CIM System Model White Paper

CIM Version 2.7

Document Version 1.0 June 10, 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.

The CIM System Common Model defines computer system-related abstractions. Many of the concepts related to computer systems derive from the CIM_System abstraction in the Core Model. CIM_System describes the aggregation of 'parts' (or components) into a single, manageable 'whole' (the system).

Besides the concept of the computer system itself, the System Model also addresses components and functionality associated with most computer systems. These include concepts such as file systems and files, operating systems, jobs, processes and threads, and diagnostics. In addition, both general purpose and 'dedicated' systems can be described. This is indicated using a simple enumeration (the Dedicated property in CIM_ComputerSystem).

The goal of this paper is to overview the concepts that are currently modeled in the CIM 2.7 System Model. This paper mirrors the organization of the classes as they are presented in the MOF and Visio diagrams.

Notice

DSP150 ***Status: Preliminary***

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

1 Introduction

1.1 Overview

The CIM System Common Model defines computer-system related abstractions. Many of the concepts related to computer system derive from the CIM_System abstraction in the Core Model. CIM_System describes the aggregation of 'parts' (or components) into a single manageable 'whole' (the system).

Important concepts related to a CIM_System are:

- Systems act as aggregation entities.
- Systems are not modeled as a collection.
- A system is more than the sum of its parts.
- Systems have status and they host services and access points.
- Systems are top-level objects that are frequently used to scope their aggregated entities.

Besides the concept of the computer system itself, the System Model also addresses components and functionality associated with most computer systems. These include concepts such as file systems and files, operating systems, jobs, processes and threads, and diagnostics. In addition, both general purpose and 'dedicated' systems can be described. This is indicated using a simple enumeration (the Dedicated property in CIM_ComputerSystem).

Note that there are no specific subclasses of CIM_ComputerSystem to describe purpose or functionality (i.e., to distinguish a router, storage array, or print server, for example). The Dedicated property exists to help classify a system, but functionality is defined by the services that are hosted on (or are capable of being hosted on) the computer. Taking this approach allows the instantiation of *one* computer serves multiple purposes – as a router, print server and storage array – instead of having to create three instances of different CIM_ComputerSystem subclasses. It is certainly conceivable that a single system could provide multiple services and be 'dedicated' to several functions.

1.2 Background Reference Material

In addition to this white paper, more information can be found in the following documents:

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/

Internet Engineering Task Force (IETF) - MIBs and Work Group information at http://www.ietf.org

Core White Paper, DSP111, June 2003 – Downloadable from http://www.dmtf.org/standards/published_documents.php#whitepapers

2 The System Model

2.1 Background and Assumptions

It is assumed that the reader is familiar with the concepts and terminology presented in the CIM Specification, CIM White Paper and Core White Paper. The System Model extends from the class structures and framework of the CIM Core Model.

It is possible (and probable) that additional objects and properties will be defined in subsequent releases of the System Model. The addition of classes, properties and associations does not have the same impact as deletions and modifications. Additions do not impact current Common and Extension Models. Existing application and instrumentation code should continue to function.

2.2 Conceptual Areas Addressed by the Model

Many of the concepts related to computer system derive from the CIM_System abstraction in the Core Model. CIM_System describes the aggregation of 'parts' (or components) into a single manageable 'whole' (the system). Besides the concept of the computer system itself, the System Model also addresses components and functionality associated with most computer systems. These include concepts such as file systems and files, operating systems, jobs, processes and threads, and diagnostics. Therefore the Common System Model is broken down into the following conceptual areas:

- System Elements – Subclasses of System (ComputerSystem, Virtual, Cluster, etc.)
- File Elements – Defines FileSystems, Files, Directories, NFS, etc.
- Operating System – Defines Operating System and its key relationships
- Processing and Jobs – Represents the processes, threads, and jobs
- Services & Service Access Points – Defines high-level clustering and boot services and service access points
- Time – Defines a setting for fully describing a timezone
- Unix – Defines Unix-specific extensions
- System Resource – Defines system hardware and software resources
- Logs – Defines message logs and log records
- Diagnostics – Represents the actual diagnostic test, configuration for that test, and their results

2.3 System Elements

As stated previously, many of the concepts related to computer system derive from the CIM_System abstraction in the Core Model. This means System, and most of its high-level subclasses, is actually defined in the Core Model and, therefore, they are not defined in the System Model MOF. The subclasses that are in the System MOF include:

- **StorageLibrary**
  The storage library describes storage locations and media. Since a StorageLibrary may be totally manual, it is not a subclass of computer system. When automated, a computer system is a component of the library.

- **ComputerSystem**
  The computer system describes a generic system that is used for computational purposes. Therefore, a computer system requires both hardware and an operating system. The role of the operating system is to control resources, execute code, and etc). The operating system may be specialized firmware. Currently there are three subclasses of ComputerSystem. They are VirtualComputerSystem, Cluster, and UnitaryComputerSystem. However, further attempts to refine the Computer System model with these categories of subclassing proved problematic. For example, when trying to define a standard storage subsystem, where does one subclass? For some implementations the storage subsystem might be a cluster, others might be a unitary computer system, while other may be virtual computer systems.

  On the other hand, creating specific subclasses to describe system functionality is also problematic. The functionality of the system should be described by the services that are hosted or are capable of being hosted. Otherwise, the result is individual subclasses for routing, storage, etc. It is conceivable that a single system could provide all of these. Therefore, the Dedicated property should be used to describe the primary functionalities the system supports.

  These issues are being addressed by the System and Device working group. Model changes regarding virtualization , clustering, and unitary computer system are expected. Changes regarding ComputerSystem are not expected.

- **AdminDomain**
  This is a special grouping of ManagedSystemElements. The grouping is viewed as a single entity, reflecting that all of its components are administered similarly (e.g., by the same user, group of users, or policy). It serves as an aggregation point to associate one or more network devices (e.g., routers and switches), servers, and other resources that can be accessed by end systems. This grouping of devices plays an essential role in ensuring that the same administrative policy and actions are applied to all of the devices in the grouping. The specific behavior and/or semantics of the AdminDomain can be identified through its aggregated and associated entities.

2.4 File Elements

File elements include the following:

- **FileSystem**
  Filesystem represents the file store and the services to access that store. It contains information such as type, block size, path to root directory, read-only indication, available space, etc. FileSystem further refines into remote or local file systems.

  Filesystem is weakly associated to the ComputerSystem that controls it. The mandatory **HostedFileSystem** association derived from SystemComponent defines the relationship between a FileSystem and its "single" hosted system. The FileSystem itself can only be hosted by one system. However, the abstract nature of "system" itself allows the FileSystem to be distributed across many physical systems.

  The **ResidesOnExtent** association derived from Dependency represents the relationship between a file system and the underlying storage where it is "physically" located.

- **LocalFileSystem**
  LocalFileSystem, derived from FileSystem, represents a file system that is directly accessed by its ComputerSystem.

- **RemoteFileSystem**
  RemoteFileSystem, derived from FileSystem, represents a file system that is accessed using a network service.

- **NFS**
  The NFS object and its associations describe the sharing of files between Computer Systems. NFS functions by allowing the "export" of directories from a Local File System and their "mount" by a remote Computer System. Mounted directories are accessed as though they were local. The remote directories are grafted into the Local File System. The relationships of "Export" and "Mount" are described by associations of the same names below, with the Directory class.

- **LogicalFile**
  LogicalFile is the superclass for various file types. A file is weakly associated to the FileSystem on which it is located. LogicalFile further refines into directories, data files, device files, symbolic links and FIFO pipes. LogicalFile contains information such as name, size, access modes, etc.

  The **FileStorage** association, derived from Component, represents the mandatory relationship between a file and its FileSystem. This relationship is mandatory since the definition of a file can only be guaranteed to be unique in the context of its FileSystem.

- **Directory**
  Directory derives from LogicalFile. It is a type of file that contains a group of LogicalFiles. Therefore, the mandatory FileStorage relationship is used to define the directories that are contained by the FileSystem.

  The **DirectoryContainsFile** association derived from Component represents the relationship between a directory and the logical files it contains. A LogicalFile can only belong to at most one directory.

  The **Mount** association, derived from Dependency, represents the operation of attaching at most one directory to a file system. Typically, the FileSystem referenced by the Mount relationship is not the same FileSystem that is referenced by the FileStorage relationship. Conceptually, the referenced FileSystem in Mount is the result of the act of mounting versus the actual backing store. When the directory's FileStorage is remote (versus local), it is recommended that the CIM_Export association be defined for remotely accessed/mounted Directories.

  The **Export** association represents the local store for the exported directory.

- **DeviceFile**
  DeviceFile derives from LogicalFile. It is a type of file that represents a device.

  The **DeviceAccessedByFile** association, derived from Dependency, indicates the device that is being accessed by the DeviceFile.

- **SymbolicLink**
  SymbolicLink derives from LogicalFile. It is a type of file that refers to another file, thus enabling the target file to be referred to by many names. On Unix, a symbolic link contains the path to the target file.

- **FIFOPipeFile**
  FIFOPipeFile derives from LogicalFile. It represents a type of file that is used as an inter-process communication mechanism, where different processes read from it and write to it in FIFO manner.

2.5 Operating System

An OperatingSystem is software/firmware that makes the ComputerSystem hardware usable, and implements and/or manages the resources, file systems, processes, user interfaces, services, etc., that are available on the ComputerSystem.

- **BootOSFromFS**
  The BootOSFromFS is used to indicate from which file system the operating system is loaded. To accommodate a distributed operating system, the operating system may have relationships to many different file systems.

- **RunningOS**
  RunningOS is used to indicate which operating system is currently running on which computer system.

- **InstalledOS**
  InstalledOS is a short cut to indicate the computer system that contains the underlying storage (disk drive or memory) for the operating system. To accommodate a distributed operating system, this relationship would run to the higher level computer system and that computer system is comprised of the underlying computer systems. Since a computer system can have multiple operating systems installed, a computer system can have multiple InstalledOS relationships to various instances of operating system.

- **OperatingSystemSoftwareFeature**
  The OperatingSystemSoftwareFeature is used to indicate the software features that are part of the operating system.

2.6 Processing and Jobs

2.6.1 Process

Process represents a running program. A user of the OperatingSystem will typically see a Process as an application or task. Within an OperatingSystem, a Process is defined by a workspace of memory resources and environmental settings that are allocated to it. On a multitasking System, this workspace prevents intrusion of resources by other Processes. Additionally, a Process can execute as multiple Threads, all which run within the same workspace.

- **OSProcess**
  A process is weakly associated to the operating system that controls it. The OSProcess association defines this relationship.

- **ProcessExecutable**
  The ProcessExecutable association defines the relationship between a data file and the process using it. The data file might be the file representation of the process itself.

- **ServiceProcess**
  The ServiceProcess relationship indicates if a Service is running in a particular Process. It is also used to indicate, via the ExecutionType property, if the Service started and is wholly responsible for the Process, or if the Service is running in an existing Process, perhaps with other unrelated Services, which is owned or started by a different entity. This association relates a Service with an externally visible system signature.

- **Thread**
  Threads represent the ability to execute units of a Process or task in parallel. A Process can have many Threads.

  Each thread is weakly associated to its controlling Process. The **ProcessThread** association defines this relationship.

2.6.2 Job & Concrete Job

Job is an abstract definition that represents a unit of work for a System, such as a print job. A Job is distinct from a Process or Thread in that a Job can be scheduled. When waiting to execute, a Job may be placed in a queue, or more generically, into a Job Destination object.

- **ConcreteJob**
  ConcreteJob is an instantiable subclass of the abstract Job class.

- **ProcessOfJob**
  The ProcessOfJob aggregation defines the processes that are apart of the Job.

- **AffectedJobElement**
  The AffectedJobElement relationship indicates that the reference element is affected by the running of the job. This association contains a property, ElementEffects, which describes the actual affect.

- **OwningJobElment**
  The OwningJobElment relationship indicates the element that owes the job.

- **JobDestination**
  JobDestination represents a Job that is submitted for processing (i.e., the logical entity users 'point to' if they request a job to be processed). It is a representation of the fact that this entity can accept jobs of specific types, and can process and execute them appropriately. Job Destinations can be, but do not have to be, queues. This association represents the fact that computing environments allow users to submit specific types of requests, such as a print job, to a named resource that is capable of processing this request. For the submitter, it is irrelevant whether the named resource is a direct printer, a single queue with an attached printer, or a complex queuing system with alternate, remote and backup queues and device servers. Job Destinations can also represent aliases for other Job Destinations.

  Note that not all schedulers provide the explicit concept of a job destination. For example, the UNIX cron(1) does not support the specification of the target by name. The implicit 'JobDestination' is the OS with the processor. In these cases, the modeler should consider whether the 'artificial' introduction of a single and fixed JobDestination object provides value on the abstraction level.

JobDestinations are hosted on Systems, similar to the way Services are hosted on Systems. The **HostedJobDestination** defines the relationship between a JobDestination and its hosting System. The **JobDestinationJobs** relationship indicates the JobDestination for a particular Job.

2.7 Services & Service Access Points

2.7.1 Services

The Service class represents the configuration, operational data and management of "function." The semantics are not about the function itself, but information needed to manage it. For example, although you might have "email" or "word processing" services on your computer, you would not use CIM to handle your individual mail messages, or open and edit documents and files. You would use the native utilities and software of your computer system. These are the operating and executing entities that implement Services. CIM is used to describe the existence of email and word processing Services, to configure them, and to diagnose them if a problem occurs. Currently, the specific services defined for Systems are ClusteringService, BootService, OBBAlertService and WakeUpService.

- **Clustering Service**
  Clustering Service represents generic clustering functionality (e.g., add or evict a node from the Cluster), and should be subclassed to add the properties and methods required for particular types of clustering. For example, for a failover Cluster, one might define a method that invokes an application's proactive fail to another node. It is assumed that the Clustering Service is hosted on the Cluster System itself, whereas the Clustering SAPs (Service Access Points) are hosted on the individual nodes of the Cluster.

- **Boot Services**
  Boot Services represent the functionality provided by a Logical Device (such as a Disk Drive with a bootable Partition, on which an Operating System is loaded), software (such as the requirements of a Virtual Computer System), or a network (such as BootP Services) to load an Operating System on a Unitary Computer System. Boot Services can be hosted on any System, not just a Computer System, since booting from the network is possible. Boot SAPs (Service Access Points) are hosted on the Unitary Computer Systems themselves (e.g., the Systems needing to boot).

2.7.2 Service Access Points

Designed as a complementary class to Service, ServiceAccessPoint models the utilization and invocation of a Service. It represents a Service that is made available for use by other entities. Access Points are not the APIs, DLLs, or OS commands; they are not the definition to invoke Services (these are actually the software implementations of

Services); but they are the abstraction of access to a Service. One could distinguish Service and Service Access Points (SAPs) in the context of a "provider-consumer" relationship. Service represents the current configuration and operational data of the *provided* function, and Service Access Points are the way to manage the *consumption* (or access) of that function.

In client-server terms, Service is the function at the server, while the SAPs are the management of a client's use of the Service. Service represents the management of any kind of function and is a very abstract concept. For example, on a personal computer, a wide variety of Services may be running - word processing, presentation preparation, email client, meeting scheduling (including a local/offline store), and much more. In addition, when specific Services are not locally available, they can often be accessed using the network. Access could be modeled as instances of Service Access Points.

Examples of the latter are SAPs to pull email from a server, to print at a network printer, or to pull the latest meetings from the corporate meeting database. In all of these examples, there is actually a layering or dependency of Access Points. The application-oriented Access Points are dependent on network access. Network access is modeled as instances and subclasses of Protocol Endpoints, which are in turn subclasses of Service Access Points in the CIM Network Model.

Currently, the specific service access points defined for System are ClusteringSAP and BootSAP.

2.8 Time

TimeZone is a Setting that represents the various time zones that a system can use. For a particular time zone, this class defines its name (long and short), when daylight savings time starts and ends, when standard time starts and ends, and when this time zone definition was or will be first used. The start and end times are defined by the values in a combination of properties.

The ElementSetting relationship defined in the Core model is used to indicate the ManagedSystemElement(s) that the TimeZone setting applies to.

2.9 Unix

The following represent class extensions that are specific to Unix-based operating systems. The classes and properties are based on the Single Unix Specification (at http://www.opengroup.org/onlinepubs/7908799/toc.htm) or the DMTF Unix MIF. MappingStrings property qualifiers are used to indicate how the value is generated or defined. For example, **MappingStrings {"POSIX.TOG|pathconf|_PC_LINK_MAX"}** shows that the property value is generated by executing the *pathconf* command with the input argument *PC_LINK_MAX*.

- **UnixLocalFileSystem**
  UnixLocalFileSystem derives from LocalFileSystem to represent a Unix file system that is accessed directly by the computer system (i.e., without relying on a file server). It introduces properties that are specific to a Unix filesystem, such as the number of free inodes and total inodes. The *statvfs* system interface, as described in the Single Unix Specification, defines the information that can be retrieved for a file system. It is weakly associated to the computer system via its super class LocalFileSystem. For a cluster, the file system is weakly associated to the cluster.

- **UnixProcess**
  UnixProcess derives from Process. It introduces properties that are specific to a Unix process. These new properties represent values produced by the "*ps*" command, as described in the Single Unix Specification. Since UnixProcess subclasses from Process, it is weakly associated to the operating system that controls it. The OSProcess association adequately defines this relationship.

- **UnixThread**
  UnixThread derives from Thread. It introduces properties that are specific to a Unix thread. These properties include information about scheduling policy, concurrency level and contention scope. More details of the properties can be obtained from the MOF and the Threads section in the Single Unix Specification at http://www.opengroup.org/onlinepubs/007908799/xsh/threads.html. The values for these properties can be found in *sched.h* and *psched.h*. As inherited from Thread, it is weakly associated to the Process that controls it. The ProcessThread association adequately defines this relationship.

- **UnixProcessStatisticalInformation**
  UnixProcessStatisticalInformation derives from StatisticalInformation. It contains the statistics that are specific to a Unix process. The statistics include information about various aspects of memory utilization. Additionally, CPU utilization statistics of the process and its children are also available. The UnixProcessStatistics association is derived from Statistics to define the relationship between a Unix process and its statistics.

- **UnixFile**
  UnixFile derives from LogicalElement. It contains information specific to Unix files such as the UID and GID owner, inode number, and various POSIX settings. These new properties represent values produced by the *ls* and *pathconf* commands, as described in the Single Unix Specification.
  UnixFile does *not* derive from LogicalFile, because its properties also apply to other types of Unix files such as directories, symbolic links, device files, etc. Instead, UnixFile is tied to its corresponding LogicalFile by the FileIdentity association, which derives from LogicalIdentity.

- **UnixDirectory**
  UnixDirectory derives from LogicalDirectory, which is a LogicalFile subclass. It has one additional property that describes the maximum size of file allowed within that directory. This value is produced by executing *pathconf* command with input argument *_PC_FILESIZEBITS,* as defined in the Single Unix Specification.

- **UnixDeviceFile**
  UnixDeviceFile derives from DeviceFile, which is a LogicalFile subclass. It contains information specific to Unix device files, such as major and minor device numbers.

2.10 System Resources

System Resources are individually identifiable entities that are managed by software, and are available for use by application software and/or LogicalDevices. Resources may be shared and are allocated entities that are assignable, reservable, counted/tracked, releasable, reset-able, etc.

Examples of software Resources are message queues, shared memory segments (identified by a key value), and named pipes, while examples of hardware Resources (in an x86 environment) are IRQs, DMA channels and memory mapped I/O. SystemResource represents system resources that may be used by a device.

- **AllocatedResource**
  AllocatedResource represents the relationship of a resource to a consuming device.

- **AllocatedDMA**
  AllocatedDMA specializes AllocatedResource to represent the relationship of a DMA resource to a computing device.

- **ResourceOfSystem**
  The ResourceOfSystem (a specialization of SystemComponent) relationship ties resources to the owning system. The owning system is the system that provides the context for the resource. The system may or may not provide the backing implementation for the resource. For example, a print queue is managed in the context of a specific print server.

- **ComputerSystemResource**
  The ComputerSystemResource (a specialization of ResourceOfSystem) ties resources to a single ComputerSystem. The **ComputerSystemMappedIO, ComputerSystemIRQ,** and **ComputerSystemDMA** relationships further refine the ComputerSystemResources from a ComputerSystem to an individual resource.

2.11 Logs

Logs include the following:

- **MessageLog**
  MessageLog represents any type of event, error, or informational register or chronicle. This object describes the existence of the log and its characteristics. Several methods are defined for retrieving, writing and deleting log entries, and maintaining the log.

  MessageLogs can make use of three different associations to indicate their backing store. **LogInStorage** is used to indicate that the backing store is Memory, **LogInDataFile**, for a file, and **LogInDeviceFile** when the log is piped to a device.

  The **UseOfMessageLog** association is used to indicate the ManagedSystemElement that is making use of the MessageLog.

  The **RecordInLog** aggregation is used to place LogRecords into the MessageLog.

- **LogRecord**
  The LogRecord object can describe the definitional format for entries in a MessageLog, or can be used to instantiate the actual records in the Log. The latter approach provides a great deal more semantic definition and management control over the individual entries in a MessageLog, than do the record manipulation methods of the Log class. It is recommended that the data in individual Log entries be modeled using subclasses of LogRecord, to avoid the creation of LogRecords with one property (such as RecordData) without semantics.

  Definitional formats for LogRecords could be specified by establishing a naming convention for the RecordID and MessageTimestamp key properties.

2.12 Diagnostics

Diagnostics include the following:

- **DiagnosticTest**
  This specifies the actual test that is to be performed. It is expected that there will be a subclass for every manufacturer of diagnostics. Diagnostic test is a subclass of Service, and is therefore weakly associated to the system that hosts it. The test can be started and stopped.
  The DiagnosticTestForMSE indicates the possible Managed System Elements that the diagnostic test can be run against.

- **DiagnosticSetting**
  This is used to store the configuration for a given test. The DiagnosticSettingForTest indicates the relationship between the setting and the test.

- **DiagnosticResult**
  This specifies the result of the execution of the diagnostic test. The result is weakly associated to the test that creates it. This is described by the DiagnosticResultForTest relationship. In addition the DiagnosticResultForMSE relationship indicates the Managed System Element that the test was executed against.

Implementation feedback is driving changes and cleanup to the diagnostic model. These changes are expected in version 2.8.

3 Relationships to Other Standards and Specifications

3.1 Overlapping Standards and Specifications

A lot of the objects, properties, and association in the system models were mapped from:

- DMI MIFs (Management Information Format files, available at http://www.dmtf.org/standards/standard_dmi.php
- SNMP MIBs (Management Information Bases, available at www.ietf.org)
- As stated previously, the Unix sub-model's classes and properties are based on the Single Unix Specification (at http://www.opengroup.org/onlinepubs/7908799/toc.htm) or the DMTF Unix MIF.

3.2 A Mapping of DMI MIFs into the Model

The MappingStrings qualifiers are used to indicate how the CIM class property's value correlates to the DMI attribute. For example,

MappingStrings {"MIF.DMTF|System Hardware Security|001.4"}

shows that the property correlates to version 4 of the DMI "System Hardware Security" group's attribute ID # 1.

In general, the MIF mapping string takes on the form:

MappingStrings {"MIF.DMTF|<DMI group name>|<attribute ID>.<group version #>"}

3.3 A Mapping of SNMP MIBs into the Model

The MappingStrings qualifiers are used to indicate how the CIM class property's value correlates to the SNMP attribute. For example,

MappingStrings {"MIB.IETF|MIB-II.sysServices"}

shows that the property correlates to the sysServices object in the IETF's MIB-II mib definition.

In general, the MIB mapping string takes on the form:

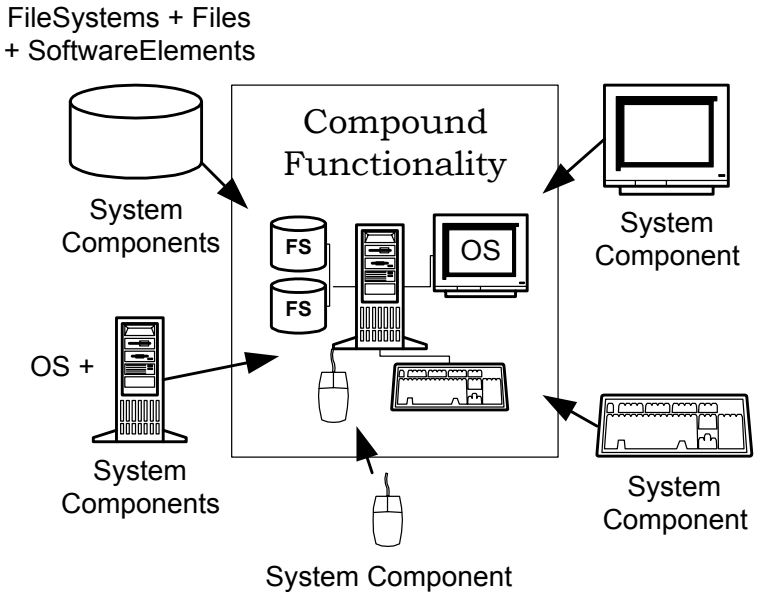
MappingStrings {"MIB.IETF|<IETF MIB Name>.<object name>"}

3.4 A Mapping of the Unix Specification into the Model

The MappingStrings qualifiers are used to indicate how the CIM class property's value is generated or defined in the context of the Unix Specification. For example, **MappingStrings {"POSIX.TOG|pathconf|_PC_LINK_MAX"}** shows that the property value is generated by executing the *pathconf* command with the input argument *PC_LINK_MAX*.

4 System Model Use Case

4.1 Server Example



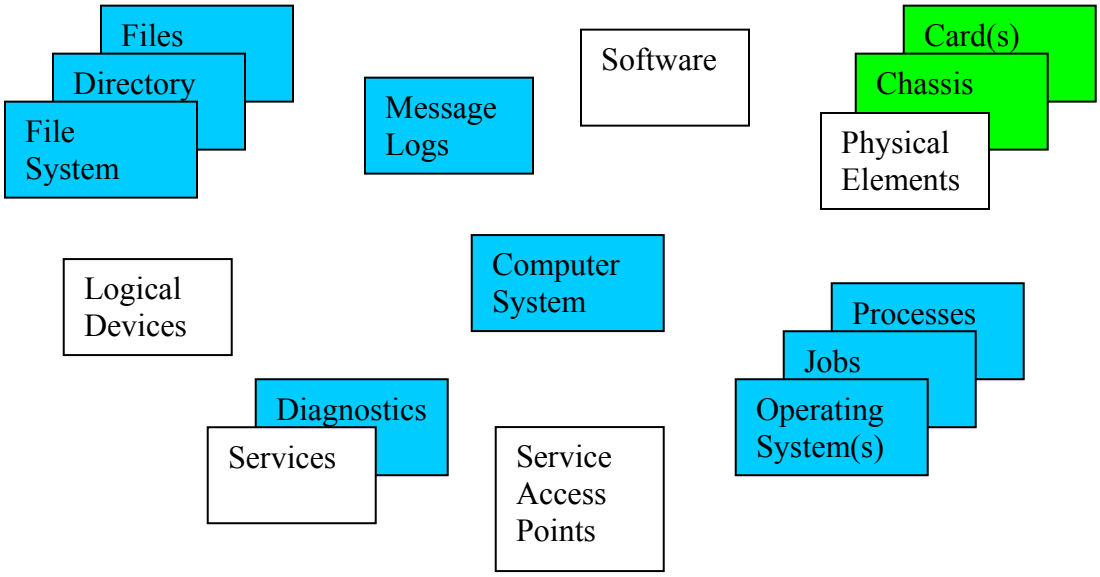
When examining a typical server, we are likely to find the following logical elements:

- Several installed operating systems, and one running operating system which has jobs and processes
- Local and remote file systems that are composed of directories and files
- Devices such as a Monitor, Keyboard, Mouse, Hard Disk, Processor, Power Supply, Fan, etc.
- Services and network interfaces that are hosted on the server itself, such as diagnostic services and IP endpoints
- Services that are available to the server via service access points, such as a print service

In addition, the server has physical aspects such as:

- A chassis or multiple chassis, which may be mounted in a rack
- Cards and components (chips) within chassis
- Locations for the physical entities

In the CIM environment, the server can be broken into its logical components (its devices), related to its physical aspects (via association), and be referenced in dependency and hosting relationships (such as the hosting of services and access points). The logical aspects of a computer are shown below. (The elements denoted in blue are defined in the System Common Model.)



As regards the other boxes in the figure above:

- The high level abstractions (services, access points, devices, etc., denoted in white) are defined in the Core Model.
- The specific subclasses of PhysicalElements (denoted in green) are defined in the Physical Common Model.
- The specific subclasses of LogicalDevice (such as power supplies, fans, monitors and keyboards) are defined in the Device Common Model.
- The asset aspects of Software are defined as instances of the SoftwareIdentity class, found in the Core Model.
- The management of software deployment is addressed by the Application Common Model.

5 Future Work

- As mentioned in section 2.3, clean up work regarding the VirtualComputerSystem, Cluster, and UnitaryComputerSystem class definitions is underway for CIM V2.9.
- As mentioned in section 2.12, changes and cleanup to the diagnostic model will be seen in CIM V2.8.
- Considerations for OperatingSystem provisioning will likely generate changes regarding how an operating systems “type” and “version” are specified.

Appendix A – Change History

Version 1.0	June 10, 2003	Initial Draft

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

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

Appendix C – Extending the Model

As previously stated in section 2.3, extending the definition of system for specialization is problematic. Most likely, any "system" specialization needs to be made as extensions to Service, LogicalDevice, Capabilities, Settings, and/or StatisticalData.

Vendor extension for describing a specific diagnostic test are expected and encouraged.

Appendix D – Considerations for Implementation

1. Use of the System object should be limited to compound functional entities with significance in the enterprise.

2. The Operating System reference in the Running OS association should also be referenced in an instance of the Installed OS association.

3. Unitary Computer System's Last Load Info property should be set to an entry from the Initial Load Info array.

4. If the Operating System's Distributed property is set to TRUE, all Computer Systems running a single copy of the OS should be grouped together in a Cluster.

5. Creation of Computer System subtypes on the basis of hardware alone should be avoided. If possible, subtypes should be defined by a combination of hardware architecture and an operating system, which results in a "functional whole".

6. Care should be taken that the Local File System is weak with respect to the same Computer System that scopes the Storage Extent on which the File System resides.

7. If data is duplicated, take care that the correct items are appropriately related and that the data is consistent across the objects.