Network Working Group
Internet-draft
Category: Standards Track

J. Strassner
Cisco Systems
E. Ellesson
B. Moore
IBM Corporation
May 1999

Policy Framework Core Information Model
draft-ietf-policy-core-schema-03.txt
May 17, 1999 15:02

Status of this Memo

Copyright Notice

Abstract

This document takes as its starting point the object-oriented information model for representing policy information currently under development as part of the Common Information Model (CIM) activity in the Desktop Management Task Force (DMTF). This CIM model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and relationship classes that indicate how instances of the structural classes are related to each other. In general, both of these class hierarchies will need to be mapped to a particular data store.

This draft defines the mapping of these DMTF-defined CIM classes to a directory that uses LDAPv3 as its access protocol. When mapping to an LDAP schema, the structural classes can be mapped more or less directly. The relationship hierarchy, however, must be mapped to a

---

form suitable for directory implementation. Since this mapping of the relationship classes could be done in a number of different ways, there is the risk of non-interoperable implementations. To avoid this possibility, this document provides a single mapping that all implementations using an LDAP directory as their policy repository SHALL use.

Classes are also added to the LDAP schema to improve the performance of a client's interactions with an LDAP server when the client is retrieving large amounts of policy-related information. These classes exist only to optimize LDAP retrievals: there are no classes in the CIM model that correspond to them.

The LDAP schema described in this document consists of six very general classes: policy (an abstract class), policyGroup, policyRule, policyCondition, policyTimePeriodCondition, and policyAction. The schema also contains two less general classes: vendorPolicyCondition and vendorPolicyAction. To achieve the mapping of the CIM relationships, the schema contains two auxiliary classes: policyGroupContainmentAuxClass and policyRuleContainmentAuxClass. Finally, the schema includes two classes policySubtreesPtrAuxClass and policyElement for optimizing LDAP retrievals, and a structural class policyInstance for attaching auxiliary classes representing policy conditions and policy actions. In all, therefore, the schema contains 13 classes.

Within the context of this document, the term "Core [Policy] Schema" is used to refer to the LDAP class definitions it contains.

Table of Contents

1. Introduction

This document takes as its starting point the object-oriented information model for representing policy information currently under development as part of the Common Information Model (CIM) activity in the Desktop Management Task Force (DMTF). This CIM model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and relationship classes that indicate how instances of the structural classes are related to each other. In general, both of these class hierarchies will need to be mapped to a particular data store.

This draft defines the mapping of these DMTF-defined CIM classes to a directory that uses LDAPv3 as its access protocol. Two types of mappings are involved:

- For the structural classes in the CIM model, the mapping is basically one-for-one: CIM classes map to LDAP classes, CIM properties map to LDAP attributes.

- For the relationship classes in the CIM model, different mappings are possible. In this document the CIM relationship classes and their properties are mapped in three ways: to LDAP auxiliary classes, to attributes representing DN pointers, and to "composite" attributes representing DN pointers with additional data elements.

Implementations that use an LDAP directory as their policy repository SHALL use the LDAP policy schema defined in this document. The use of the CIM information model as the starting point enables the schema and the relationship class hierarchy to be extensible, such that

---

other types of policy repositories, such as relational databases, can also use this information.

These policy classes and their relationships are sufficiently generic to allow them to represent policies related to anything. However, it is expected that their initial application will be for representing policies related to QoS (DiffServ and IntServ) and to IPSec. Policy models for application-specific areas such as these may extend the Core Schema in several ways. The preferred way is to use the policyGroup, policyRule, and policyTimePeriodCondition classes directly, as a foundation for representing and communicating policy information. Then, specific subclasses derived from policyCondition

and policyAction can capture application-specific definitions of conditions and actions of policies. These subclasses of policyCondition and policyAction MUST be defined as auxiliary classes, so that they can be used to form both simple and complex policy rules, as described below in Section 5.3.

Two subclasses, vendorPolicyCondition and vendorPolicyAction, are also included in this document, to provide a standard escape mechanism for vendor-specific extensions to the Core Policy Schema.

This document fits into the overall framework for representing, deploying, and managing policies being developed by the Policy Framework Working Group. The initial work to define this framework is in reference [1]. More specifically, this document builds on the core policy classes first introduced in references [2] and [3]. It also draws on the work done for the Directory-enabled Networks (DEN) specification, reference [4]. Work on the DEN specification by the DEN Ad-Hoc Working Group itself has been completed. Further work to standardize the models contained in it will be the responsibility of selected working groups of the CIM effort in the Desktop Management Task Force (DMTF). Standardization of the core policy model is the responsibility of the SLA Policy working group.

This document is organized in the following manner:

- Section 2 provides a general overview of policies and how they are modeled.
- Section 3 takes a brief look at the DMTF's CIM policy classes and relationships. The complete CIM policy definitions are available on the DMTF's web site; see reference [9].
- The remainder of the document presents the mapping of the CIM policy classes and relationships into an LDAP schema.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, reference [5].

2. Modeling Policies

The classes comprising the Core Policy Schema are intended to serve

as an extensible class hierarchy (through specialization) for defining policy objects that enable application developers, network administrators, and policy administrators to represent policies of different types.

One way to think of a policy-controlled network is to 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.

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. Please note that it is explicitly NOT a goal of this specification to represent more complicated conditions (such as those that may be found in a procedural language) at this time.

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, a general QoS policy rule might specify 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, or aggregations of policy groups, but not both. 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.

Stand-alone policies are those that can be expressed in a simple statement. They can be represented effectively in schemata or MIBs. Examples of this are VLAN assignments, simple YES/NO QoS requests, and IP address allocations. A specific design goal of this schema 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. It 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). Describing each of these concepts in more detail,

- Motivational Policies are solely targeted at whether or how a policy's goal is accomplished. Configuration and Usage Policies are specific kinds of Motivational Policies. Another example is the scheduling of file backup based on disk write activity from 8am to 3pm, M-F.

- Configuration Policies define the default (or generic) setup of a managed entity (for example, a network service). Examples of Configuration Policies are the setup of a network forwarding service or a network-hosted print queue.

- Installation Policies 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 be previously successfully installed or uninstalled).

- Error and Event Policies. For example, if a device fails between

8am and 9pm, call the system administrator, else call the Help Desk.

- Usage Policies 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.

- Security Policies 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.

- Service Policies characterize network and other services (not use them). For example, all wide-area backbone interfaces shall use a specific type of queuing.

  Service policies describe services available in the network. Usage policies describe the particular binding of a client of the network to services available in the network.

These categories are represented in the Core Schema by special values defined for the policyKeywords attribute of the abstract class policy.

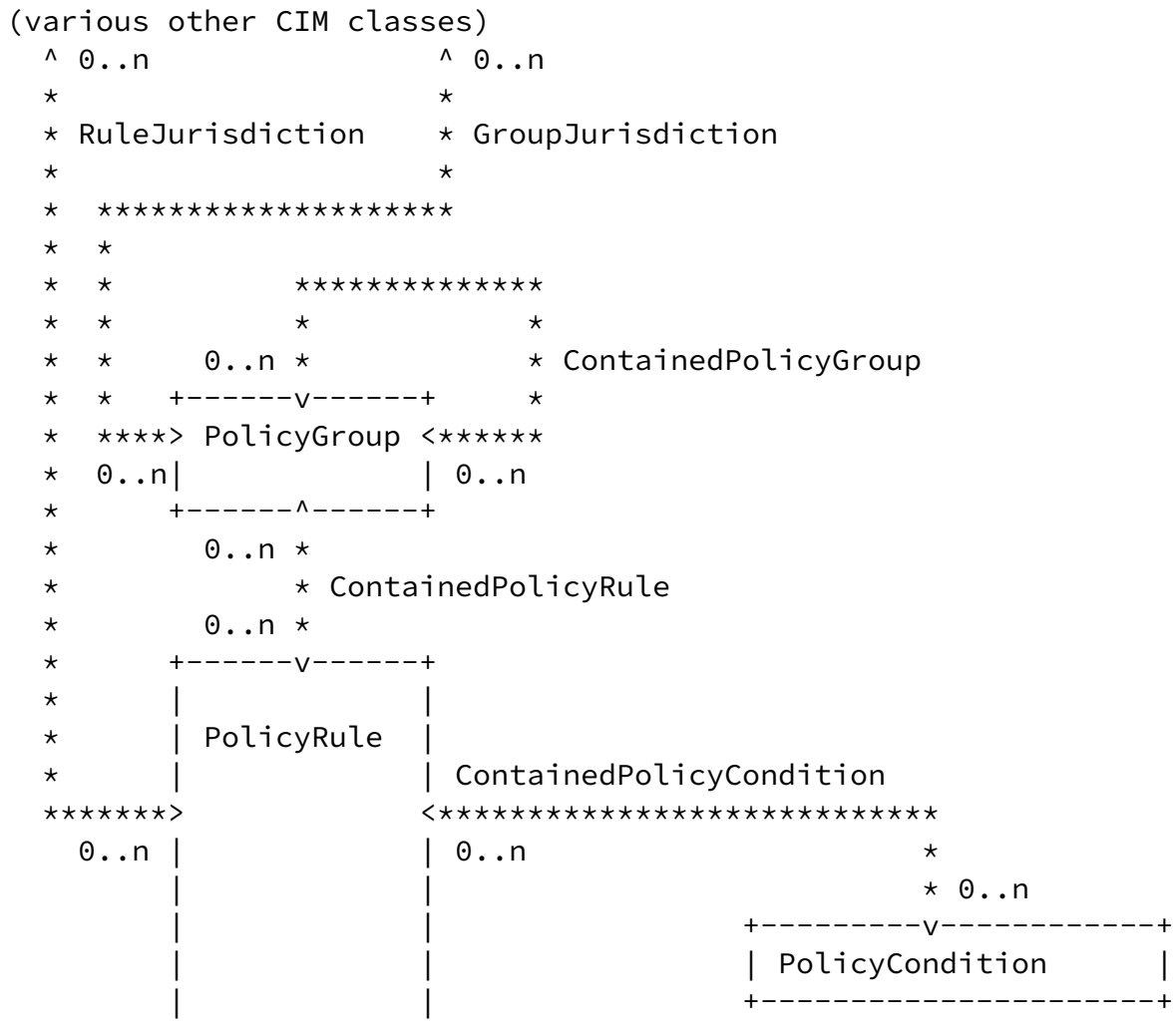
2.1. Policy Scope

Policies represent business goals and objectives. A translation must be made between these goals and objectives and their realization in the network. An example of this could be a Service Level Agreement (SLA), and its objectives and metrics (Service Level Objectives, or SLOs), that are used to specify services that the network will provide for a given client [8]. 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 services and metrics must be translated into lower-level, but also vendor- and device-independent specifications. The Core Schema classes are intended to serve as the foundation for these vendor- and device-independent specifications.

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

3. Overview of the Schema

The following diagram provides an overview of the five central classes comprising the CIM core policy schema, and their relationships to each other. Note that the abstract class Policy and the two extension classes VendorPolicyCondition and VendorPolicyAction are not shown.



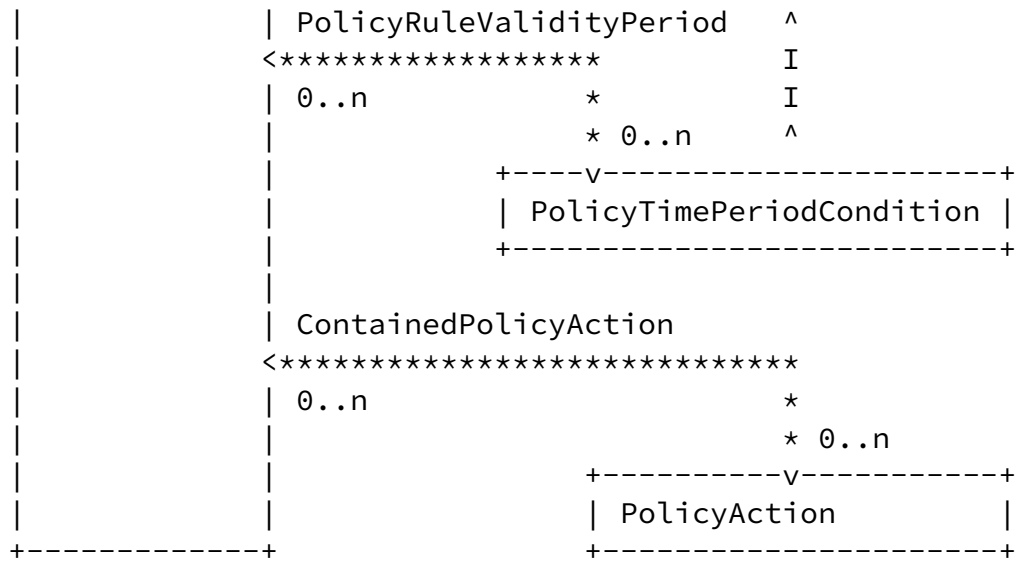


Figure 1. Overview of the CIM Policy Classes and Their Relationships

3.1. Relationships

Relationships are a central feature of information models. A relationship represents a physical or conceptual connection between

objects. CIM and DEN define the general concept of an association between two (or more) objects. Two types of associations are aggregations (which express whole-part relationships) and other relationships, such as those that express dependency. Both are used in this model.

3.2. Associations

An association is a class that contains two or more references, where each reference identifies another object. An association is defined using a class. Associations can be defined between classes without affecting any of the related classes. That is, addition of an association does not affect the interface of the related classes.

3.3. Aggregations

An aggregation is a strong form of an association. An aggregation is

usually used to represent a "whole-part" relationship. This type of relationship defines the containment relationship between a system and the components that make up the system. Aggregation often implies, but does not require, that the aggregated objects have mutual dependencies.

3.4. Key Relationships in the CIM Policy Model

The following relationships are shown in the preceding figure:

- The ContainedPolicyGroup relationship enables policy groups to be nested. This is critical for scalability and manageability, as it enables complex policies to be constructed from multiple simpler policies for administrative convenience. For example, a policy group representing policies for the US might have nested within it policy groups for the Eastern and Western US.

  In the LDAP schema, the ContainedPolicyGroup relationship is mapped to the policyGroupsAuxContainedSet attribute in the auxiliary class policyGroupContainmentAuxClass. (Other data stores may define a different mapping). This attribute enables a policyGroup to identify another policyGroup as its offspring.

- A policy group may aggregate one or more policy rules, via the ContainedPolicyRule relationship. Grouping of policy rules into a policy group is again for administrative convenience; a policy rule may also be used by itself, without belonging to a policy group. In the LDAP schema, the ContainedPolicyRule relationship is mapped to the policyRulesAuxContainedSet attribute in the auxiliary class policyRuleContainmentAuxClass.

- A policy group or policy rule may also be aggregated by an instance of any class to which the policyGroupContainmentAuxClass or policyRuleContainmentAuxClass class has been attached. Again, this is for administrative convenience. If the directory entry to which the policyGroupContainmentAuxClass or policyRuleContainmentAuxClass has been attached is a policy group, then the pointer in the auxiliary class realizes one of the relationships discussed above; a separate attribute is not needed in the policyGroup class. If the directory entry is something other than a policy group, then the pointer in the

auxiliary class realizes a Jurisdiction relationship from the CIM model.

- A policy rule aggregates zero or more instances of the PolicyCondition class, via the ContainedPolicyCondition association. A policy rule that aggregates zero policy conditions is not a valid rule -- it may, for example, be in the process of being entered into the policy repository. A policy rule has no effect until it is valid. The conditions aggregated by a policy rule are grouped into two levels of lists: either an ORed set of ANDed sets of conditions (DNF, the default) or an ANDed set of ORed sets of conditions (CNF). Individual conditions in these lists may be negated. The attribute PolicyRuleConditionListType specifies which of these two grouping schemes applies to a particular PolicyRule.

  Since conditions may be defined explicitly in a subclass of PolicyRule, the AND/OR mechanism to combine these conditions with other (associated) PolicyConditions MUST be specified by the PolicyRule's subclass.

  In either case, the conditions are used to determine whether to perform the actions associated with the PolicyRule.

- One or more policy time periods may be among the conditions associated with a policy rule via the ContainedPolicyCondition association. In this case, the time periods are simply additional conditions to be evaluated along with any other conditions specified for the rule.

- A different relationship between a policy rule and a policy time period is represented by the PolicyRuleValidityPeriod association: scheduled activation and deactivation of the policy rule. If a policy rule is associated with multiple policy time periods via this association, then the rule is active if at least one of the time periods indicates that it is active. (In other words, the time periods are ORed to determine whether the rule is active.) A policy time period may be aggregated by multiple policy rules. A rule that does not point to a policy time period via this association is, from the point of view of scheduling, always active. It may, however, be inactive for other reasons.

Time periods are a general concept that can be used in other

applications. However, they are mentioned explicitly here in this specification since they are frequently used in policy applications.

- A policy rule may aggregate zero or more policy actions. A policy rule that aggregates zero policy actions is not a valid rule -- it may, for example, be in the process of being entered into the policy repository. A policy rule has no effect until it is valid. The actions associated with a PolicyRule may be given a required order, a recommended order, or no order at all. For actions represented as separate objects, the ContainedPolicyAction aggregation can be used to express an order. For actions defined explicitly in a subclass of PolicyRule, the ordering mechanism must be specified in the subclass definition.

4. Inheritance Hierarchy for the LDAP Core Policy Schema

The following diagram illustrates the class hierarchy for the LDAP policy schema classes:

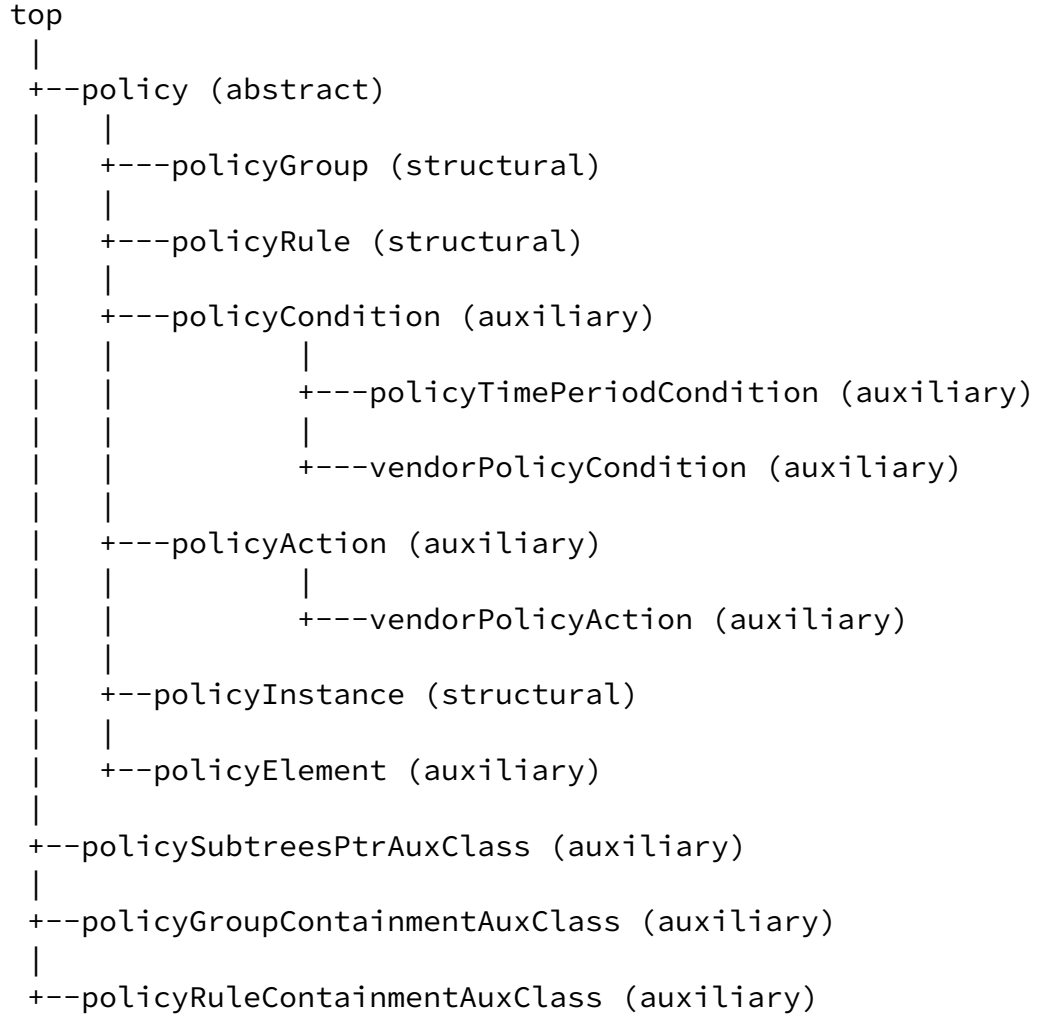


Figure 2. LDAP Class Inheritance Hierarchy for the Core Policy Schema

5. General Discussion of the CIM-to-LDAP Mapping

The classes described in Section 6 below contain certain optimizations for a directory that uses LDAP as its access protocol. One example of this is the use of auxiliary classes to represent CIM relationships. Other data stores might need to implement these relationships differently. A second example is the introduction of classes specifically designed to optimize retrieval of large amounts of policy-related data from a directory. This section discusses some

general topics related to the mapping from CIM to LDAP.

5.1. Summary of Class and Relationship Mappings

Eight of the classes in the LDAP Core Policy Schema come directly from corresponding CIM classes. Note that names of classes begin with an upper case character in CIM, but with a lower case character in LDAP.

CIM Class	LDAP Class
Policy	policy
PolicyGroup	policyGroup
PolicyRule	policyRule
PolicyCondition	policyCondition
PolicyAction	policyAction
VendorPolicyCondition	vendorPolicyCondition
VendorPolicyAction	vendorPolicyAction
PolicyTimePeriodCondition	policyTimePeriodCondition

Figure 3. Mapping of CIM Structural Classes to LDAP

The relationships in the CIM model map to pointer attributes in LDAP. Two of these attributes appear in auxiliary classes, which allows each of them to represent several CIM associations.

CIM Relationship	LDAP Attribute / Class
GroupJurisdiction	policyGroupsAuxContainedSet in policyGroupContainmentAuxClass
ContainedPolicyGroup	policyGroupsAuxContainedSet in

	policyGroupContainmentAuxClass
RuleJurisdiction	policyRulesAuxContainedSet in policyRuleContainmentAuxClass
ContainedPolicyRule	policyRulesAuxContainedSet in policyRuleContainmentAuxClass
ContainedPolicyCondition	policyRuleConditionList in policyRule
ContainedPolicyAction	policyRuleActionList in policyRule
PolicyRuleValidityPeriod	policyRuleValidityPeriodList in policyRule

Figure 4. Mapping of CIM Relationships to LDAP

The remaining classes in the LDAP Core Schema (policyInstance, policyElement, and policySubtreesPtrAuxClass) are all included to make navigation through the Directory Information Tree (DIT) and retrieval of the entries found there more efficient. This topic is discussed in Section 5.4 below.

5.2. Naming Attributes in the Core Schema

Instances in a directory are identified by distinguished names (DNs), which provide the same type of hierarchical organization that a file system provides in a computer system. A distinguished name is a sequence of relative distinguished names (RDNs), where an RDN provides a unique identifier for an instance within the context of its immediate superior, in the same way that a filename provides a unique identifier for a file within the context of the folder in which it resides.

To preserve maximum naming flexibility for policy administrators, each of the structural classes defined in this schema has its own naming attribute. (The structural class policyInstance gets its naming attribute from one of two auxiliary classes defined in the schema: either policyConditionName from the auxiliary class policyCondition, or policyActionName from the auxiliary class policyAction.) Since the naming attributes are different, a policy

administrator can, by using these attributes, guarantee that there will be no name collisions between instances of different classes, even if the same VALUE is assigned to the instances' respective naming attributes.

The X.500 attribute commonName (cn) is included as a MAY attribute in the abstract class policy, and thus by inheritance in policyGroup, policyRule, policyCondition, policyAction, policyInstance, policyElement, and all of their subclasses. In X.500 commonName typically functions as an RDN attribute, for naming instances of such classes as X.500's person. It has a different role in the Core Schema, however: to hold a short, human-friendly text string to be displayed on a user interface.

To avoid the interoperability problems that could arise if different implementations took different approaches to naming their policy-related instances, the following constraints apply:

- A management tool or other entity that creates policy-related instances SHOULD NOT use the commonName (cn) attribute for naming these instances, that is, SHOULD NOT use cn as the attribute for an instance's final RDN.

- A PDP or other entity that retrieves policy-related instances SHOULD NOT take any action that presupposes these instances are named with the commonName (cn) attribute.

5.3. Flexibility Gained through Auxiliary Classes

A key feature of the Core Schema is the use of auxiliary classes for modeling policy conditions and policy actions. These auxiliary classes make it possible to model a policy rule in two different ways:

- Simple Policy Rule: The conditions and/or the actions for the rule are attached to the rule object itself.

- Complex Policy Rule: The conditions and/or the actions for the rule are attached to instances of the structural class policyInstance, and these instances are pointed to by one of three attributes in the policy rule object - policyRuleConditionList, policyRuleActionList, or policyRuleValidityPeriodList (for the special case of a policyTimePeriodCondition object).

The simple/complex distinction for a policy rule is not all or nothing. A policy rule may have its conditions attached to itself and its actions attached to instances of policyInstance, or it may have its actions attached to itself and its conditions attached to instances of policyInstance. However, it SHALL NOT have either its conditions or its actions attached both to itself and to instances of policyInstance, with one exception: a policy rule may point to its validity periods with the policyRuleValidityPeriodList attribute, but have its other conditions attached to itself.

The tradeoffs between simple and complex policy rules are between the efficiency of simple rules and the flexibility and greater potential for reuse of complex rules. With a simple policy rule, the semantic options are limited:

- All conditions are ANDed together. This combination can be represented in two ways in the DNF / CNF expressions characteristic of policy conditions: as a DNF expression with a single AND group, or as a CNF expression with multiple single-condition OR groups. The first of these is arbitrarily chosen as the representation for the ANDed conditions in a simple policy rule.

- If multiple actions are included, no order can be specified for them.

Thus if a policy administrator needs to combine conditions in some other way, or if there is a set of actions that must be ordered, then the only option is to use a complex policy rule. The cost of a complex rule lies in the overhead of following DN pointers from the rule object to condition and/or action objects. Section 5.4 below describes a technique for minimizing this overhead, by making the following of these pointers a local operation for a PDP.

Note that whether a policy condition or policy action is represented as simple or complex has no effect on the relationship cardinality between the condition or action and the policy rule that aggregates it.

The classes policyCondition and policyAction do not themselves represent actual conditions and actions: these are introduced in

subclasses of policyCondition and policyAction. What policyCondition and policyAction do introduce, in addition to the naming attributes policyConditionName and policyActionName, are the semantics of being a policy condition or a policy action. These are the semantics that all the subclasses of policyCondition and policyAction inherit. Among these semantics are those of being an object to which, respectively, the policyRuleConditionList and policyRuleActionList attributes may point.

In order to preserve the flexibility of attaching either to policyRule or to policyInstance, all the subclasses of policyCondition and policyAction MUST also be auxiliary classes.

5.4. Location and Retrieval of Policy Objects in the Directory

When a PDP goes to an LDAP directory to retrieve the policy object instances relevant to the PEPs it serves, it is faced with two related problems:

- How does it locate and retrieve the directory entries that apply to its PEPs? These entries may include instances of the Core Schema classes, instances of domain-specific subclasses of these classes, and instances of other classes modeling such resources as user groups, interfaces, and address ranges.

- How does it retrieve the directory entries it needs in an efficient manner, so that retrieval of policy information from the directory does not become a roadblock to scaleability? There are two facets to this efficiency: retrieving only the relevant directory entries, and retrieving these entries using as few LDAP calls as possible.

The placement of objects in the Directory Information Tree (DIT) involves considerations other than how the policy-related objects will be retrieved by a PDP. Consequently, all that the Core Schema can do is to provide a "toolkit" of classes to assist the policy administrator as the DIT is being designed and built. A PDP must be able to take advantage of any tools that the policy administrator is able to build into the DIT, but also able to use a less efficient means of retrieval if that is all it has available to it.

The basic idea behind the LDAP optimization classes is a simple one: make it possible for a PDP to retrieve all the policy-related objects

it needs, and only those objects, using as few LDAP calls as possible. Figure 5 illustrates how these goals can be accomplished.

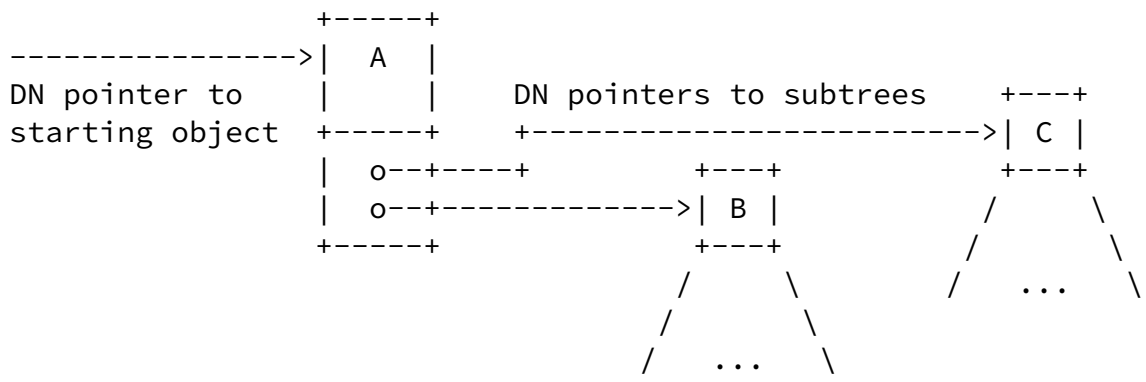


Figure 5. Using a policyContainer Object to Scope Policies

The PDP is configured initially with a DN pointer to some entry in the DIT. The structural class of this entry is not important; the PDP is interested only in the policySubtreesPtrAuxClass attached to it. This auxiliary class contains a multi-valued attribute with DN pointers to objects that anchor subtrees containing policy-related objects of interest to the PDP. Since policySubtreesPtrAuxClass is an auxiliary class, it can be attached to an entry that the PDP would need to access anyway - perhaps an entry containing initial configuration settings for the PDP, or for a PEP that uses the PDP.

Once it has retrieved the DN pointers, the PDP will direct to each of the objects identified by them an LDAP request that all entries in its subtree be evaluated against the selection criteria specified in the request. The LDAP-enabled directory then,returns all entries in that subtree that satisfy the specified criteria.

The selection criteria always specify that object class = "policy". Since all classes representing policy rules, policy conditions, and policy actions, both in the Core Schema and in any domain-specific schema derived from it, are subclasses of the abstract class policy, this criterion evaluates to TRUE for all instances of these classes. To accommodate special cases where a PDP needs to retrieve objects that are not inherently policy-related (for example, an IP address range object pointed to by a subclass of policyAction representing the DHCP action "assign from this address range), the auxiliary class policyElement can be used to "tag" an entry, so that it will be found by the selection criterion "object class = policy".

The approach described in the preceding paragraph will not work for certain directory implementations, because these implementations do not support matching of auxiliary classes in the objectClass attribute. For environments where these implementations are expected to be present, the "tagging" of entries as relevant to policy can be

accomplished by inserting the special value "POLICY" into the list of values contained in the policyKeywords attribute.

In a PDP needs only a subset of the policy-related objects in the indicated subtrees, then it can be configured with additional selection criteria based on the policyKeywords attribute defined in the policy class. This attribute supports both standardized and administrator-defined values. Thus a PDP could be configured to request only those policy-related objects containing the keywords "DHCP" and "Eastern US".

Returning to the example in Figure 5, we see that in the best case, a PDP can get all the policy-related objects it needs, and only these objects, with exactly three LDAP requests: one to its starting object to get the pointers to B and C, and then one each to B and C to get all the policy-related objects that pass the selection criteria with which it was configured. Once it has retrieved all of these objects, the PDP can then traverse their various DN pointers locally to understand the semantic relationships among them. The PDP must also be prepared to find a pointer to another subtree attached to any of the objects it retrieves, and to follow this pointer first, before it follows any of the semantically significant pointers it has received. This recursion permits a structured approach to identifying related policies. In Figure 5, for example, if the subtree under B includes departmental policies and the one under C includes divisional policies, then there might be a pointer from the subtree under C to an object D that roots the subtree of corporate-level policies.

Since a PDP has no guarantee that the entity that populates the directory won't use the policySubtreesPtrAuxClass, a PDP MUST understand this class, MUST be capable of retrieving and processing the entries in the subtrees it points to, and MUST be capable of doing all of this recursively. The same requirements apply to any other entity needing to retrieve policy information from the directory. Thus a Policy Management Tool that retrieves policy entries from the directory in order to perform validation and conflict detection MUST also understand and be capable of using the policySubtreesPtrAuxClass.

When it is serving as a tool for creating policy entries in the directory, a Policy Management Tool SHOULD support creation of policySubtreePrtAuxClass entries and their DN pointers.

5.4.1. Aliases

Additional flexibility in DIT structure is available to the policy administrator via LDAP aliasing. Figure 6 illustrates this flexibility.

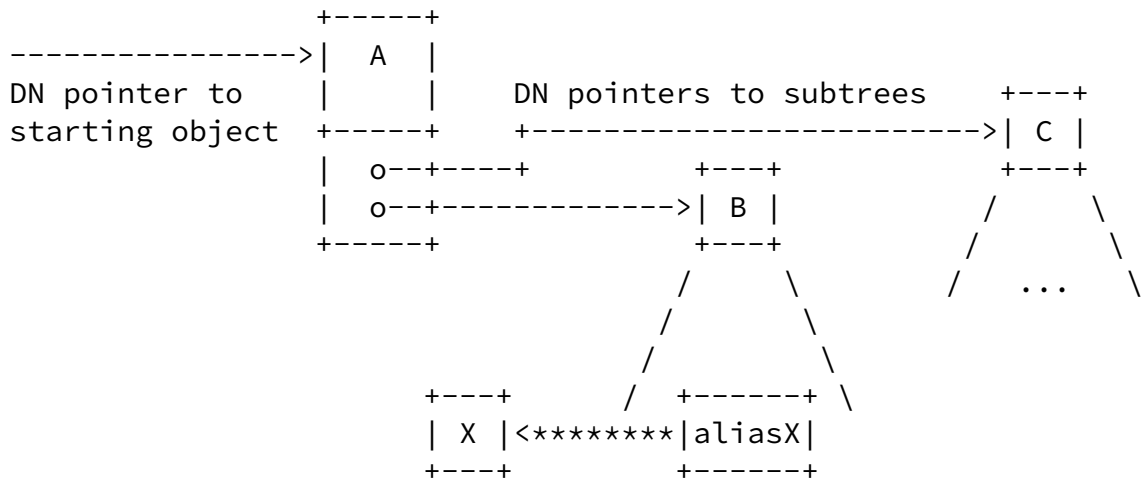


Figure 6. Addition of an Alias Object

Even if it is necessary to store a policy entry X in a directory location separate from the other policy entries, batch retrieval using policy subtrees can still be done. The administrator simply inserts into one of the subtrees of policy entries an alias entry aliasX pointing to the outlying entry X. When the PDP requests all entries in the subtree under B, a response will be returned for entry X, just as responses are returned for all the (non-alias) entries that actually are in the subtree.

Since resolution of an alias to its true entry is handled entirely by the LDAP directory, and is invisible to directory clients, PDPs need not do anything extra to support aliases. A Policy Management Tool MAY make available to a policy administrator the ability to create alias entries like the one in Figure 6.

6. Class Definitions

6.1. The Abstract Class policy

The abstract class policy collects four attributes that may be included in instances of any of the Core Policy classes (or their subclasses). The class value "policy" is also used as the mechanism for identifying policy-related instances in the Directory Information Tree. An instance of any class may be "tagged" with this class value by attaching to it the auxiliary class policyElement.

The class definition is as follows:

```
NAME              policy
DESCRIPTION      An abstract class with four attributes for
                  describing a policy-related instance.
DERIVED FROM     top
TYPE              abstract
AUXILIARY CLASSES   none
```

```
OID               <to be assigned>
MAY               cn caption description policyKeywords
```

6.1.1. The Attribute commonName (cn)

The cn, or commonName, attribute is an X.500 attribute. It specifies one or more user-friendly names (typically only one name) by which an object is commonly known, names that conform to the naming conventions of the country or culture with which the object is associated. A commonName may be ambiguous by itself, so it is typically used in a limited scope (such as an organization).

Unlike X.500, the Core Policy Schema specifies that the commonName attribute SHOULD NOT be used for forming the final RDN in the Distinguished Name of a policy-related instance.

```
NAME              cn
DESCRIPTION      A user-friendly name of a policy-related object.
SYNTAX            DirectoryString
OID               2.4.5.3
EQUALITY         caseIgnoreMatch
```

```
  MULTI-VALUED
```

6.1.2. The Attribute caption

This attribute corresponds to the Caption attribute defined in CIM. It provides a place for a one-line description of an entry.

```
  NAME             caption
  DESCRIPTION      A one-line description of this policy-related
                    object.
  SYNTAX            IA5String
  OID               <to be assigned>
  EQUALITY         caseExactIA5Match
  SINGLE-VALUED
```

6.1.3. The Attribute description

This attribute corresponds to the Description attribute defined in CIM. It provides a place for a longer description than that provided by the caption attribute.

```
  NAME             description
  DESCRIPTION      A long description of this policy-related object.
  SYNTAX            IA5String
  OID               <to be assigned>
  EQUALITY         caseExactIA5Match
  SINGLE-VALUED
```

6.1.4. The Attribute policyKeywords

This attribute provides a set of one or more keywords that a policy administrator may define to assist directory clients in locating the policy objects applicable to them. Keywords are of one of two types:

Keywords defined in this document, or in documents that define subclasses of the classes defined in this document. These keywords provide a vendor-independent, installation-independent way of identifying and locating policy objects.

Installation-dependent keywords for identifying and locating policy objects. Examples include "Engineering", "Billing", and "Review in December 1999".

This document defines the following keywords for identifying policy objects: "UNKNOWN", "CONFIGURATION", "USAGE", "SECURITY", "SERVICE", "MOTIVATIONAL", "INSTALLATION", and "EVENT". These concepts were defined in Section 2. In addition to these, the keyword "POLICY" is defined, to serve as a tag for policy-related entries when a directory implementation does not support matching on auxiliary classes in the objectClass attribute.

Documents that define subclasses of the Core Schema classes should define additional keywords to identify policy objects associated with instances of these subclasses. By convention, keywords defined in conjunction with class definitions are in uppercase. Installation-defined keywords can be in any case.

The attribute definition is as follows:

```
NAME             policyKeywords
DESCRIPTION     A set of keywords to assist directory clients in
                 locating the policy objects applicable to them.
SYNTAX           IA5String
OID              <to be assigned>
EQUALITY        caseExactIA5Match
MULTI-VALUED
```

6.2. The Class policyGroup

This class is a generalized aggregation container. It enables either policyRules or policyGroups, but not both, to be aggregated in a single container. Loops, including the degenerate case of a policyGroup that contains itself, are not allowed when policyGroups contain other policyGroups.

PolicyGroups and their nesting capabilities are shown in Figure 7 below. Note that a policyGroup can nest other policyGroups, and there is no restriction on the depth of the nesting in sibling policyGroups.

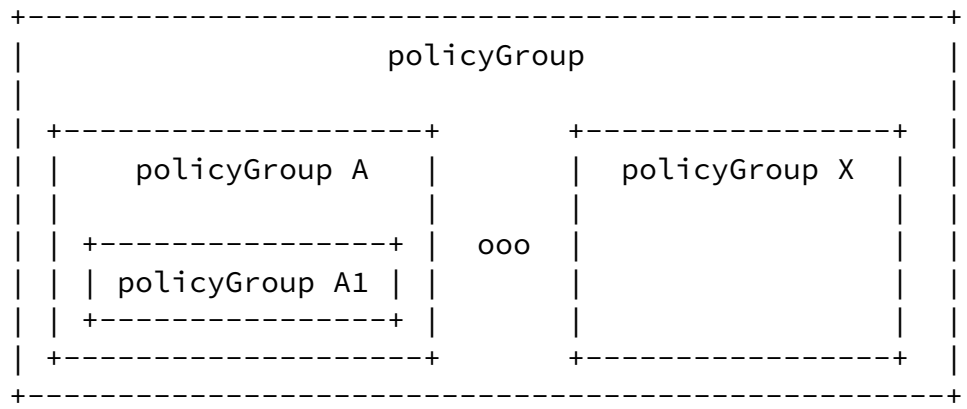


Figure 7. Overview of the policyGroup class

As a simple example, think of the highest level policyGroup shown in Figure 7 above as a logon policy for US employees of a company. This policyGroup may be called USEmployeeLogonPolicy, and may aggregate several policyGroups that provide specialized rules per location. Hence, policyGroup A in Figure 5 above may define logon rules for employees on the West Coast, while another policyGroup might define logon rules for the Midwest (e.g., policyGroup X), and so forth.

Note also that the depth of each policyGroup does not need to be the same. Thus, the WestCoast policyGroup might have several additional layers of policyGroups defined for any of several reasons (different locales, number of subnets, etc.). The policyRules are therefore contained at n levels from the USEmployeeLogonPolicyGroup. Compare this to the Midwest policyGroup (policyGroup X), which might directly contain policyRules.

The class definition for policyGroup is as follows. Note that this class definition does not include attributes to realize the ContainedPolicyRule and ContainedPolicyGroup associations from the object model, since a policyGroup object points to instances of policyGroup and policyRule via, respectively, the pointer in policyGroupContainmentAuxClass and the pointer in policyRuleContainmentAuxClass.

```
NAME              policyGroup
DESCRIPTION      A container for either a set of related
                  policyRules or a set of related policyGroups.
DERIVED FROM     top
TYPE              structural
AUXILIARY CLASSES   policyGroupContainmentAuxClass,
                  policyRuleContainmentAuxClass
POSSIBLE SUPERIORS  container, organization, organizationalUnit,
                  policyGroup
```

```
OID              <to be assigned>
MUST             policyGroupName
```

6.2.1. The Attribute policyGroupName

This attribute provides a user-friendly name for a policy group, and is normally what will be displayed to the end-user as the name of this class. It is defined as follows:

```
NAME             policyGroupName
DESCRIPTION      The user-friendly name of this policy group.
SYNTAX           IA5String
OID              <to be assigned>
EQUALITY         caseExactIA5Match
SINGLE-VALUED
```

6.3. The Class policyRule

This class represents the "If Condition then Action" semantics associated with a policy. A policyRule condition, in the most general sense, is represented as either an ORed set of ANDed conditions (Disjunctive Normal Form, or DNF) or an ANDed set of ORed conditions (Conjunctive Normal Form, or CNF). Individual conditions may either be negated (NOT C) or unnegated (C). The actions specified by a policyRule are to be performed if and only if the policyRule condition (whether it is represented in DNF or CNF) evaluates to TRUE.

The conditions and actions associated with a policy rule are modeled, respectively, with auxiliary subclasses of the classes policyCondition and policyAction. These auxiliary classes are attached either to an instance of policyRule itself, or to an instance of the structural class policyInstance identified by a DN pointer in an instance of policyRule.

As discussed above in section 3, a policy rule may also be associated with one or more policy time periods, indicating the schedule according to which the policy rule is active and inactive.

A policy rule is illustrated conceptually in Figure 8. below.

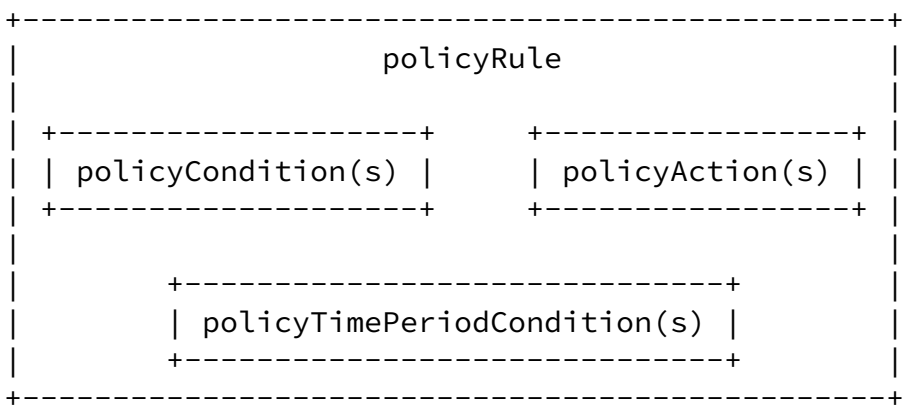


Figure 8. Overview of the policyRule Class

The policyRule class uses the attribute policyRuleConditionListType, to indicate whether the conditions for the rule are in DNF or CNF. The DN pointers from a policyRule to its associated policyConditions also contain an integer to partition the referenced conditions into one or more sets, and a plus ('+') or minus ('-') character to indicate whether the referenced condition is negated. An example shows how the attribute, the grouping integer, and the '+' / '-' provide a unique representation of a set of conditions in either DNF or CNF.

Suppose we have pointers to five policyConditions from an instance of policyRule, grouped as follows:

```
(C1: groupNumber = 1: unnegated,
 C2: groupNumber = 1: negated,
 C3: groupNumber = 1: unnegated,
 C4: groupNumber = 2: unnegated,
 C5: groupNumber = 2: unnegated)
```

If policyRuleConditionListType = DNF, then the overall condition for the policyRule is:

```
     (C1 AND (NOT C2) AND C3) OR (C4 AND C5)
```

On the other hand, if policyRuleConditionListType = CNF, then the overall condition for the policyRule is:

(C1 OR (NOT C2) OR C3) AND (C4 OR C5)

In both cases, there is an unambiguous specification of the overall condition that is tested to determine whether to perform the actions associated with the policyRule. This class also contains several attributes designed to help directory clients locate the policy rules applicable to them. The class definition is as follows:

```
NAME              policyRule
```



```
DESCRIPTION      The central class for representing the "If
                  Condition then Action" semantics associated with a
                  policy rule.
DERIVED FROM     top
TYPE              structural
AUXILIARY CLASSES   none
POSSIBLE SUPERIORS  policyGroup
OID               <to be assigned>
MUST              policyRuleName
MAY               policyRuleEnabled policyRuleConditionListType
                  policyRuleConditionList policyRuleActionList
                  policyRuleValidityPeriodList policyRuleUsage
                  policyRulePriority policyRuleMandatory
                  policyRuleSequencedActions
```

6.3.1. The Attribute policyRuleName

This attribute provides a user-friendly name for a policy rule. The attribute definition is as follows:

```
NAME              policyRuleName
DESCRIPTION      The user-friendly name of this policy rule.
SYNTAX            IA5String
OID               <to be assigned>
EQUALITY         caseExactIA5Match
SINGLE-VALUED
```

6.3.2. The Attribute policyRuleEnabled

This attribute indicates whether a policy rule is currently enabled, from an ADMINISTRATIVE point of view. Its purpose is to allow a policy administrator to enable or disable a policy rule without having to add it to, or remove it from, the directory.

The attribute also supports the value 'enabledForDebug'. When the attribute has this value, the Policy Decision Point is being told to evaluate the conditions for the policy rule, but not to perform the actions if the conditions evaluate to TRUE. This value serves as a debug vehicle when attempting to determine what policies would execute in a particular scenario, without taking any actions to change state during the debugging.

The attribute definition is as follows:

```
NAME             policyRuleEnabled
DESCRIPTION      A flag indicating whether this policy rule is
                  enabled from an administrative point of view.
SYNTAX           INTEGER
OID              <to be assigned>
EQUALITY         integerMatch
```

```
SINGLE-VALUED
DEFAULT VALUE    enabled(1)
```

The defined values for this attribute are enabled(1), disabled(2), and enabledForDebug(3).

6.3.3. The Attribute policyRuleConditionListType

This attribute is used to specify whether the list of policy conditions associated with this policy rule is in disjunctive normal form (DNF) or conjunctive normal form (CNF). If this attribute is not present, the list type defaults to DNF. The attribute definition is as follows:

```
NAME             policyRuleConditionListType
DESCRIPTION      Indicates whether the list of policy conditions
                  associated with this policy rule is in disjunctive
                  normal form (DNF) or conjunctive normal form
```

```
                    (CNF).  Defined values are 'DNF (1)' and 'CNF
                    (2)'.
  SYNTAX            INTEGER
  OID               <to be assigned>
  EQUALITY          integerMatch
  SINGLE-VALUED
  DEFAULT VALUE     1 (DNF)
```

6.3.4. The Attribute policyRuleConditionList

This attribute provides an unordered list of DN pointers that identify a set of policy conditions associated with this policy rule. There is an integer associated with each pointer, to provide the grouping of the conditions into first-level groups for the DNF or CNF representation of the overall policyRule condition. In addition, each pointer has associated with it a plus ('+') or minus ('-') to indicate whether the condition is negated: the '+' indicates that the condition is not negated, and the '-' indicates that it is negated. To accommodate this grouping, the syntax of this attribute is a string of the form 'groupNumber:+|-:DN'.

Existing matching rules are built to operate on a single data type. This attribute is conceptually composed of three data types, an Integer (groupNumber), an enumeration ('+' or '-'), and a DistinguishedName (DN). There are three ways to address this.

Collapse the three attribute types into a single structured DirectoryString with the format 'groupNumber:+|-:DN'. This approach has the advantage of not requiring any new support in the directory server implementations, since these servers already support a DirectoryString matching rule. Its disadvantage is that a

---

DirectoryString match works somewhat differently from a DN match with respect to subtleties such as preserving versus ignoring versus removing repeated whitespace characters. Thus DNs that would match with the distinguishedNameMatch matching rule might fail to be found as substrings of 'groupNumber:+|-:DN' strings by the DirectoryString matching rules, or vice versa.

Define a new syntax '<integer>:<+|->:<DN>', with its own matching rules. With this approach, the matching problems introduced by the DirectoryString could be avoided, since the new type would have its own matching rules. The disadvantage of defining a new type in this

way is that a directory server must add new code that recognizes the type and implements its matching rules. A directory server would thus be unable to support the Core Policy Schema until it had added this new code.

Use three objects in the directory to represent the three data types, and relate the objects with the Families of Entries model currently being discussed in the LDAP Extensions working group. This approach has the same problem as the previous one: without the addition of new code to support Families of Entries, a directory server would be unable to support the Core Policy Schema at all. There is also the additional complication here, that the Families of Entries model itself may take some time to reach approved status in the LDAP Extensions (LDAPEXT) working group.

For now, this document will move forward with the 'groupNumber:+|-:DN' structured DirectoryString approach for mapping ContainedPolicyCondition, as well as with an analogous 'n:DN' approach for mapping ContainedPolicyAction. To minimize problems arising from differences in matching rules, this document will provide a series of guidelines for constructing DNs that behave identically with respect to the DirectoryString matching rules and the distinguishedNameMatch. These guidelines are in Appendix A. Note that even if the DNs are chosen so that the matching rules behave the same, automatic processes such as "Modify RDN" that count on finding objects with the DistinguishedName syntax will not find attributes with the structured-string syntaxes.

The attribute definition is as follows:

```
NAME             policyRuleConditionList
DESCRIPTION     An unordered list of strings of the form
                 'groupNumber:+|-:DN', indicating a set of policy
                 conditions that determine when the policyRule is
                 applicable.
SYNTAX           DirectoryString
OID              <to be assigned>
EQUALITY        caseIgnoreSubstringsMatch
MULTI-VALUED
FORMAT           groupNumber:+|-:DN
```


6.3.5. The Attribute policyRuleActionList

This attribute provides an unordered list of strings of the form 'n:DN' that identify a set of policy actions associated with this policy rule. (See section 6.3.4 for a discussion of the issues surrounding the use of a syntax of this type.) When 'n' is a positive integer, it indicates a place in the sequence of actions to be performed, with smaller integers indicating earlier positions in the sequence. The special value '0' indicates "don't care". If two or more actions have the same non-zero sequence number, they may be performed in any order, but they must all be performed at the appropriate place in the overall action sequence.

A series of examples will make ordering of actions clearer:

- If all actions have the same sequence number, regardless of whether it is '0' or non-zero, any order is acceptable.

- The values

  1:DN-A
  2:DN-B
  1:DN-C
  3:DN-D

  indicate two acceptable orders: A,C,B,D or C,A,B,D, since A and C can be performed in either order, but only at the '1' position.

- The values

  0:DN-A
  2:DN-B
  3:DN-C
  3:DN-D

  require that B,C, and D occur either as B,C,D or as B,D,C. Action A may appear at any point relative to B,C, and D. Thus the complete set of acceptable orders is: A,B,C,D; B,A,C,D; B,C,A,D; B,C,D,A; A,B,D,C; B,A,D,C; B,D,A,C; B,D,C,A.

- Note that the non-zero sequence numbers need not start with '1', and they need not be consecutive. All that matters is their relative magnitude.

This attribute indicates the actions of a policyRule and their order (or absence of order). However, another attribute, policyRuleSequencedActions, indicates whether the indicated order is required, recommended, or not to be used at all.

All actions specified in the policyRuleActionList will be executed as long as the overall policy condition as defined by the

policyRuleConditionListType and policyRuleConditionList attributes evaluates to TRUE.

The attribute definition is as follows:

NAME	policyRuleActionList
DESCRIPTION	An unordered list of strings of the form 'n:DN', indicating an ordered set of policy actions to be performed if the associated condition(s) of the policyRule evaluates to true.
SYNTAX	DirectoryString
OID	<to be assigned>
EQUALITY	caseIgnoreSubstringsMatch
MULTI-VALUED	
FORMAT	n:DN

6.3.6. The Attribute policyRuleValidityPeriodList

This attribute provides an unordered set of DN pointers to one or more policyTimePeriodConditions, indicating when the policy rule is scheduled to be active and when it is scheduled to be inactive. The rule is scheduled to be active if it is active according to AT LEAST ONE of the policyTimePeriodConditions pointed to by this attribute.

The attribute definition is:

NAME	policyRuleValidityPeriodList
DESCRIPTION	Distinguished names of policyTimePeriodConditions that determine when the policyRule is scheduled to be active / inactive. No order is implied.
SYNTAX	DN
OID	<to be assigned>
EQUALITY	distinguishedNameMatch
MULTI-VALUED	

6.3.7. The Attribute policyRuleUsage

This attribute is a free-form string that recommends how this policy should be used. The attribute definition is as follows:

```
NAME              policyRuleUsage
DESCRIPTION      This attribute is used to provide guidelines on
                  how this policy should be used.
SYNTAX            DirectoryString
OID               <to be assigned>
EQUALITY         caseIgnoreMatch
SINGLE-VALUED
```


6.3.8. The Attribute policyRulePriority

This attribute provides a non-negative integer for prioritizing policy rules relative to each other. For policy rules that have this attribute, larger integer values indicate higher priority. Since one purpose of this attribute is to allow specific, ad hoc policy rules to temporarily override established policy rules, an instance that has this attribute set has a higher priority than all instances that lack it.

Prioritization among policy rules provides a simple and efficient mechanism for resolving policy conflicts.

The attribute definition is as follows:

```
NAME              policyRulePriority
DESCRIPTION      A non-negative integer for prioritizing this
                  policyRule relative to other policyRules.  A
                  larger value indicates a higher priority.
SYNTAX            INTEGER
OID               <to be assigned>
EQUALITY         integerMatch
SINGLE-VALUED
DEFAULT VALUE    0
```

6.3.9. The Attribute policyRuleMandatory

This attribute indicates whether evaluation (and possibly action execution) of a policyRule is mandatory or not. Its concept is similar to the ability to mark packets for delivery or possible discard, based on network traffic and device load.

The evaluation of a policyRule MUST be attempted if the

policyRuleMandatory attribute value is True. If the policyRuleMandatory attribute value of a policyRule is False, then the evaluation of the rule is "best effort" and MAY be ignored.

The attribute definition is as follows:

NAME	policyRuleMandatory
DESCRIPTION	A flag indicating that the evaluation of the policyConditions and execution of policyActions (if the condition list evaluates to True) is required.
SYNTAX	Boolean
OID	<to be assigned>
EQUALITY	booleanMatch
SINGLE-VALUED	
DEFAULT VALUE	TRUE

6.3.10. The Attribute policyRuleSequencedActions

This attribute gives a policy administrator a way of specifying how the ordering of the policy actions associated with this policyRule is to be interpreted. Three values are supported:

- mandatory (1): Do the actions in the indicated order, or don't do them at all.
- recommended (2): Do the actions in the indicated order if you can, but if you can't do them in this order, do them in another order if you can.
- dontCare (3): Do them -- I don't care about the order.

When error / event reporting is addressed for the Policy Framework, suitable codes will be defined for reporting that a set of actions could not be performed in an order specified as mandatory (and thus were not performed at all), that a set of actions could not be performed in a recommended order (and moreover could not be performed in any order), or that a set of actions could not be performed in a recommended order (but were performed in a different order). The attribute definition is as follows:

NAME	policyRuleSequencedActions

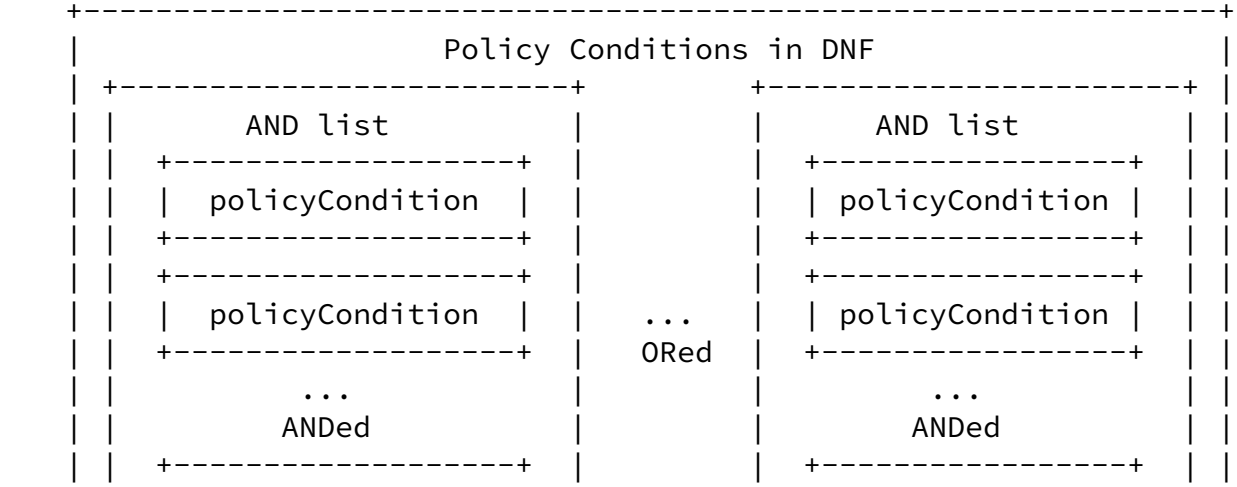
```
  DESCRIPTION     An enumeration indicating how to interpret the
                   action ordering indicated via the
                   policyRuleActionList attribute.
  SYNTAX           INTEGER
  OID              <to be assigned>
  EQUALITY        integerMatch
  SINGLE-VALUED
  DEFAULT VALUE   3
```

The defined values for this attribute are mandatory(1), recommended(2), and dontCare(3).

6.4. The Class policyCondition

The purpose of a policy condition is to determine whether or not the set of actions (contained in the policyRule that the condition applies to) should be executed or not. For the purposes of the Core Policy Schema, all that matters about an individual policyCondition is that it evaluates to TRUE or FALSE. (The individual policyConditions associated with a policyRule are combined to form a compound expression in either DNF or CNF, but this is accomplished via the groupNumber component of the policyRuleConditionList string and by the policyRuleConditionListType attribute, both of which are discussed above.) A logical structure WITHIN an individual

policyCondition may also be introduced, but this would have to be done in a subclass of policyCondition.



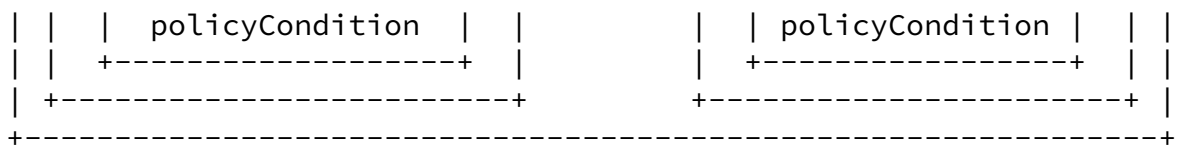


Figure 9. Overview of Policy Conditions in DNF

This figure illustrates that when policy conditions are in DNF, there are one or more sets of conditions that are ANDed together to form AND lists. An AND list evaluates to TRUE if and only if all of its constituent conditions evaluate to TRUE. The overall condition then evaluates to TRUE if and only if at least one of its constituent AND lists evaluates to TRUE.

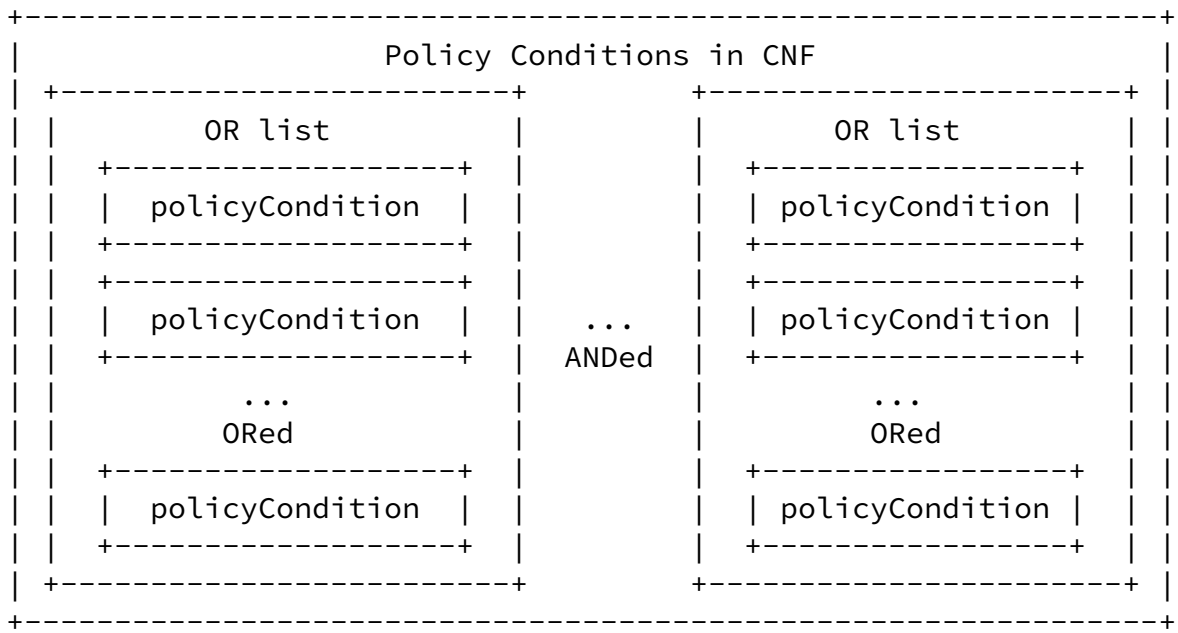


Figure 10. Overview of Policy Conditions in CNF

In this figure, the policy conditions are in CNF. Consequently, there are one or more OR lists, each of which evaluates to TRUE if and only if at least one of its constituent conditions evaluates to TRUE. The overall condition then evaluates to TRUE if and only if ALL of its constituent OR lists evaluate to TRUE. The class definition is as follows:

```
NAME              policyCondition
DESCRIPTION      A class representing a condition to be evaluated
                  in conjunction with a policy rule.
DERIVED FROM     top
TYPE              auxiliary
AUXILIARY CLASSES   none
POSSIBLE SUPERIORS  policyRule
OID               <to be assigned>
MUST              policyConditionName
MAY
```

6.4.1. The Attribute policyConditionName

This attribute provides a user-friendly name for a policy condition. The attribute definition is as follows:

```
NAME              policyConditionName
DESCRIPTION      The user-friendly name of this policy condition.
SYNTAX            IA5String
OID               <to be assigned>
EQUALITY         caseExactIA5Match
SINGLE-VALUED
```

6.5. The Class policyTimePeriodCondition

This class provides a means of representing the time periods during which a policy rule is valid, i.e., active. At all times that fall outside these time periods, the policy rule has no effect. A policy rule is treated as valid at all times if it does not specify a policyTimePeriodCondition.

In some cases a PDP may need to perform certain setup / cleanup actions when a policy rule becomes active / inactive. For example, sessions that were established while a policy rule was active might need to be taken down when the rule becomes inactive. In other cases, however, such sessions might be left up: in this case, the effect of deactivating the policy rule would just be to prevent the establishment of new sessions. Any such setup / cleanup behaviors on validity period transitions must be specified in a subclass of policyRule. If such behaviors need to be under the control of the policy administrator, then a mechanism to allow this control must also be specified in the subclass.

policyTimePeriodCondition is defined as a subclass of policyCondition. This is to allow the inclusion of time-based criteria in the AND/OR condition definitions for a policyRule.

Instances of this class may have up to five attributes identifying time periods at different levels. The values of all the attributes present in an instance are ANDed together to determine the validity period(s) for the instance. For example, an instance with an overall validity range of January 1, 1999 through December 31, 1999; a month mask of "001100000000" (March and April); a day-of-the-week mask of "0000100" (Fridays); and a time of day range of 0800 through 1600 would represent the following time periods:

```
    Friday, March  5, 1999, from 0800 through 1600;
    Friday, March 12, 1999, from 0800 through 1600;
    Friday, March 19, 1999, from 0800 through 1600;
    Friday, March 26, 1999, from 0800 through 1600;
    Friday, April  2, 1999, from 0800 through 1600;
    Friday, April  9, 1999, from 0800 through 1600;
    Friday, April 16, 1999, from 0800 through 1600;
    Friday, April 23, 1999, from 0800 through 1600;
    Friday, April 30, 1999, from 0800 through 1600.
```

Attributes not present in an instance of policyTimePeriodCondition are implicitly treated as having their value "always enabled". Thus, in the example above, the day-of-the-month mask is not present, and so the validity period for the instance implicitly includes a day-of-the-month mask containing 31 1's. If we apply this "missing attribute" rule to its fullest, we see that there is a second way to indicate that a policy rule is always enabled: have it point to an instance of policyTimePeriodCondition whose only attributes are its naming attributes.

The class definition is as follows. Note that instances of this class are named with the attributes cn and policyConditionName that they inherit from policyCondition.

```
  NAME              policyTimePeriodCondition
  DESCRIPTION      A class that provides the capability of enabling /
                    disabling a policy rule according to a pre-
                    determined schedule.
  DERIVED FROM     policyCondition
  TYPE              auxiliary
  AUXILIARY CLASSES   none
  POSSIBLE SUPERIORS  policyRule
  OID               <to be assigned>
  MUST
  MAY               ptpConditionTime ptpConditionMonthOfYearMask
```

6.5.1. The Attribute ptpConditionTime

This attribute identifies an overall range of calendar dates and times over which a policy rule is valid. It is formatted as a string consisting of a start date and time, then a colon (':'), and followed by an end date and time. The first date indicates the beginning of the range, while the second date indicates the end. Thus, the second date and time must be later than the first. Dates are expressed as substrings of the form "yyyymmddhhmmss". For example:

19990101080000:19990131120000

January 1, 1999, 0800 through January 31, 1999, noon

The attribute definition is as follows:

NAME	ptpConditionTime
DESCRIPTION	The range of calendar dates on which a policy rule is valid.
SYNTAX	PrintableString
OID	<to be assigned>
EQUALITY	caseIgnoreMatch
SINGLE-VALUED	
FORMAT	yyyymmddhhmmss:yyyymmddhhmmss

6.5.2. The Attribute ptpConditionMonthOfYearMask

The purpose of this attribute is to refine the definition of the valid time period that is defined by the ptpConditionTime attribute by explicitly specifying which months the policy is valid for. These attributes work together, with the ptpConditionTime used to specify the overall time period that the policy is valid for, and the ptpConditionMonthOfYearMask used to pick out which months of that time period the policy is valid for.

This attribute is formatted as a string containing 12 ASCII '0's and

'1's, where the '1's identify the months (beginning with January) in which the policy rule is valid. The value "000010010000", for example, indicates that a policy rule is valid only in the months May and August.

If this attribute is omitted, then the policy assumes that it is valid for all twelve months. The attribute definition is as follows:

```
NAME             ptpConditionMonthOfYearMask
DESCRIPTION      A mask identifying the months of the year in which
                  a policy rule is valid.
SYNTAX           Printable String
```

```
OID              <to be assigned>
EQUALITY         caseIgnoreMatch
SINGLE-VALUED
FORMAT           A string of 12 ASCII '0's and '1's.
```

6.5.3. The Attribute ptpConditionDayOfMonthMask

The purpose of this attribute is to refine the definition of the valid time period that is defined by the ptpConditionTime attribute by explicitly specifying which days of the month the policy is valid for. These attributes work together, with the ptpConditionTime used to specify the overall time period that the policy is valid for, and the ptpConditionDayOfMonthMask used to pick out which days of the month that time period the policy is valid for.

This attribute is formatted as a string containing 31 ASCII '0's and '1's, where the '1's identify the days of the month (beginning with day 1 and going up through day 31) on which the policy rule is valid. The value "1110000000000000000000000000000", for example, indicates that a policy rule is valid only on the first three days of each month. For months with fewer than 31 days, the digits corresponding to days that the months do not have are ignored. The attribute definition is as follows:

```
NAME             ptpConditionDayOfMonthMask
DESCRIPTION      A mask identifying the days of the month on which
                  a policy rule is valid.
SYNTAX           PrintableString
OID              <to be assigned>
EQUALITY         caseIgnoreMatch
```

```
SINGLE-VALUED
FORMAT          A string of 31 ASCII '0's and '1's.
```

6.5.4. The Attribute ptpConditionDayOfWeekMask

The purpose of this attribute is to refine the definition of the valid time period that is defined by the ptpConditionTime attribute by explicitly specifying which days of the week the policy is valid for. These attributes work together, with the ptpConditionTime used to specify the overall time period that the policy is valid for, and the ptpConditionDayOfWeekMask used to pick out which days of the week of that time period the policy is valid for.

This attribute is formatted as a string containing 7 ASCII '0's and '1's, where the '1's identify the days of the week (beginning with Monday and going up through Sunday) on which the policy rule is valid. The value "1111100", for example, indicates that a policy rule is valid Monday through Friday.

The attribute definition is as follows:

```
NAME             ptpConditionDayOfWeekMask
DESCRIPTION      A mask identifying the days of the week on which a
                  policy rule is valid.
SYNTAX           PrintableString
OID              <to be assigned>
EQUALITY         caseIgnoreMatch
SINGLE-VALUED
FORMAT           A string of 7 ASCII '0's and '1's.
```

6.5.5. The Attribute ptpConditionTimeOfDayMask

The purpose of this attribute is to refine the definition of the valid time period that is defined by the ptpConditionTime attribute by explicitly specifying a range of times in a day the policy is valid for. These attributes work together, with the ptpConditionTime used to specify the overall time period that the policy is valid for, and the ptpConditionTimeOfDayMask used to pick out which range of time periods in a given day of the week of that time period the policy is valid for.

This attribute is formatted as a string containing two times,

separated by a colon (':'). The first time indicates the beginning of the range, while the second time indicates the end. Times are expressed as substrings of the form "hhmmss".

The second substring always identifies a later time than the first substring. To allow for ranges that span midnight, however, the value of the second string may be smaller than the value of the first substring. Thus, "080000:210000" identifies the range from 0800 until 2100, while "210000:080000" identifies the range from 2100 until 0800 of the following day.

When a range spans midnight, it by definition includes parts of two successive days. When one of these days is also selected by either the ptpConditionMonthOfYearMask, ptpConditionDayOfMonthMask, and/or ptpConditionDayOfWeekMask, but the other day is not, then the policy is active only during the portion of the range that falls on the selected day. For example, if the range extends from 2100 until 0800, and the day of week mask selects Monday and Tuesday, then the policy is active during the following three intervals:

From midnight Sunday until 0800 Monday;
From 2100 Monday until 0800 Tuesday;
From 2100 Tuesday until 21:59:59 Tuesday.

The attribute definition is as follows:

NAME	ptpConditionTimeOfDayMask

DESCRIPTION	The range of times at which a policy rule is valid. If the second time is earlier than the first, then the interval spans midnight.
SYNTAX	Printable String
OID	<to be assigned>
EQUALITY	caseIgnoreMatch
SINGLE-VALUED	
FORMAT	hhmmss:hhmmss[:<UTC offset>]

6.5.6. The Attribute ptpConditionTimeZone

This attribute is used to explicitly define a time zone for use by

the ptpConditionTime and the various Mask attributes. If this attribute is NULL, then local time (at the location where the policyRule is enforced -- in other words, at the Policy Enforcement Point) is assumed.

This attribute specifies time in UTC, using an offset indicator. The UTC offset indicator is either a 'Z', indicating UTC, or a substring of the following form:

'+' or '-'	direction from UTC: '+' = east, '-' = west
hh	hours from UTC (00..13)
mm	minutes from UTC (00..59)

The attribute definition is as follows:

NAME	ptpConditionTimeZone
DESCRIPTION	The definition of the time zone for the policyTimePeriodCondition.
SYNTAX	PrintableString
OID	<to be assigned>
EQUALITY	caseIgnoreMatch
SINGLE-VALUED	
FORMAT	either 'Z' (UTC) or {'+' | '-'}'hhmm'

6.6. The Class vendorPolicyCondition

The purpose of this class is to provide a general escape mechanism for representing policy conditions that have not been modeled with specific attributes. Instead, the two attributes vendorPolicyConstraintData and vendorPolicyConstraintEncoding are used to define the content and format of the condition, as explained below.

As its name suggests, this class is intended for vendor-specific extensions to the Core Policy Schema. Standardized extensions are not expected to use this class.

The class definition is as follows:

```
NAME              vendorPolicyCondition
DESCRIPTION      A class that defines a registered means to
                  describe a policy condition.
DERIVED FROM     policyCondition
TYPE              auxiliary
AUXILIARY CLASSES   none
POSSIBLE SUPERIORS  policyRule
OID               <to be assigned>
MUST              vendorPolicyConstraintData
                  vendorPolicyConstraintEncoding
MAY
```

6.6.1. The Attribute vendorPolicyConstraintData

This attribute provides a general escape mechanism for representing policy conditions that have not been modeled with specific attributes. The format of the OctetString is left unspecified in this definition. It is determined by the OID value stored in the attribute vendorPolicyConstraintEncoding. Since vendorPolicyConstraintEncoding is single-valued, all the values of vendorPolicyConstraintData share the same format and semantics.

A policy decision point can readily determine whether it supports the values stored in an instance of vendorPolicyConstraintData by checking the OID value from vendorPolicyConstraintEncoding against the set of OIDs it recognizes. The action for the policy decision point to take in case it does not recognize the format of this data could itself be modeled as a policy rule, governing the behavior of the policy decision point.

The attribute definition is as follows:

```
NAME              vendorPolicyConstraintData
DESCRIPTION      Escape mechanism for representing constraints that
                  have not been modeled as specific attributes. The
                  format of the values is identified by the OID
                  stored in the attribute
                  vendorPolicyConstraintEncoding.
SYNTAX            OctetString
OID               <to be assigned>
EQUALITY         octetStringMatch
MULTI-VALUED
```

6.6.2. The Attribute vendorPolicyConstraintEncoding

This attribute identifies the encoding and semantics of the values of vendorPolicyConstraintData in this instance. The value of this attribute is a single OID.

The attribute definition is as follows:

```
NAME              vendorPolicyConstraintEncoding
DESCRIPTION      An OID identifying the format and semantics for
                  this instance's vendorPolicyConstraintData
                  attribute.
SYNTAX            OID
OID               <to be assigned>
EQUALITY         objectIdentifierMatch
SINGLE-VALUED
```

6.7. The Class policyAction

The purpose of a policy action is to execute one or more operations that will affect network traffic and/or systems, devices, etc. in order to achieve a desired policy state. This (new) policy state provides one or more (new) behaviors. A policy action ordinarily changes the configuration of one or more elements.

A policyRule contains one or more policy actions. Unlike a condition, however, only one list of policy actions is contained in a policyRule. A policy administrator can assign an order to the actions associated with a policyRule, complete with an indication of whether the indicated order is mandatory, recommended, or of no significance.

The actions associated with a policyRule are executed if and only if the overall condition(s) of the policyRule evaluates to TRUE.

The class definition is as follows:

```
NAME              policyAction
DESCRIPTION      A class representing an action to be performed as
                  a result of a policy rule.
DERIVED FROM     top
TYPE              auxiliary
AUXILIARY CLASSES   none
POSSIBLE SUPERIORS  policyRule
OID               <to be assigned>
MUST              policyActionName
MAY
```

6.7.1. The Attribute policyActionName

This attribute provides a user-friendly name for a policy action.

The attribute definition is as follows:

```
NAME              policyActionName
DESCRIPTION      The user-friendly name of this policy action.
SYNTAX            IA5String
OID               <to be assigned>
EQUALITY         caseExactIA5Match
SINGLE-VALUED
```

6.8. The Class vendorPolicyAction

The purpose of this class is to provide a general escape mechanism for representing policy actions that have not been modeled with specific attributes. Instead, the two attributes vendorPolicyActionData and vendorPolicyActionEncoding are used to define the content and format of the condition, as explained below.

As its name suggests, this class is intended for vendor-specific extensions to the Core Policy Schema. Standardized extensions are not expected to use this class.

The class definition is as follows:

```
NAME              vendorPolicyAction
DESCRIPTION      A class that defines a registered means to
                  describe a policy action.
DERIVED FROM     policyAction
TYPE              auxiliary
AUXILIARY CLASSES   none
POSSIBLE SUPERIORS  policyRule
OID               <to be assigned>
MUST              vendorPolicyActionData vendorPolicyActionEncoding
MAY
```

6.8.1. The Attribute vendorPolicyActionData

This attribute provides a general escape mechanism for representing policy actions that have not been modeled with specific attributes. The format of the OctetString is left unspecified in this definition. It is determined by the OID value stored in the attribute vendorPolicyActionEncoding. Since vendorPolicyActionEncoding is single-valued, all the values of vendorPolicyActionData share the same format and semantics.

A policy decision point can readily determine whether it supports the values stored in an instance of vendorPolicyActionData, by checking the OID value from vendorPolicyActionEncoding against the set of OIDs it recognizes. The action for the policy decision point to take in case it does not recognize the format of this data could itself be

modeled as a policy rule, governing the behavior of the policy decision point.

The attribute definition is as follows:

NAME	vendorPolicyActionData
DESCRIPTION	Escape mechanism for representing actions that have not been modeled as specific attributes. The format of the values is identified by the OID stored in the attribute vendorPolicyActionEncoding.
SYNTAX	OctetString
OID	<to be assigned>
EQUALITY	octetStringMatch
MULTI-VALUED	

6.8.2. The Attribute vendorPolicyActionEncoding

This attribute identifies the encoding and semantics of the values of vendorPolicyActionData in this instance. The value of this attribute is a single OID.

The attribute definition is as follows:

NAME	vendorPolicyActionEncoding
DESCRIPTION	An OID identifying the format and semantics for this instance's vendorPolicyActionData attribute.
SYNTAX	OID

```
    OID              <to be assigned>
    EQUALITY        objectIdentifierMatch
    SINGLE-VALUED
```

6.9. The Class policyInstance

This class introduces no additional attributes, beyond those defined in the class policy from which it is derived. Its role in the Core Schema is to serve as the structural class to which the auxiliary classes policyCondition and policyAction may be attached when the complex policy rule structure is required. With its attached auxiliary class, an instance of policyInstance represents a single policy condition or a single policy action, but not both.

An instance of this class is named by an attribute it acquires via an attached auxiliary class. In the Core Schema, the naming attributes available for this purpose are policyConditionName and policyActionName.

The class definition is as follows:

```
    NAME            policyInstance
```

```
    DESCRIPTION     A structural class to which the auxiliary classes
                    policyCondition and policyAction may be attached,
                    when the complex policy rule structure is
                    required.
    DERIVED FROM    policy
    TYPE            structural
    AUXILIARY CLASSES    policyCondition, policyAction
    OID             <to be assigned>
```

6.10. The Auxiliary Class policyElement

Like policyInstance, this class also introduces no additional attributes, beyond those defined in the class policy from which it is derived. Its role is to "tag" an instance of a class defined outside the realm of policy as being nevertheless relevant to a policy specification. This tagging can potentially take place at two levels:

o Every instance to which policyElement is attached becomes an

instance of the class policy, since policyElement is a subclass of policy. Thus a DIT search with the filter "objectClass=policy" will return the instance. (As noted earlier, this approach does not work for some directory implementations. To accommodate these implementations, policy-related entries SHOULD be tagged with the keyword "POLICY".)

- With the policyKeywords attribute that it inherits from policy, an instance to which policyElement is attached can be tagged as being relevant to a particular type or category of policy, using standard keywords, administrator-defined keywords, or both.

The class definition is as follows:

```
NAME              policyElement
DESCRIPTION      An auxiliary class used to tag instances of
                  classes defined outside the realm of policy as
                  relevant to a particular policy specification.
DERIVED FROM     policy
TYPE              auxiliary
AUXILIARY CLASSES   none
OID               <to be assigned>
```

6.11. The Auxiliary Class policySubtreesPtrAuxClass

This auxiliary class provides a single, multi-valued attribute that points to a set of objects that are at the root of DIT subtrees containing policy-related information. By attaching this attribute to instances of various other classes, a policy administrator has a flexible way of providing an entry point into the directory that

allows a client to locate and retrieve the policy information relevant to it.

This object does not provide the semantic linkages between individual policy objects, such as those between a policy group and the policy rules that belong to it. Its only role is to enable efficient bulk retrieval of policy-related objects, as described in Section 5.4. Once the objects have been retrieved, a directory client can determine the semantic linkages by following DN pointers such as policyRulesAuxContainedSet locally.

Since policy-related objects may or may not be included in the DIT subtree beneath an object to which this auxiliary class is attached, there is no automatic presumption that this subtree should be searched by a directory client. If this subtree does contain policy-related objects, this is indicated by including a DN pointer to this object itself in the policySubtreesAuxContainedSet attribute.

The class definition is as follows:

```
NAME              policySubtreesPtrAuxClass
DESCRIPTION      An auxiliary class providing a DN pointer to roots
                  of DIT subtrees contained policy-related objects.
DERIVED FROM     top
TYPE              auxiliary
AUXILIARY CLASSES   none
OID               <to be assigned>
MUST              policySubtreesAuxContainedSet
```

6.11.1. The Attribute policySubtreesAuxContainedSet

This attribute provides an unordered set of DN pointers to one or more objects under which policy-related information is present. The objects pointed to may or may not themselves contain policy-related information.

The attribute definition is as follows:

```
NAME              policySubtreesAuxContainedSet
DESCRIPTION      Distinguished names of objects that serve as roots
                  for DIT subtrees containing policy-related
                  objects.  No order is implied.
SYNTAX            DN
OID               <to be assigned>
EQUALITY         distinguishedNameMatch
MULTI-VALUED
```


6.12. The Auxiliary Class policyGroupContainmentAuxClass

This auxiliary class provides a single, multi-valued attribute that points to a set of policyGroups. By attaching this attribute to instances of various other classes, a policy administrator has a flexible way of providing an entry point into the directory that allows a client to locate and retrieve the policyGroups relevant to it.

As is the case with policyRules, a policy administrator might have several different pointers to a policyGroup in the overall directory structure. The policyGroupContainmentAuxClass is the mechanism that makes it possible for the policy administrator to define all these pointers.

The class definition is as follows:

NAME	policyGroupContainmentAuxClass
DESCRIPTION	An auxiliary class used to bind policyGroups to an appropriate container object.
DERIVED FROM	top
TYPE	auxiliary
AUXILIARY CLASSES	none
OID	<to be assigned>
MUST	policyGroupsAuxContainedSet

6.12.1. The Attribute policyGroupsAuxContainedSet

This attribute provides an unordered set of DN pointers to one or more policyGroups associated with the instance of a structural class to which this attribute has been appended. The attribute definition is as follows:

NAME	policyGroupsAuxContainedSet
DESCRIPTION	Distinguished names of policyGroups associated in some way with the instance to which this attribute has been appended. No order is implied.
SYNTAX	DN
OID	<to be assigned>
EQUALITY	distinguishedNameMatch
MULTI-VALUED	

6.13. The Auxiliary Class policyRuleContainmentAuxClass

This auxiliary class provides a single, multi-valued attribute that points to a set of policyRules. By attaching this attribute to instances of various other classes, a policy administrator has a flexible way of providing an entry point into the directory that

allows a client to locate and retrieve the policyRules relevant to it.

A policy administrator might have several different pointers to a policyRule in the overall directory structure. For example, there might be pointers to all policyRules for traffic originating in a particular subnet from a directory entry that represents that subnet. At the same time, there might be pointers to all policyRules related to a particular DiffServ setting from an instance of a policyGroup explicitly introduced as a container for DiffServ-related policyRules. The policyRuleContainmentAuxClass is the mechanism that makes it possible for the policy administrator to define all these pointers.

Note that the cn attribute does NOT need to be defined for this class. This is because an auxiliary class is used as a means to collect common attributes and treat them as properties of an object. A good analogy is a #include file, except that since an auxiliary class is a class, all the benefits of a class (e.g., inheritance) can be applied to an auxiliary class.

The class definition is as follows:

```
NAME              policyRuleContainmentAuxClass
DESCRIPTION      An auxiliary class used to bind policyRules to an
                  appropriate container object.
DERIVED FROM     top
TYPE              auxiliary
AUXILIARY CLASSES   none
OID               <to be assigned>
MUST              policyRulesAuxContainedSet
```

6.13.1. The Attribute policyRulesAuxContainedSet

This attribute provides an unordered set of DN pointers to one or more policyRules associated with the instance of a structural class to which this attribute has been appended. The attribute definition is:

```
NAME              policyRulesAuxContainedSet
DESCRIPTION      Distinguished names of policyRules associated in
                  some way with the instance to which this attribute
```

```
                    has been appended.  No order is implied.
  SYNTAX            DN
  OID               <to be assigned>
  EQUALITY          distinguishedNameMatch
  MULTI-VALUED
```


7. Extending the Core Schema

The following subsections provide general guidance on how to create a domain-specific schema derived from the Core Schema, discuss how the vendor classes in the Core Schema should be used, and explain how policyTimePeriodConditions are related to other policy conditions.

7.1. Subclassing policyCondition and policyAction

In Section 5.3 above, there is a discussion of how, by representing policy conditions and policy actions as auxiliary classes in a schema, the flexibility is retained to instantiate a particular policy as either a simple policy rule or a complex one. This flexibility is lost if a schema takes either of two "hard-coded" paths:

- If the schema subclasses policyRule to add domain-specific characteristics that implicitly express a policy condition, a policy action, or both, then the schema supports only the simple policy rule form.

- If the schema introduces structural subclasses of policyCondition, policyAction, or both, then the schema supports only the complex policy rule form.

Even if the authors of a domain-specific schema can only envision one of these forms of policy rules being used when the schema is instantiated, it costs nothing extra to express the schema as auxiliary subclasses of policyCondition and policyAction. For standardized schemata, this document specifies that domain-specific information MUST be expressed in auxiliary subclasses of policyCondition and policyAction. It is RECOMMENDED that non-standardized schemata follow this practice as well.

7.2. Using the Vendor Policy Encoding Attributes

As discussed Section 6.6 "The Class vendorPolicyCondition", the attributes vendorPolicyConstraintData and vendorPolicyConstraintEncoding are included in the vendorPolicyCondition to provide an escape mechanism for representing "exceptional" policy conditions. The attributes vendorPolicyActionData and vendorPolicyActionEncoding in the vendorPolicyAction class play the same role with respect to actions. This enables interoperability between different vendors.

For example, imagine a network composed of access devices from vendor A, edge and core devices from vendor B, and a policy server from vendor C. It is desirable for this policy server to be able to configure and manage all of the devices from vendors A and B. Unfortunately, these devices will in general have little in common

---

(e.g., different mechanisms, different ways for controlling those mechanisms, different operating systems, different commands, and so forth). The escape conditions provide a way for vendor-specific commands to be encoded as OctetStrings, so that devices from different vendors can be commonly managed by a single policy server.

7.3. Using Time Validity Periods

Time validity periods are defined as a subclass of policyCondition, called policyTimePeriodCondition. This is to allow their inclusion in the AND/OR condition definitions for a policyRule. Care should be taken not to subclass policyTimePeriodCondition to add domain-specific condition properties. For example, it would be incorrect to add IPSec- or QoS-specific condition properties to the policyTimePeriodCondition class, just because IPSec or QoS includes time in its condition definition. The correct subclassing would be to create IPSec or QoS-specific subclasses of policyCondition and then combine instances of these domain-specific condition classes with the validity period criteria. This is accomplished using the AND/OR aggregation capabilities for policyConditions in policyRules.

8. Security Considerations

Security and denial of service considerations are not explicitly considered in this memo, as they are appropriate for the underlying policy architecture. However, the policy architecture must be secure as far as the following aspects are concerned. First, the mechanisms proposed under the framework must minimize theft and denial of service threats. Second, it must be ensured that the entities (such as PEPs and PDPs) involved in policy control can verify each other's identity and establish necessary trust before communicating. The schema defined in this document MUST not compromise either of these goals.

9. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11.

Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of

such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

10. Acknowledgments

This document is closely aligned with the work being done in the Desktop Management Task Force (DMTF) Service Level Agreements and Networks working groups. As noted, the Core Schema described here was initially defined in references [2] and [3]. We would especially

like to thank Raju Rajan, Sanjay Kamat, Andrea Westerinen, Lee Rafalow, Raj Yavatkar, Glenn Waters, David Black, Michael Richardson, Mark Stevens, David Jones, and Hugh Mahon for their helpful comments.

11. References

[1] J. Strassner and E. Ellesson, "Terminology for describing network policy and services", draft-strassner-policy-terms-00.txt, August 1998.

[2] Bhattacharya, P., and R. Adams, W. Dixon, R. Pereira, R. Rajan, "An LDAP Schema for Configuration and Administration of IPSec based Virtual Private Networks (VPNs)", Internet-Draft work in progress, October 1998

[3] Rajan, R., and J. C. Martin, S. Kamat, M. See, R. Chaudhury, D. Verma, G. Powers, R. Yavatkar, "Schema for Differentiated Services and Integrated Services in Networks", Internet-Draft work in progress, October 1998

[4] J. Strassner and S. Judd, "Directory-Enabled Networks", version 3.0c5 (August 1998).

[5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[6] Hovey, R., and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[7] Wahl, M., and S. Kille, T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[8] J. Strassner, policy architecture BOF presentation, 42nd IETF Meeting, Chicago, Illinois, October, 1998

---

[9] DMTF web site, http://www.dmtf.org.

12. Authors' Addresses

John Strassner
Cisco Systems, Bldg 1

170 West Tasman Drive
San Jose, CA 95134
Phone: +1 408-527-1069
Fax: +1 408-527-1722
E-mail: johns@cisco.com

Ed Ellesson
IBM Corporation, JDGA/501
4205 S. Miami Blvd.
Research Triangle Park, NC 27709
Phone: +1 919-254-4115
Fax: +1 919-254-6243
E-mail: ellesson@raleigh.ibm.com

Bob Moore
IBM Corporation, JDGA/501
4205 S. Miami Blvd.
Research Triangle Park, NC 27709
Phone: +1 919-254-4436
Fax: +1 919-254-6243
E-mail: remoore@us.ibm.com

13. Full Copyright Statement

TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

14. Appendix A - Guidelines for Construction of DNs

To Be Provided