CIM Overview Document

June 2003

The Value of the Common Information Model (Why CIM?)

The DMTF Technical Committee

Managing a business' information technology environment is a difficult problem. It is exacerbated by each technology (and sometimes each product offering) representing its management data in a different way – by creating its own semantics, terminology, data structures and protocols. Instead of focusing on solving problems or improving overall performance, much time and effort is spent in tying together these "silos" of management data – trying to achieve a single consistent representation for all management data. This document reviews the DMTF's efforts in addressing these issues, and shows how CIM provides a broad-reaching abstraction layer for a business' management applications.

Introduction

Collecting the data to manage a business' computer and networking environments is one piece of the management problem. Another huge effort is normalizing and organizing that data. For example, a person knows that "OK", "Functional", "Operational" and "Working" are all synonyms for a "good" status. But, how does a computer program know this? Even worse, how does a computer program know where to look to find this information?

Unfortunately, the problem does not end with locating data and determining its semantics. There is a growing need to operate and manage a business in terms of the business' processes and services (both required and provided). In this environment, it is not very important for a management system to provide the low level detail of a "bad fan." Instead, it becomes critical to understand what service is lost, or what process cannot be completed (or must be failed over) due to the fan's failure.

End-to-end management, across multiple components, in a distributed environment is both a reality and a requirement. It is no longer sufficient to manage personal computers, servers, subnets, the network core, storage and software in isolation. These components all interoperate to provide connectivity and services. Information passes between these boundaries. Management must pass across these boundaries as well.

These are the problems addressed by the CIM Schema. The goal of the schema is to provide a common way to represent the computing and networking elements that make up a business' systems, and the relationships between these elements. The result is that both FCAPS management (fault, configuration, accounting, performance and security management) and the abstraction and decomposition of services and business operations are addressed. The information model defines and organizes common and consistent semantics for equipment and services. The model's organization is based on an object-oriented paradigm [1] – promoting the use of inheritance, relationships, abstraction, and encapsulation to improve the quality and consistency of management data.

A consistent information model is a required, basic building block for successful integration and use of management data both within an organization's business and IT departments, and between management vendors. CIM provides this information model for the enterprise and Internet management domains. Through CIM, more advanced customer solutions for inventory tracking, root cause analysis, and cross-vendor administration can be cooperatively developed.

Combining the best of all worlds, the CIM Schema incorporates concepts from other standards – such as the Internet Engineering Task Force's (IETF's) standard MIBs [2], and the International Telecommunication Union (ITU) [3] - and organizes the information using the CIM hierarchy. The other standards are not ignored or dismissed. They are reused and mapped into CIM.

Object-Oriented Modeling

The value of CIM stems from its object orientation [1]. Object design provides support for the following capabilities, that other "flat" data formats do not allow:

- Abstraction and classification – To reduce the complexity, high level and fundamental concepts (the "objects" of the management domain) are clearly defined. These objects are then grouped into types of management data ("classes") by identifying their common characteristics and features (properties), relationships (associations) and behavior (methods).

- Object inheritance – Below the high level and fundamental objects, additional detail can be provided through subclassing. 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, and to deliver continuity of management information. This object inheritance 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.

- 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 richer and more expressive approach in that relationships and associations are directly modeled. In addition, the way that these relationships are named and defined describe the semantics of the object associations (consider, for example, the Dependency and Component associations). Further semantics and information can be provided in properties (specifying common characteristics and features) of the associations.

- Standard, inheritable methods – An object's behavior is called a method, and by including methods CIM enables IT managers to act on management data. 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…regardless of the hardware, operating system or device.

The Goals and Benefits of CIM

Modeling goals were briefly mentioned in the Introduction (FCAPS management and the abstraction/decomposition of services). However, it is important to discuss the goals and uses of CIM, in a bit more detail.

Basically, all goals and uses derive from the ability to define a single model for management information and service semantics – and to position everything relative to that model. Low-level equipment details and high-level service composition are both supported – using different abstractions in the object hierarchy. Via relationships and associations, the roles of the equipment, software and low-level services (in providing functionality and supporting business processes) can be described. The tie of multiple settings (configurations) or statistics (performance management) to an element is also enabled via associations and the use of a well-understood object hierarchy. The object hierarchy indicates "where to look" for certain data, while the associations describe the relationships and applicability of the instances.

Of special significance is CIM's facilitation of data reuse, delivering consistency of information across products and releases of products. For example, a chassis is identified by the same class of objects regardless of whether it is a personal computer chassis or the packaging of a high-end router. The basic abstraction of a chassis holds true in both cases – similar to a person's ability to identify a "dog", despite variations across species.

In the computing and networking world, there are many general abstractions that can be defined across vendors, products and problem domains. Consider, for example, the concepts of service (such as diagnostics and databases), devices (such as power supplies or monitors), or administrative domains. These common and consistent data semantics can be provided independent of the repository of the data or the protocol transport used to retrieve it.

Taking a customer's perspective, costs can be contained since a single set of management tools and applications can be written, operating against clearly defined data. And, these tools need not change for each product and release since they are based on a single model and consistent abstractions. (Of course, change to support new product or management features would be encouraged, but the extent of this change is minimized.)

For example, let us examine the concept of service in more detail. All functions can be abstracted to a high-level concept called a "Service" in CIM. This class defines methods to start and stop an instance of Service, as well as properties indicating the Service's status and whether it is currently running. From this abstract class, "standard" print, diagnostic or storage management subclasses can be defined. All of these subclasses can be started and stopped using the same, high level methods. All of their statuses can be checked. Well understood relationships can be traversed to obtain more information, such as analyzing dependencies.

Another goal and benefit of CIM is its flexibility and support for extensions. This means a vendor or user can build on CIM to cover particular management areas. New subclasses may be defined, and/or new instances of existing classes created, as required to describe a computing or networking environment. For example, a customer could define a new high-level Service unique to their environment, and then specify dependencies on specific sub-services provided by various vendors. The vendors' services would also be defined using the model and therefore inherit/use the same methods and properties. Default settings for the sub-services could also be used and manipulated.

Using CIM, tools can now focus on managing, versus bringing together "silos" of data. Coupling the model with standard, interoperable access mechanisms (such as the DMTF's CIM Operations over HTTP [4] and a CIM XML DTD [5], IETF's LDAP APIs [2] in a directory environment, or OMG's Corba [6]) creates a complete management environment. Both standard information and standard access are available. The DMTF's goal is to enable a wide variety of access and implementation options. Our focus is on the data – which can be implemented in various ways (through repositories and infrastructures), and accessed via various protocols and encodings.

The CIM Schema is about well-understood and abstracted *information*. Mappings to a relational database, object oriented store, LDAP directory and other repositories are possible. The schema is designed to be "technology-neutral" with respect to its repository and access protocol. This is a key benefit of the model since there is no single implementation that fits all environments.

Closing Remarks

Modeling a business' computing and networking environments can be a daunting task. This is the ultimate goal of the CIM Schema. With technology continually advancing, the work will never be complete. For this reason, the model is written assuming an incremental development approach, with both top-down and bottom-up design. Details are present to allow FCAPS analyses (the bottom-up approach) and to map the best details from the various other standards bodies such as IETF and ITU. CIM also provides general abstractions and the ability to compose/decompose higher-level services and functions, permitting management at a less granular level (the top-down approach). No longer are there "silos" of data that must be translated, interpreted and normalized. With CIM, data has meaning, is consistent across vendors and products, and is more than bits on the wire.

References

[1] DMTF CIM Concepts White Paper and, DMTF CIM Specification
[2] IETF RFCs
[3] ITU
[4] DMTF CIM Operations over HTTP
[5] DMTF CIM XML DTD
[6] OMG Corba