CIM Policy Model White Paper

CIM Version 2.7

2.7.0 18 June 2003

Abstract

The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in Internet, enterprise and service provider 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.

This white paper describes the CIM Policy Model, as defined by the DMTF Policy Working Group, for the CIM Schema Release 2.7.

In today's complex, multi-vendor environments, successful, scaleable management of service levels depends on the specification of unified, scaleably administered policies. These policies must then map to the configuration of multiple heterogeneous systems, devices, applications, and networks, for the purpose of policy enforcement. The resulting cooperation of these multiple managed entities produces aggregate behavior consistent with the desired policies, and which, in turn, enables the delivery of service at agreed-upon levels.

The CIM Policy Model is a key component in enabling application developers, network administrators, and policy administrators to represent and manage policy across a spectrum of technical domains, including networking, security, and system admin. Policy-related work in other DMTF working groups and standards bodies is also referenced in this paper.

Notice

DSP108 ***Status: Preliminary***

Table of Contents

1 Introduction

1.1 CIM Policy Model Documentation and Influences

This white paper describes the Policy Model, in the CIM Schema Release 2.7.

Together with the associated MOF file and Visio drawing, this paper documents the extensions to the DMTF's Common Information Model for representing policy information. This model is heavily influenced by the CIM-based IETF (Internet Engineering Task Force) Policy Core Information Model (PCIM), which is defined by the IETF Policy Framework Working Group [RFC3060, RFC3460]. In turn, the Common Information Model has heavily influenced the IETF's Core Policy Information Model, since it is also CIM-based, originally deriving from CIM Schema Version 2.2. Both models draw on the pre-standard work of the DEN Ad Hoc Working Group [DEN], as well as synergistic work in other DMTF working groups, including the Network Working Group.

Liberal use was made of the text from the above referenced PCIM RFC, in the creation of this document. Many of the contributors to that document worked on this model as well. The models proceeded in parallel in the two organizations, with liberal idea exchange between the two groups. The source of the differences between the two models can be attributed largely to the fact that PCIM utilized CIM Schema 2.2 as its base, since that was publicly available at the time the IETF's work was proceeding. The model documented here utilizes CIM Schema 2.6 as its base. In addition, as work proceeded in the IETF, some aspects were not included in the DMTF Policy Model. Classes representing variables and values have not been included in the DMTF schema, since the Policy Working Group is instead examining an alternative (the definition of generic conditions and actions).

1.2 Objectives

In today's complex, multi-vendor environments, successful, scaleable management of service levels depends on the specification of unified, scaleably administered policies. These policies must then map to the configuration of multiple heterogeneous systems, devices, applications, and networks, for the purpose of policy enforcement. The resulting cooperation of these multiple managed entities produces aggregate behavior consistent with the desired policies, and which, in turn, enables the delivery of service at agreed-upon levels.

The CIM Policy Model facilitates such consistent behavior by enabling an administrator to represent policy in a vendor-independent and device-independent way. Thus, service level and other high-level policy abstractions can be supported, and be translated to device-specific configuration parameters at a lower level, across an aggregate of heterogeneous managed entities.

Thus, the CIM Policy Model becomes a key component in enabling application developers, network administrators, and policy administrators to represent and manage policy across a spectrum of technical domains, including networking, security, and system admin.

1.3 Use and Extension of Model

The policy classes and associations defined in this model are sufficiently generic to allow them to represent policies related to virtually anything. However, it is expected that initial application of this model will be to represent policies related to IP networking, including QoS (DiffServ and IntServ), and to IPSec, as well as to other applicable technical domains including access control, and storage system backup policies.

Models for application-specific areas may extend the CIM Policy Model in several ways. The preferred way is to use the CIM_PolicyGroup, CIM_PolicyRule, and CIM_PolicyTimePeriodCondition classes directly, as a foundation for representing and communicating policy information. Then, specific subclasses derived from CIM_PolicyCondition and CIM_PolicyAction can capture application-specific definitions of conditions and actions of policies. New work on generic conditions (conditional expressions involving CIM classes and properties) and generic actions (invoking CIM Schema methods and operations) will avoid significant subclassing of PolicyCondition and PolicyAction. This work will be released as part of CIM V2.9.

Two subclasses, CIM_VendorPolicyCondition and CIM_VendorPolicyAction, are also included in this model, to provide a mechanism for *legacy*, vendor-specific extensions to the CIM Policy Model.

2 The Policy Model

2.1 Background and Assumptions

The classes comprising the Policy Core Information Model serve as an extensible class hierarchy for defining policy objects that enable application developers, network administrators, and policy administrators to represent policies of different types.

One approach to modeling a policy-controlled collection of systems and devices, such as a network, is as follows: First model the network as a state machine and then use policy to control which state a policy-controlled device should be in, or is allowed to be in, at any given time. Given this approach, policy is applied using a set of policy rules. Each policy rule consists of a set of conditions and a set of actions. Policy rules may be aggregated into policy groups. These groups may be nested, to represent a hierarchy of policies.

2.2 Overview

The set of conditions associated with a policy rule specifies when the policy rule is applicable. The set of conditions can be expressed as either an ORed set of ANDed sets of condition statements or an ANDed set of ORed sets of statements. Individual condition statements can also be negated. These combinations are termed, respectively, Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF) for the conditions.

If the set of conditions associated with a policy rule evaluates to TRUE, then a set of actions that either maintain the current state of the object or transition the object to a new state may be executed. For the set of actions associated with a policy rule, it is possible to specify an order of execution, as well as an indication of whether the order is required or merely recommended. It is also possible to indicate that the order in which the actions are executed does not matter.

Policy rules themselves can be prioritized. One common reason for doing this is to express an overall policy that has a general case with a few specific exceptions.

For example, let’s say that a network administrator wants to establish a policy such that certain users or applications get more bandwidth or better delay performance across a given network domain, than other users or applications. In such a network, different service levels could be established. Let’s call them “Bronze Service”, for the lower priority users, and “Gold Service” for the higher priority users. Such a policy might be implemented by a general QoS policy rule, which specifies that traffic originating from members of the engineering group is to get “Bronze Service.” A second policy rule might express an exception: traffic originating from John, a specific member of the engineering group, is to get “Gold Service”. Since traffic originating from John satisfies the conditions of both policy rules, and since the actions associated with the two rules are incompatible, a priority needs to be established. By giving the second rule (the exception) a higher priority than the first rule (the general case), a policy administrator can get the desired effect: traffic originating from John gets Gold Service, and traffic originating from all the other members of the engineering group gets Bronze Service.

Policies can either be used in a stand-alone fashion or aggregated into policy groups to perform more elaborate functions. Stand-alone policies are called policy rules. Policy groups are aggregations of policy rules, and/or aggregations of policy groups. Policy groups can model intricate interactions between objects that have complex interdependencies. Examples of this include a sophisticated user logon policy that sets up application access, security, and reconfigures network connections based on a combination of user identity, network location, logon method and time of day. A policy group represents a unit of reusability and manageability in that its management is handled by an identifiable group of administrators, and its policy rules apply equally to the scope of the policy group.

The CIM_PolicyGroup class is a generalized aggregation container. It enables CIM_PolicyRules and/or CIM_PolicyGroups, to be aggregated in a single container. Loops, including the degenerate case of a CIM_PolicyGroup that contains itself, are not allowed when CIM_PolicyGroups contain other CIM_PolicyGroups. Note that a CIM_PolicyGroup can nest other CIM_PolicyGroups, and that there is no restriction on the depth of the nesting in sibling CIM_PolicyGroups.

As a simple example, think of the highest-level CIM_PolicyGroup as a logon policy for US employees of a company. This CIM_PolicyGroup might be called USEmployeeLogonPolicy, and might aggregate several CIM_PolicyGroups that provide specialized rules per location. Let's say CIM_PolicyGroup, instance "A" defines logon rules for employees on the West Coast, while another CIM_PolicyGroup, instance "B", might define logon rules for the Midwest, and so forth.

Note also that the depth of each CIM_PolicyGroup does not need to be the same. Thus, the WestCoast PolicyGroup might have several additional layers of CIM_PolicyGroups defined for any of several reasons (different locales, number of subnets, etc.). The CIM_PolicyRules are therefore contained at n levels from the US Employee Logon PolicyGroup. Compare this to the Midwest CIM_PolicyGroup (CIM_PolicyGroup B), which might directly contain instances of CIM_PolicyRule.

Stand-alone policies are those that can be expressed in a simple statement. They can be represented effectively in schemata, CIM-XML or MIBs. Examples of this are VLAN assignments, simple YES/NO QoS requests, and IP address allocations. A specific design goal of this model is to support both stand-alone and aggregated policies.

Policy groups and rules can be classified by their purpose and intent. This classification is useful in querying or grouping policy rules, and indicates whether the policy is used to motivate when or how an action occurs, or to characterize services (that can then be used, for example, to bind clients to network services). The following list describes the classifications, which are enumerated for this purpose, in this model:

a. Motivational Policies: These are solely targeted at whether or how a policy's goal is accomplished. (Configuration and Usage Policies are specific kinds of Motivational Policies.) An example is the scheduling of file backup based on disk write activity from 8am to 3pm, M-F.
b. Configuration Policies: These define the default (or generic) setup of a managed entity (for example, a network service). Examples of Configuration Policies include the setup of a network forwarding service or a network-hosted print queue.
c. Installation Policies: These define what can and cannot be put on a system or component, as well as the configuration of the mechanisms that perform the install. Installation policies typically represent specific administrative permissions, and can also represent dependencies between different components (e.g., to complete the installation of component A, components B and C must have been previously successfully installed or uninstalled).
d. Error and Event Policies: These define, for example, what action should be taken if a device fails between 8am and 9pm, such as: call the system administrator, otherwise call the Help Desk.
e. Usage Policies: These control the selection and configuration of entities based on specific "usage" data. Configuration Policies can be modified or simply re-applied by Usage Policies. Examples of Usage Policies include upgrading network forwarding services after a user is verified to be a member of a "gold" service group, or reconfiguring a printer to be able to handle the next job in its queue.
f. Security Policies: These deal with verifying that the client is actually who the client purports to be, permitting or denying access to resources, selecting and applying appropriate authentication mechanisms, and performing accounting and auditing of resources.
g. Service Policies: These characterize network and other services (but not the use of them). For example, a policy could specify that all wide-area backbone interfaces shall use a specific type of queuing. Service policies describe services available in the network. Usage Policies, on the other hand, could describe the particular binding of a client of the network to services available in the network.

The above categories are represented in the CIM Policy Model, by special values defined for the PolicyKeywords property of the abstract class CIM_Policy.

2.3 Conceptual Areas Addressed by the Model

Policies represent and serve business goals and objectives. A translation must be made between these goals and objectives and their realization in a collection of systems and devices, such as a network. An example, of the expression of such business level goals and objectives, is a Service Level Agreement (SLA), including the objectives and metrics (Service Level Objectives, or SLOs) that are used to specify services that the network will provide for a given client. The SLA will usually be written in high-level business terminology. SLOs address more specific metrics in support of the SLA. These high-level descriptions of network or complex system services and metrics must be translated into lower-level, but also vendor- and device-independent specifications. The CIM Policy Model classes are intended to serve as the foundation for these vendor- and device-independent specifications.

It is envisioned that the definition of the CIM Policy Model in this draft is generic in nature and is applicable to Quality of Service (QoS), as well as to non-QoS networking applications (e.g., DHCP and IPSEC), and to non-networking applications, such as storage backup policies, auditing access, etc.

2.4 Policy Languages

The design of the Policy Core Information Model is influenced by a declarative, not procedural, approach to defining policy rules. More formally, a declarative language describes relationships between variables in terms of functions or inference rules, to which the interpreter or compiler can apply a fixed algorithm in order to produce a result. An imperative (or procedural) language, on the other hand, specifies an explicit sequence of steps to follow in order to produce a result.

It is important to note that this information model does not rule out the use of procedural languages. It recognizes that both declarative as well as procedural languages can be used to implement policy. However, the information model itself is better viewed as being declarative, because the sequence of steps for doing the processing of declarative statements tends to be left to the implementer. On the other hand, we have provided the option of expressing the desired order of action execution in this policy information model, and for expressing whether the order is mandatory or not. In addition, rather than trying to define algorithms or sets of instructions or steps that must be followed by a policy rule, we instead define a set of modular building blocks and relationships that can be used in a declarative or procedural fashion to implement policies.

Compare this to a strictly procedural model. Taking such an approach would require that we specify the condition testing sequence, and the action execution sequence, in any policy element which instantiates this information model, such as in a policy repository. This would, indeed, constrain the implementer. This is why the Policy Model is characterized as a declarative one. That is, the information model defines a set of properties, and a set of entities that contain these properties. However, it does NOT define either the algorithm to produce a result using the properties or an explicit sequence of steps to produce a result.

There are several design considerations and trade-offs to make in this respect.

a. On the one hand, we would like a policy definition language to be reasonably human-friendly for ease of definitions and diagnostics. On the other hand, given the diversity of devices (in terms of their processing capabilities) that could support the function of interpreting policy, we would like to keep the language somewhat machine-friendly. That is, it should be relatively simple to automate the parsing and processing of the language in systems and devices, such as network elements or servers. The approach taken is to provide a set of classes and properties that can be combined in either a declarative or procedural implementation that manages such things as network elements, services, or servers. The key point is to avoid trying to standardize rules or sets of steps to be followed in defining a policy. These must be left up to an implementation. Interoperability is achieved by standardizing the building blocks that are used to represent policy data and information.

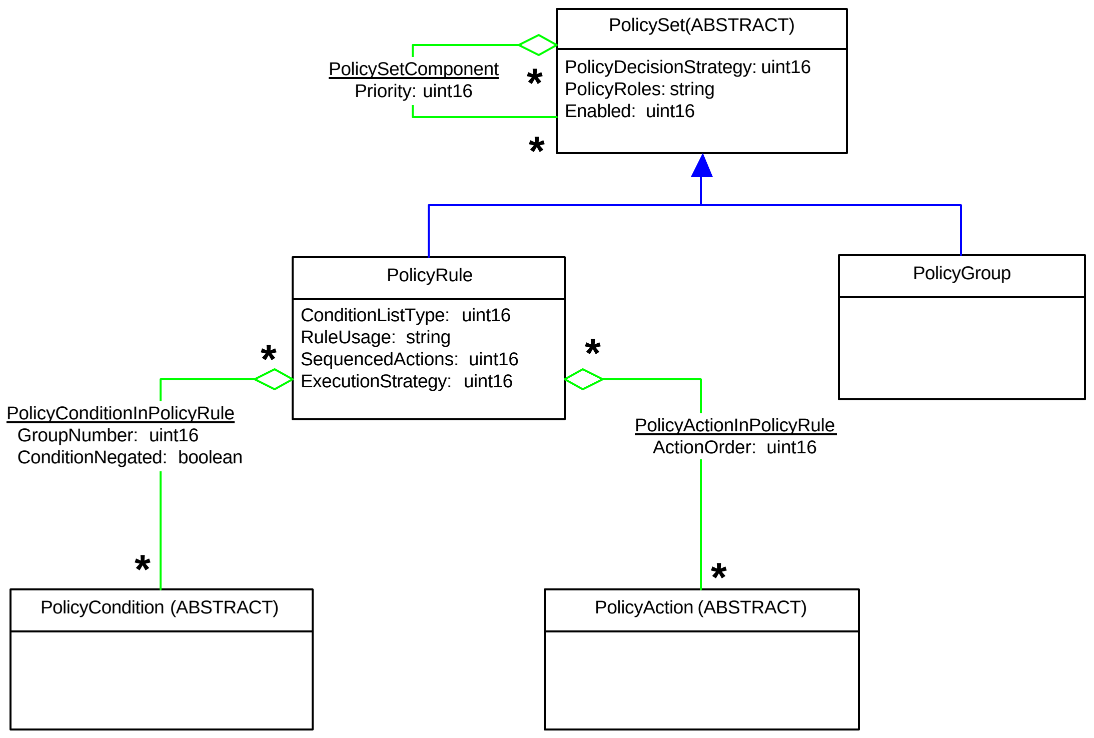
b. It is important to control the complexity of the specification, trading off richness of expression of data in the Policy Model, for ease of implementation and use. It is important to acknowledge the collective lack of experience in the field, with policies that control and manage network and system services and; hence, avoid the temptation of aiming for "completeness". We should instead strive to facilitate definition of a set of common policy data representations, that customers require today (e.g., VPN and QoS), and allow migration paths towards supporting more complex policies as customer needs and our understanding of the effect of these policy mechanisms evolve with experience. Specifically, in the context of the declarative style language discussed above, it is important to avoid having full blown predicate calculus as the language, because it would render many important problems such as consistency checking and policy decision point algorithms intractable. It is useful to consider a reasonably constrained language from these perspectives.

The CIM Policy Model strikes a balance between complexity and lack of power by using the well understood logical concepts of Disjunctive Normal Form and Conjunctive Normal Form for combining simple policy conditions into more complex ones.

3 Overview of the Policy Model

3.1 Classes and Associations, Overall Structure

The following diagram provides an overview of the classes that comprise the CIM Policy Model, their associations to each other, and their associations to other classes in the overall CIM Schema. The majority of this model is documented in the IETF RFCs 3060 and 3460, and in the Policy MOF itself. This information will not be repeated here and the reader is referred to these documents for a complete discussion of the policy classes and properties.



3.2 Selected Details of the Model

3.2.1 Reusable versus Rule-Specific Conditions and Actions

Policy conditions and policy actions can be partitioned into two groups: those associated with a single policy rule, and those that are reusable, in the sense that they may be associated with more than one policy rule. Conditions and actions in the first group are termed "rule-specific" conditions and actions; those in the second group are characterized as "reusable".

It is important to understand that the difference between a rule-specific condition or action and a reusable one is based on the intent of the policy administrator for the condition or action, rather than on the current associations in which the condition or action participates. Thus a reusable condition or action (that is, one that a policy administrator has created to be reusable) may at some point in time be associated with exactly one policy rule, without thereby becoming rule-specific.

There is no inherent difference between a rule-specific condition or action and a reusable one. There are, however, differences in how they are treated in a policy element which instantiates this information model, such as a policy repository. For example, it's natural to make the access permissions for a rule-specific condition or action identical to those for the rule itself. It's also natural for a rule-specific condition or action to be removed from the policy repository at the same time the rule is. With reusable conditions and actions, on the other hand, access permissions and existence criteria must be expressible without reference to a policy rule.

The preceding paragraph does not contain an exhaustive list of the ways in which reusable and rule-specific conditions should be treated differently. Its purpose is merely to justify making a semantic distinction between rule-specific and reusable, and then reflecting this distinction in the policy repository / reusable policy container itself.

Another issue is highlighted by reusable and rule-specific policy conditions and actions: the lack of a capability in CIM for expressing complex constraints involving multiple associations. Taking CIM_PolicyCondition as an example, there are two aggregations to examine. CIM_PolicyConditionInPolicyRule has the cardinality * at both ends, and CIM_ReusablePolicy has the cardinality * at the CIM_PolicyCondition end, and [0..1] at the CIM_ReusablePolicyContainer end.

Globally, these cardinalities are correct. However, there's more to the story, which only becomes clear if we examine the cardinalities separately for the two cases of a rule-specific CIM_PolicyCondition and a reusable one.

For a rule-specific CIM_PolicyCondition, the cardinality of CIM_PolicyConditionInPolicyRule at the CIM_PolicyRule end is [1..1], rather than [0..n] (recall that * is an abbreviation for [0..n]), since the condition is unique to one policy rule. And the cardinality of CIM_ReusablePolicy at the CIM_ReusablePolicyContainer end is [0..0]. This is reasonable, since these are both subsets of the specified cardinalities.

For a reusable CIM_PolicyCondition, however, the cardinality of CIM_ReusablePolicy at the CIM_ReusablePolicyContainer end is [1..1], and that of the CIM_PolicyConditionInPolicyRule at the CIM_PolicyRule end is [0..n]. This last point is important: a reusable CIM_PolicyCondition may be associated with 0, 1, or more than 1 CIM_PolicyRules, via exactly the same association CIM_PolicyConditionInPolicyRule that supports manual propagation of key values (from a single CIM_PolicyRule) in the case of a rule-specific CIM_PolicyCondition. But the reusable CIM_PolicyCondition gets its key values via a different association, CIM_ReusablePolicy.

Currently the only way to document constraints of this type in CIM is textually.

3.2.2 Roles

The idea behind roles is a simple one. In order to implement and update resource behavior to conform to policies, an administrator would face configuring, and then later reconfiguring, hundreds or thousands (or more) of resources in a collection of cooperating systems or devices, such as a network. Instead, with the introduction of the policy-based approach, a policy administrator assigns each resource to one or more roles, and then specifies the policies for each of these roles.

When applying this model in the context of the IETF's current policy framework, policy elements, such as policy decision points, then interpret the policy, on behalf of specific types of resources, (ie. on behalf of the systems/devices which directly provide or control those resources.) Such policy decision points are then responsible for configuring each of the resources associated with a role, in such a way that their behavior is in accordance with the policies specified for that role. When collective (e.g. network) behavior must be changed, the policy administrator can perform a single update to a policy for a role, and the elements noted above will ensure that the necessary configuration updates are performed on all the resources playing that role.

A more formal definition of a role is as follows:

> A role is a type of property that is used to select one or more policies for a set of entities and/or components from among a much larger set of available policies.

It is important to note that a role is more than a property. A role identifies a particular function of an entity or component that can be used to specify particular behavior associated with that entity or component. This difference is critical, and is most easily understood by thinking of a role as a selector. When used in this manner, one role selects a different set of policies than a different role does.

Roles are especially useful in selecting which policies are applicable to a particular set of entities or components when the policy repository can store thousands, or hundreds of thousands of policies. This use emphasizes the ability of the role to select the small subset of policies that are applicable from a huge set of policies that are available.

The PolicyRoles Property: PolicyRoles is a property associated with a policy rule or policy group. Using this property, it is possible to mark a policy rule or group as applying to a specific role, for example, to a Frame Relay interface or to a backbone ATM interface. All contained PolicyRule or PolicyGroup instances inherit the values of the PolicyRoles of the aggregating PolicyGroup, but the values are not explicitly copied. A contained instance may, however, add additional PolicyRoles to those it inherits from its aggregating PolicyGroup(s). Although not officially designated as 'role combinations', multiple roles may be specified using the form: \n"
  <RoleName>[&&<RoleName>]*
where the individual role names appear in alphabetical order (according to the collating sequence for UCS-2).

The policy administrator specifies the role names used in an environment.

3.2.3 Naming in the CIM Policy Model

While the CommonName property is present in the abstract superclass Policy, and is thus available in all of its instantiatable subclasses, the Policy Core Information Model does not use this property for naming instances. The following sections discuss how naming is handled in each of the instantiatable classes in the CIM Policy Model:

Role of the CreationClassName Property in Naming: To provide for more flexibility in instance naming, we make use the property CreationClassName. CreationClassName provides another dimension that can be used to avoid naming collisions, in the specific case of instances belonging to two different subclasses of a common superclass.

For example, suppose we have instances of two different subclasses of CIM_PolicyCondition: CIM_FrameRelayPolicyCondition and CIM_BgpPolicyCondition, and that these instances apply to the same context. If we had only the single key property PolicyConditionName available for distinguishing the two instances, then a collision would result from naming both of the instances with the key value PCName = "PC-1". Thus policy administrators from widely different disciplines would have to coordinate their naming of CIM_PolicyConditions for this context.

With CreationClassName, collisions of this type can be eliminated, without requiring coordination among the policy administrators. The two instances can be distinguished by giving their CreationClassNames different values. One instance is now identified with the two keys CreationClassName = "CIM_FrameRelayPolicyCondition" + PCName = "PC-1", while the other is identified with CreationClassName = "CIM_BgpPolicyCondition" + PCName = “PC-1".

CreationClassName cannot always provide the naming flexibility illustrated by this example. An implementation may elect to return, as the value of CreationClassName, the name of the instantiatable class HIGHEST in the inheritance hierarchy for an object, rather than the name of the most refined class. In the example, such an implementation would use “CIM_PolicyCondition" as the value for CreationClassName in both the Frame Relay policy condition and the BGP policy condition. These two policy condition objects would thus have to return different values for their other key property PolicyConditionName in order to be uniquely identifiable.

Each of the instantiatable classes in the CIM Policy Model includes the CreationClassName property as a key, in addition to its own class-specific key property.

Naming Instances of CIM_PolicyGroup and CIM_PolicyRule: A policy group always exists in some context. In the CIM Policy Model, this contextual character of a policy group is captured by the weak association (aka. weak aggregation), CIM_PolicyGroupInSystem, between an instance of CIM_PolicyGroup and an instance of CIM_System.

A policy rule must also exist in some context. This contextual character of a policy rule is captured by the weak association CIM_PolicyRuleInSystem, between a CIM_PolicyRule and a CIM_System.

Naming Instances of PolicyCondition and Its Subclasses: As indicated above, the single class CIM_PolicyCondition is used to represent both reusable and rule-specific policy conditions. The distinction between the two types of policy conditions lies in the associations that different instances of CIM_PolicyCondition participate in, and in how the different instances are named. Conceptually, a reusable policy condition resides in a policy repository – known in the model as a CIM_ReusablePolicyContainer - and is named within the scope of that repository. On the other hand, a rule-specific policy condition is, as the name suggests, named within the scope of the single policy rule to which it is related.

Naming instances of CIM_PolicyAction and its subclasses: From the point of view of naming, the CIM_PolicyAction class and its subclasses work exactly like the CIM_PolicyCondition class and its subclasses. See above for details.

Naming instances of CIM_ReusablePolicyContainer: Instances of CIM_ReusablePolicyContainer are named directly in the global CIM name space, using the CreationClassName and Name properties that the class inherits from CIM_System.

3.2.4 Local Time and UTC Time in CIM_PolicyTimePeriodConditions:

An instance of CIM_PolicyTimePeriodCondition has up to five properties that represent times: TimePeriod, MonthOfYearMask, DayOfMonthMask, DayOfWeekMask, and TimeOfDayMask. All of the time-related properties in an instance of CIM_PolicyTimePeriodCondition represent one of two types of times: local time at the place where a policy rule is applied, or UTC time. The property LocalOrUtcTime indicates which time representation applies to an instance of CIM_PolicyTimePeriodCondition.

Since the CIM Policy Model provides only for local time and UTC time, any management application which instantiates or modifies instances of this class, must provide for other useful time representations, such as a fixed time at a particular location. It will need to map from such administratively useful representations of time, to either local time or UTC time. An example will illustrate the nature of this mapping.

Suppose a policy rule, applied simultaneously across multiple worldwide time zones, is tied to the hours of operation for a Help Desk situated on the east coast of the United States: 0800 to 2000 Monday through Friday [US] Eastern Time. In order to express these times in CIM_PolicyTimePeriodCondition, a management application must convert them to UTC times. (They are not local times, because they refer to a single time interval worldwide, not to intervals tied to the local clocks at the locations where the CIM_PolicyRule is being applied.) As [RFC2445] points out, mapping from [US] Eastern Time to UTC time is not simply a matter of applying an offset: the offset between [US] Eastern Time and UTC time switches between -0500 and -0400 depending on whether Daylight Savings Time is in effect in the US.

Suppose the policy administrator's goal is to have a policy rule be valid from 0800 until 1200 [US] Eastern Time on every Monday, within the overall time period from the beginning of 2000 until the end of 2001. The management application could either be configured with the definition of what [US] Eastern Time means, or it could be configured with knowledge of where to go to get this information. RFC2445 contains further discussion of time zone definitions and where they might reside.

Armed with knowledge about [US] Eastern Time, the management app would create however many instances of CIM_PolicyTimePeriodCondition it needed to represent the desired intervals. Note that while there is an "explosion" of CIM_PolicyTimePeriodCondition instances, there is still just one CIM_PolicyRule, which is tied to all the CIM_PolicyTimePeriodCondition instances via the aggregation CIM_PolicyRuleValidityPeriod. Here are the first two of these instances:

1. TimePeriod: 20000101T050000/20000402T070000
   DayOfWeekMask: { Monday }
   TimeOfDayMask: T130000/T170000
   LocalOrUtcTime: UTC

2. TimePeriod: 20000402T070000/20001029T070000
   DayOfWeekMask: { Monday }
   TimeOfDayMask: T120000/T160000
   LocalOrUtcTime: UTC

There would be three more similar instances, for winter 2000-2001, summer 2001, and winter 2001 up through December 31.

Had the example been chosen differently, there could have been even more instances of CIM_PolicyTimePeriodCondition. If, for example, the time interval had been from 0800 - 2200 [US] Eastern Time on Mondays, instance 1 above would have split into two instances: one with a UTC time interval of T130000/T240000 on Mondays, and another with a UTC time interval of T000000/T030000 on Tuesdays. So the end result would have been ten instances of CIM_PolicyTimePeriodCondition, not five.

By restricting CIM_PolicyTimePeriodCondition to local time and UTC time, the model places the difficult and expensive task of mapping from "human" time representations to machine-friendly ones in the management application. Another approach would have been to place in CIM_PolicyTimePeriodCondition a means of representing a named time zone, such as [US] Eastern Time. This, however, would have passed the difficult mapping responsibility down to the PDPs and PEPs. It is better to have a mapping such as the one described above done once in a Policy Management Tool, rather than having it done over and over in each of the PDPs (and possibly PEPs) that need to apply a CIM_PolicyRule.

3.2.5 CIM Data Types:

The following CIM data types are used in the class definitions that follow:

• uint8	unsigned 8-bit integer
• uint16	unsigned 16-bit integer
• boolean	Boolean
• string	UCS-2 string.

Strings in CIM are stored as UCS-2 characters, where each character is encoded in two octets. Thus string values may need to be converted when moving between an environment that uses these encodings, and one that uses a different string encoding. For example, in an LDAP-accessible directory, attributes of type DirectoryString are stored in UTF-8 format. [RFC2279] explains how to convert between these two formats.

When it is applied to a CIM string, a MaxLen value refers to the maximum number of characters in the string, rather than to the maximum number of octets.

In addition to the CIM data types listed above, the association classes use the following type:

• <classname> ref	strongly typed reference

Appendix A – Change History

Version 1.0 Version 1.1	May 1, 2000 May 12, 2000	Initial Draft Final Draft -Misc. editorial changes -Changed Time Zone Representation
Version 2.6.0	March 14, 2002	Add preface describing V2.6.0
Version 2.7.0	June 18, 2003	Updated for latest CRs

Appendix B – References

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", January 1998.

[RFC2445] Dawson, F., and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", November 1998.

[RFC3060] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, “Policy Core Information Model – Version 1 Specification”, dated February 2001.

[RFC3460] B. Moore (ed), "Policy Core Information Model (PCIM) – Extensions", dated January 2003.

[DEN] J. Strassner and S. Judd, “Directory-Enabled Networks”, version 3.005 (August 1998).