relationships (associations) and ability to affect state (methods)
Object-Oriented Modeling	Object-oriented modeling is a representation of a domain (describing its basic concepts, state changes and relationships), using traditional OO/classification theory
Schema	A set of data models that describe a set of objects to be managed
UML	Unified Modeling Language. CIM is defined by MOF and visualized using UML.

2 CIM Concepts

2.1 Background and Assumptions

It is assumed that the reader is familiar with the concepts and terminology covered in the CIM Specification.

2.2 MOF representation of the Model

The following figure illustrates MOF (Managed Object Format), the syntax of the CIM Schemas.


```
    [Abstract, Description (
     "An abstraction or emulation of a hardware entity, that may "
     "or may not be Realized in physical hardware. ... ") ]
class CIM_LogicalDevice : CIM_LogicalElement
{
. . .
        [Key, MaxLen (64), Description (
         "An address or other identifying information to uniquely "
         "name the LogicalDevice.") ]
   string DeviceID;
        [Description (
         "Boolean indicating that the Device can be power "
         "managed. ...") ]
   boolean PowerManagementSupported;
        [Description (
         "Requests that the LogicalDevice be enabled (\"Enabled\" "
         "input parameter = TRUE) or disabled (= FALSE). ...)" ]
   uint32 EnableDevice([IN] boolean Enabled);
. . .
};
```

Inheritance is indicated by placing a colon and the superclass name after the class name. In the figure above, the CIM_LogicalDevice class is defined. It subclasses from CIM_LogicalElement.

Qualifiers are mentioned, and several are used in the MOF example above. Qualifiers are meta-data, providing information about a class, property, method or reference in the CIM Schema. The intent and purpose of most of them is quite obvious (for example, *Description ("xxx"), MaxLen (256)* or *Units ("Seconds")*). All of the CIM qualifiers are listed and explained in Section 2.5 of the *Common Information Model (CIM)*

Specification, V2.2 (June 14, 1999) [1], with additional explanations in the CIM Core White Paper, DSP0111 [2].

2.3 UML Representation of the Model

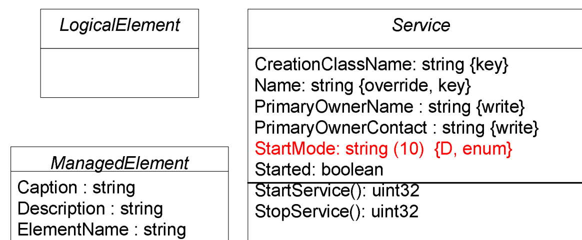
The visual representation of the model is based on UML (Unified Modeling Language) [3]. There are distinct symbols for all of the major constructs in the schema.

2.3.1 Boxes

A MOF class definition is represented in UML by a rectangle. A class definition consists of three areas. The class name, the class properties, and the class methods.

- Class Name

The class name is in the uppermost segment. An italicized class name indicates that the class is defined to be abstract. All three classes below are defined to be abstract.



- Class Properties

The segment below the class name contains the name contains the properties of the class. The property declarations have the form of:

<Property Name> **:** <Property Type> **{**<additional information>**}**

In the example:

- LogicalElement has no properties.

- ManagedElement has three string properties named Caption, Description, and ElementName

- Service has five string properties named CreationClassName, Name, PrimaryOwnerName, PrimaryContact and StartMode. In addition, Service has 1 boolean property named Started.

Only the properties in the Service class specify additional information. The additional information presented between the **{ }** is related to presents of special CIM qualifiers. These special qualifiers include:

- Key – Indicating that the key qualifier is specified, defining a property used for naming and identity.
- Enum – Indicating the a ValueMap and likely Values qualifier is specified
- Units – Indicates that Unit qualifier is specified.
- Write – Indicates that the Write qualifier is specified.
- Req'd – Indicates that the Required qualifier is specified.
- Override – Indicates that the Override qualifier is specified.
- D – Indicates that the Deprecated qualifier is specified.

Red colored text is used to highlight that a class or property has been marked for Deprecation.

- Class Methods

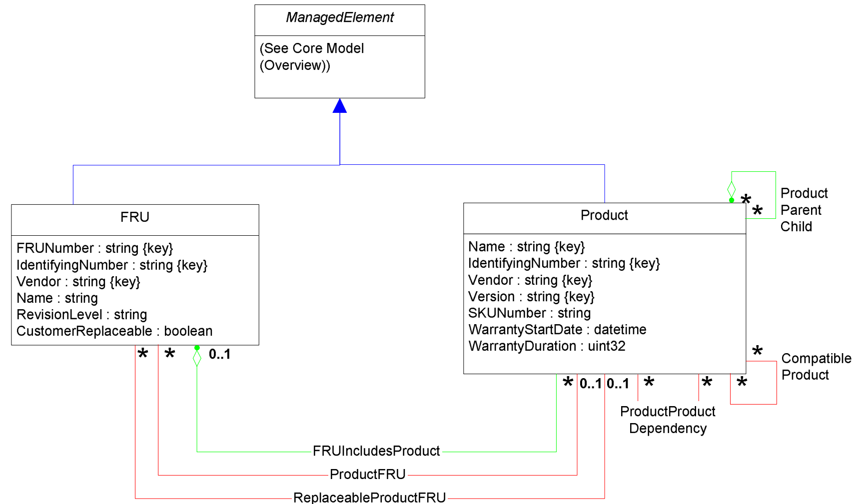
If present, a third region indicates the presence of **methods** (operations)**.** The method declaration has the following form:

<Method Name> **(**<Method Parameters>**) :** <return type>

In the figure above, only the Service class has methods. It defines two methods. StartService and StopService. Neither method has parameters and both return a uint32 result code.

2.3.2 Lines

The lines in a UML diagram are separated into two major categories: Inheritance and Associations.



- Inheritance

Inheritance relationships (blue lines with arrows) are also known as “is-a” relationships. Inheritance relationships are not specifically labeled or named.

In the example, FRU and Product inherit from ManagedElement.

- Associations

There are two distinct types of associations.

- Aggregation (a green line with a diamond shape at the “aggregating” end)
  An aggregation or component relationship is also known as a “has-a” or “member-of” relationship.

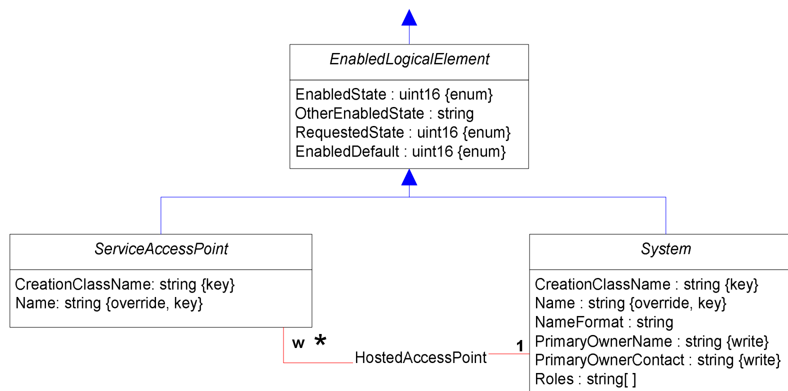
  A composition relationship (a green line with a diamond shape and filled in circle at the “aggregating” end) further refines an aggregation to a “whole” / “part” relationship.

- Non-Aggregating associations (red lines)
  Typically in CIM, these relationships are a subclass of CIM_Dependency. However, there are other top-level, non-aggregation relationships defined.

All associations are defined as classes. The cardinalities of the references on both sides of an association are indicated by numeric values or an asterisk (*). The following cardinalities are typically used in the CIM Schema:

0..1	Indicates an optional single-valued reference
1	Indicates a required, single-valued reference
1..n or 1..*	Indicates either a single or multi-valued reference, that is required
, 0..n or 0..	Indicates an optional, single or multi-valued reference

In the table, above, a "required" reference means that the object and the association MUST be instantiated when the other referenced class is instantiated. For example, in the figure below, the System class has a cardinality of 1 in the Hosted Access Point association. Therefore, when a Service Access Point is instantiated, a scoping System and the Hosted Access Point association must also be instantiated.



The symbol "w" is also used to label an association and can be seen in this figure. It indicates that the referenced endpoint or class is "weak" with respect to the other class participating in the association. This means that the referenced class is scoped or named relative to the other class, and the identifying keys of the "other" class are propagated to the "weak" class. (Note that key properties are designated using the "Key" qualifier.) The concept of "weak" is not standard UML convention, but an added symbol in the CIM diagrams, as is the use of color to denote the different types of relationships.

Taking the Hosted Access Point association as an example, one can conclude:

- It is likely to be a Dependency relationship (which it indeed is), where the System object is the Antecedent or independent reference
- There is one System object that must be referenced by a Service Access Point (the cardinality on System, “1”, indicates a required, single-valued reference)
- There can be many Service Access Points associated with a System
- Service Access Points are scoped or named relative to the System to which they are “weak”

Occasionally, there is confusion related to the cardinalities of the model associations. Cardinalities define the number of INSTANCES OF THE ASSOCIATION for single INSTANCES of the referenced classes. They do not define the total number of instances for the class as a whole.

For example, a particular instance of a Protocol Endpoint (a kind of Service Access Point) is associated with a particular instance of a Computer System, on which the interface is configured (and, by which it is scoped). However, the CIM_ProtocolEndpoint CLASS has many instances of many interfaces, each associated with their own Computer Systems. Does this then mean that the association, Hosted Access Point, should relate many endpoints to many Computer Systems? No – it defines the relationship between an instance of Computer System and an instance of Protocol Endpoint, not for the Protocol Endpoint class as a whole.

2.4 Subclassing in CIM

One of the strengths of the CIM Schema is its use of object-oriented design techniques. Inheritance (via subclassing) is one of these techniques. It is a powerful concept, but often questions arise regarding the appropriateness of subclassing - especially with respect to associations.

Listed below are various rationale for creating subclasses. Some of these are specific to associations since they address references and cardinalities.

The most common reason to subclass is to specialize semantics, moving from more general to more specific semantics. For example, a Logical Device is a subclass and specialization of Managed System Element. As another example, the association CIM_ServiceServiceDependency subclasses from CIM_Dependency, describing the specific dependency of one Service on another.

As part of specialization, new properties, methods or associations are defined for the subclass. The semantics conveyed by the new constructs are not appropriate for the entire population of the superclass, hence the subclass must be created.

When specializing associations (i.e., relationships), you often constrain their references or restrict their cardinalities. If semantics are specialized, it is likely that the references of the association are also specialized. For example, CIM_Dependency relates Managed Elements, but CIM_ServiceServiceDependency only relates instances of CIM_Service. On the cardinalities side of the argument, a superclass may define a many-to-many

relationship, but a subclass could restrict one of the references by modifying its cardinality to "0..1".

Sometimes, subclassing is done to indicate where a specific concept is located in the model, or what is particularly important and/or frequently instantiated in the model. Related to this, a subclass may be defined to easily query for its instances. This is the reasoning behind the many specializations of the Based On association in the Device Model. For example, the CIM_LogicalDiskBasedOnPartition association describes how a "C:" drive is built on top of a disk partition. It is a specific subclass of the Based On relationship between general Storage Extents.

Lastly, subclasses may be defined to specifically align with and "map" an existing standard. For example, the definition of Protected Space Extents and their relationship to Physical Extents is described in the Device Model as a direct mapping of the SCC (SCSI Controller Command) Specification.

2.4.1 Extending CIM's Enumerations

The addition of values to an enumeration (a *Values* or *ValueMap* array) is not permitted in a subclass. A subclass can restrict the permissible values, but not extend them. The reason for this limitation is that an administrator or application querying for the value of the enumerated property, may be querying at the superclass level. At this level, a fixed set of values is defined. To return different values would not allow for consistent and predictable responses and processing.

Let's go through an example. The Managed System Element class has an OperationalStatus property. Its permissible values are 0 through 17 ("Unknown", "Other", "OK", "Error" and "Stressed" being the first four). Now a printer, which inherits from Managed System Element, might have additional statuses such as "Jammed", "Low Toner", etc. These values certainly are not applicable to all Managed System Elements, nor would an application working at the very high level of Managed System Element be prepared to deal with such extended statuses as "Low Toner", "Dial Tone Lost" or "Incoming Call". For this reason, extensions to enumerations are not allowed.

But, all is not lost. There are three possible mechanisms to supplement the values in an enumeration:

- Most *Values* and *ValueMap* arrays include a value of "Other". Typically, an "Other Description" string property is also defined, with a *ModelCorrespondence* to the property with the *Values*/*ValueMap* arrays. To extend an enumeration, "Other" can be used, and any additional meaning conveyed in the "Other Description" string.

- Additional properties, supplementing an enumerated property, may be defined in subclasses. The printer example above is actually taken from the CIM Device Model. In the CIM_Printer class, both Printer Status and Detected Error State properties are defined (again as enumerations) - supplementing the information in Managed System Element's OperationalStatus property. Detected Error State's

Values array includes "Low Paper", "No Paper", "Low Toner", "Jammed" and "Output Bin Full". Printer Status' *Values* array includes "Idle", "Printing", "Warmup" and "Offline".

- A developer may contact the DMTF Technical Committee (technical@dmtf.org) to recommend the addition of new permissible values for an enumeration. This would be handled via a Change Request. If approved, the new values would appear in the next release of the CIM Schema.

2.5 Naming in CIM

In any instrumentation and management scheme, it is necessary to uniquely identify instances of classes. Various means of identification exist such as:

- Simple numeric counter - for example, an incrementing counter on messages sent between two entities
- GUIDs (globally unique identifiers)
- One or more parameters, which (when taken together) provide a unique identifier - for example, key properties in a database table

The latter is the approach chosen by the CIM designers.

The CIM Schema is a "keyed" object model, as opposed to an object identity model (for example, a GUID-oriented model). All class instances are uniquely named and referenced by the class' keys. Associations are uniquely identified by their keys, which have always included their reference properties. References consist of a class' keys and an instance's values for these keys. All concrete classes (those that are instantiated) must define or inherit a key structure. If inherited, the key structure cannot be changed.

As noted earlier, CIM specifies the key properties of a class using the *Key* qualifier. In native CIM implementations, it is required that the combination of properties with the *Key* qualifier is unique across all instances of a class, within a namespace. This combination of *Key* properties is the identification scheme for a native CIM class.

Since CIM is an information model, it may be implemented using various protocols and/or repositories. Different implementations of the CIM Schema may require different identification schemes and additional properties. For example, GUIDs may be used as one of the values in a class' key structure. On the other hand, a directory implementation of CIM (as specified by the DEN initiative) uses Distinguished Name as its native identification scheme. There, a combination of CIM *Key* properties with their individual values could be an RDN (Relative Distinguished Name). If this is done, maintaining the CIM key values is critical when communication with a native CIM implementation is required.

A question often arises regarding guidelines for class identification/naming in CIM - when should a modeler use regular keys and when should a class also have *Propagated*/*Weak* keys? The design is decided for almost all the CIM classes (and

therefore by inheritance to their subclasses). Today, most classes are weak to another, and therefore use propagated keys. For some classes, however - for example, subclasses of CIM_Setting - the key structure is left up to the modeler.

It must be noted that starting in CIM V2.7, a simplified naming convention is adopted. Previously, most classes' keys were multi-property, compound values. These were difficult to manipulate and usually carried redundant information (such as the class name). Starting in CIM V2.7, new classes (whose key structures are not dictated by inheritance) are identified using the single property, InstanceID. The definition of InstanceID is as follows:

> Within the scope of the instantiating Namespace, InstanceID
> opaquely and uniquely identifies an instance of this class.
> In order to ensure uniqueness within the NameSpace, the
> value of InstanceID SHOULD be constructed using the
> following 'preferred' algorithm:
> <OrgID>:<LocalID>
> Where <OrgID> and <LocalID> are separated by a colon ':',
> and where <OrgID> MUST include a copyrighted, trademarked
> or otherwise unique name that is owned by the business entity
> creating/defining the InstanceID, or is a registered ID that is
> assigned to the business entity by a recognized global
> authority (This is similar to the <Schema Name>_<Class Name>
> structure of Schema class names.) In addition, to ensure
> uniqueness <OrgID> MUST NOT contain a colon (':'). When using
> this algorithm, the first colon to appear in InstanceID
> MUST appear between <OrgID> and <LocalID>.
>
> <LocalID> is chosen by the business entity and SHOULD not be
> re-used to identify different underlying (real-world) elements.
> If the above 'preferred' algorithm is not used, the defining
> entity MUST assure that the resultant InstanceID is not
> re-used across any InstanceIDs produced by this or other
> providers for this instance's NameSpace.
>
> For DMTF defined instances, the 'preferred' algorithm MUST be
> used with the <OrgID> set to 'CIM'.

It is recommended that all new classes are defined using the single key property, InstanceID. This is the identification scheme to be used in CIM V3.

3 Do's and Don'ts in CIM Design

Sometimes it is as informative to review what was NOT done in a model, as to understand what is modeled. This section attempts to describe some modeling decisions and approaches taken by the CIM designers - both from the "DO" as well as the "DON'T" sides.

- Do not collapse Logical Devices and Physical Elements.

When using the CIM Schema and/or defining extensions, care should be taken to understand the Logical/Physical dichotomy and correctly analyze classes and properties. Be aware that there are usually different discovery and instrumentation mechanisms for these two classes of objects.

- Take care to distinguish between abstract and concrete associations on the basis of whether the association itself can be instantiated.

The concepts of abstract (conceptual) versus concrete (able to be instantiated) classes are important in CIM. Take care when linking concrete objects, as a concrete association must be defined. Alternately, it is allowed to define a concrete association, linking abstract objects. The latter can occur where the association is reasonably instantiated "as is" and where the association makes sense for the concrete subclasses of the original, abstract classes.

- It is not necessary to create a normalized model. Redundant/duplicated data may be needed.

A totally normalized model is not a requirement. Duplicated or derived data is sometimes desirable when the data is difficult to calculate, conceptually valuable in multiple locations in the model, or for other similar reasons. An example of redundant data can be found in the System Model, where diagnostic setting information is found in both CIM_DiagnosticSetting and CIM_DiagnosticResult. In the first case, the properties define the configuration TO BE USED for a test, and in the second case, they define the specific values USED BY the test whose results are reported.

- Do not force the model to be completely typed.

There are many examples where the CIM Schema specifies subclasses distinguished by the type of the object – for example, CIM_Sensor's Numeric Sensor, Binary Sensor, Discrete Sensor and Multi-State Sensor subclasses (in the Device Model) or all the subclasses of CIM_MediaAccessDevice (also in the Device Model). Subclasses exist where the various "types" have different properties or associations, where the Working

Groups felt that the classes were critical or where further vendor-supplied extensions would be created. In other cases, subclasses are not defined - but a "type" property is specified for a class. For example, CIM_PointingDevice has no subclasses but has a Type enumeration distinguishing between mice, track balls, etc. "Type" properties exist where subclasses would likely not have different properties or associations, or where the majority of the Working Groups felt that the classes would not have great impact/meaning in a management solution.

4 Modeling Methodology

Models are abstractions of “real world” objects and events. However, different abstractions or perspectives may exist for the same “real world”. For example, for Logical Devices, a Product, Physical, data flow or configuration perspective may be taken and would result in a very different model, if designed in isolation.

In order to address these issues and aid in the modeling of Core and Common Model extensions, the following methodology may be helpful.

- Define the various objects and methods (nouns and verbs) that the model(s) should address. It is important to first analyze the domain being modeled before any mapping to CIM is undertaken.
- Define the various “perspectives” that the model(s) should address, and the information required from each perspective
- Separate the information into object/class data and information specific to relationships between objects
- Review the CIM Schemas for similar concepts
- Given the separation of the data and the existing CIM class hierarchy, define appropriate classes and associations
- Attempt to place each property or attribute in one class only, and if it is not possible to do this (for example, data must be duplicated), thoroughly analyze why
- Where possible, collapse similar abstractions from several models and perspectives
- Define levels of granularity for the data (from high level to complex), allowing instantiation and detailed analysis to occur, when and as necessary

Appendix A – Change History

Version 0.9	June 18, 2003	Initial Draft – based on the CIM V2.4 Core White Paper

Appendix B – References

[1] Common Information Model (CIM) Specification, V2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

DMTF Specifications - Approved Errata - Downloadable from http://www.dmtf.org/standards/standard_cim.php

[2] CIM Core White Paper, DSP0111 – Downloadable from http://www.dmtf.org/standards/published_documents.php

[3] Unified Modeling Language (UML) from the Open Management Group (OMG) - Downloadable from http://www.omg.org/uml/