Policy Framework Working Group
Internet-draft
Category: Standards Track

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

Policy Framework LDAP Core Schema
draft-ietf-policy-core-schema-05.txt
October 05, 1999 15:16

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html

Abstract

This document takes as its starting point the object-oriented information model for representing policy information currently under joint development in the Service Level Agreements (SLA) Policy working group of the Distributed Management Task Force (DMTF) and in the IETF's Policy Framework working group. The IETF document defining this information model is the "Policy Framework Core Information Model" [10]. This 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 information model 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 information model that correspond to them.

The LDAP schema described in this document consists of six very general classes: policy (an abstract class), policyGroup, policyRule, policyConditionAuxClass, policyTimePeriodConditionAuxClass, and policyActionAuxClass. The schema also contains two less general classes: vendorPolicyConditionAuxClass and vendorPolicyActionAuxClass. To achieve the mapping of the information model's relationships, the schema contains two auxiliary classes: policyGroupContainmentAuxClass and policyRuleContainmentAuxClass. Capturing the distinction between rule-specific and reusable policy conditions and policy actions introduces five other classes: policyRuleConditionAssociation, policyRuleActionAssociation, policyConditionInstance, policyActionInstance, and policyRepository. Finally, the schema includes two classes policySubtreesPtrAuxClass and policyElementAuxClass for optimizing LDAP retrievals. In all, therefore, the schema contains 17 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

joint development in the Service Level Agreements working group of the Distributed Management Task Force (DMTF) and in the IETF's Policy Framework working group. The IETF document defining this information model is the "Policy Framework Core Information Model" [10]. This

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 information model classes to a directory that uses LDAPv3 as its access protocol. Two types of mappings are involved:

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

- For the relationship classes in the information model, different mappings are possible. In this document the information model's relationship classes and their properties are mapped in three ways: to LDAP auxiliary classes, to attributes representing DN pointers, and to containment in the Directory Information Tree (DIT).

Implementations that use an LDAP directory as their policy repository SHALL use the LDAP policy schema defined in this document. The use of the information model defined in reference [10] 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.

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]. Current work appears in references [12] through [15]. 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 Common Information Model (CIM) effort in the Distributed Management Task Force (DMTF). Standardization of the core policy model in the DMTF is the responsibility of the SLA Policy working group.

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. The Policy Core Information Model

This document contains an LDAP schema representing the Policy Core Information Model, which is defined in the companion document "Policy

Framework Core Information Model" [10]. Other documents may subsequently be produced, with mappings of this same Core Information Model to other storage technologies. Since the detailed semantics of the Core Policy classes appear only in reference [10], that document is a prerequisite for reading and understanding this document.

3. Inheritance Hierarchy for the LDAP Core Policy Schema

The following diagram illustrates the class hierarchy for the LDAP Core Policy Schema classes:

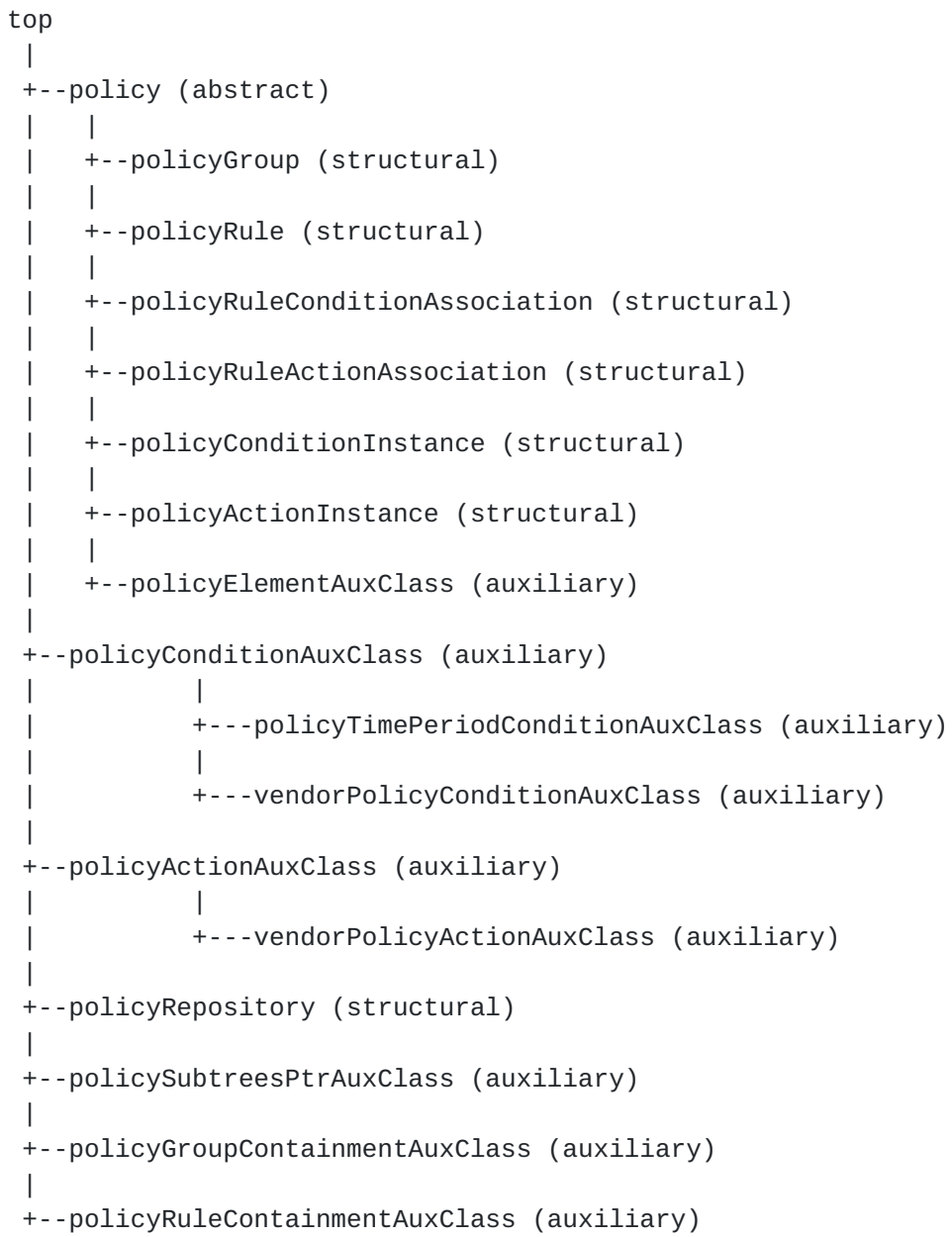

Figure 1. LDAP Class Inheritance Hierarchy for the Core Policy Schema

4. General Discussion of Mapping the Information Model to LDAP

The classes described in Section 5 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 some of the relationships defined in the information model. 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 the information model to LDAP.

4.1. Summary of Class and Relationship Mappings

Eight of the classes in the LDAP Core Policy Schema come directly from corresponding classes in the information model. Note that names of classes begin with an upper case character in the information model (although for CIM in particular, case is not significant in class and property names), but with a lower case character in LDAP.

Information Model	LDAP Class
Policy	policy
PolicyGroup	policyGroup
PolicyRule	policyRule
PolicyCondition	policyConditionAuxClass
PolicyAction	policyActionAuxClass
VendorPolicyCondition	vendorPolicyConditionAuxClass
VendorPolicyAction	vendorPolicyActionAuxClass
PolicyTimePeriodCondition	policyTimePeriodConditionAuxClass

Figure 2. Mapping of Information Model Classes to LDAP

The relationships in the information model map to DN-pointer attributes or to Directory Information Tree (DIT) containment in LDAP. Two of the DN-pointer attributes appear in auxiliary classes, which allows each of them to represent several relationships from the

information model.

Information Model Relationship	LDAP Attribute / Class
GroupJurisdiction	policyGroupsAuxContainedSet in policyGroupContainmentAuxClass
PolicyGroupInPolicyGroup	policyGroupsAuxContainedSet in policyGroupContainmentAuxClass
RuleJurisdiction	policyRulesAuxContainedSet in policyRuleContainmentAuxClass
PolicyRuleInPolicyGroup	policyRulesAuxContainedSet in policyRuleContainmentAuxClass
ConditionInPolicyRule	DIT containment [+ policyConditionDN in policyRuleConditionAssociation]
ActionInPolicyRule	DIT containment [+ policyActionDN in policyRuleActionAssociation]
PolicyRuleValidityPeriod	policyRuleValidityPeriodList in policyRule
ConditionInAdminDomain	DIT containment
ActionInAdminDomain	DIT containment

Figure 3. Mapping of Information Model Relationships to LDAP

Of the remaining classes in the LDAP Core Schema, two (policyElementAuxClass, and policySubtreesPtrAuxClass) are included to make navigation through the DIT and retrieval of the entries found there more efficient. This topic is discussed in Section 4.4 below.

The remaining five classes in the LDAP Core Schema, policyRuleConditionAssociation, policyRuleActionAssociation, policyConditionInstance, policyActionInstance, and policyRepository are all involved with the representation of policy conditions and policy actions in an LDAP directory. This topic is discussed in Section 4.3 below.

4.2. 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 naming attribute policyConditionName is used in two structural class: policyRuleConditionAssociation and policyConditionInstance. As discussed below in Section 4.3, these are the two structural classes to which the auxiliary class policyConditionAuxClass may be attached. The naming attribute policyActionName is similarly associated with two structural classes.) 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 all of its subclasses. In X.500, commonName typically functions as an RDN attribute, for naming instances of such classes as X.500's person.

Each of the Core Schema classes thus has two attributes suitable for naming: cn and its own class-specific attribute. Either of these attributes MAY be used for naming an instance of a Core Schema class. Consequently, implementations MUST be able to accommodate instances named in either of these ways.

Note that since they are required ("MUST") attributes, the class-specific naming attributes are always present in instances of their respective classes, even if they are not being used for naming the instances. In these cases the class-specific naming attributes may be used for other purposes. Note that "cn" and a class-specific naming attribute SHOULD NOT be used together to form a multi-part RDN, since support for multi-part RDNs is limited among existing directory implementations.

4.3. Rule-Specific and Reusable Conditions and Actions

The Core Information Model [10] distinguishes between two types of policy conditions and policy actions: ones associated with a single policy rule, and ones that are reusable, in the sense that they may be associated with more than one policy rule. There is no inherent difference between a rule-specific condition or action and a reusable one. There are, however, differences in how they are treated in a policy repository. For example, it's natural to make the access permissions for a rule-specific condition or action identical to those for the rule itself. It's also natural for a rule-specific condition

or action to be removed from the policy repository at the same time the rule is. With reusable conditions and actions, on the other hand, access permissions and existence criteria must be expressible without reference to a policy rule.

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

When the policy repository is realized in an LDAP-accessible directory, the distinction between rule-specific and reusable conditions and actions is realized via DIT containment. Figure 4 illustrates a policy rule Rule1 with one rule-specific condition CA and one rule-specific action AB. Because the condition and action are specific to Rule1, the auxiliary classes ca and ab that represent them are attached, respectively, to the structural classes CA and AB. These structural classes represent not the condition ca and action ab themselves, but rather Rule1's ASSOCIATION to ca and ab.

Note that the existence dependency of a rule-specific condition or action on its policy rule follows in this case from the semantics of DNs. Note also that for directory implementations supporting subtree-based access permissions, it's easy to indicate that parties with access to Rule1 also have access to its condition and action.

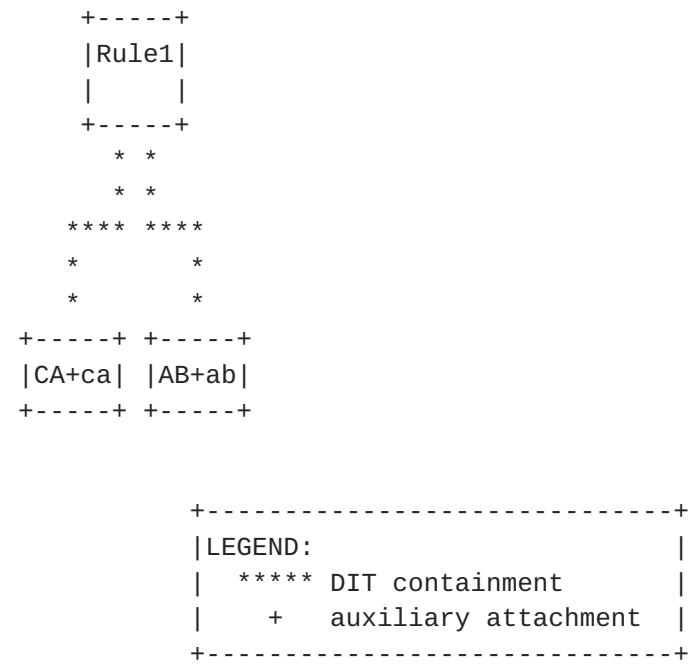

Figure 4. Rule-Specific Policy Conditions and Actions

Figure 5 illustrates the same policy rule Rule1, but this time its condition and action are reusable. The association classes CA and AB are still present, and they are still DIT contained under Rule1. But rather than having the auxiliary classes ca and ab attached to

themselves, CA and AB now contain DN pointers to other entries to which these auxiliary classes are attached. These other entries, CIA and AIB, are DIT contained under RepositoryX, which is an instance of

the class policyRepository. Because they are named under an instance of policyRepository, ca and ab are clearly identified as reusable.

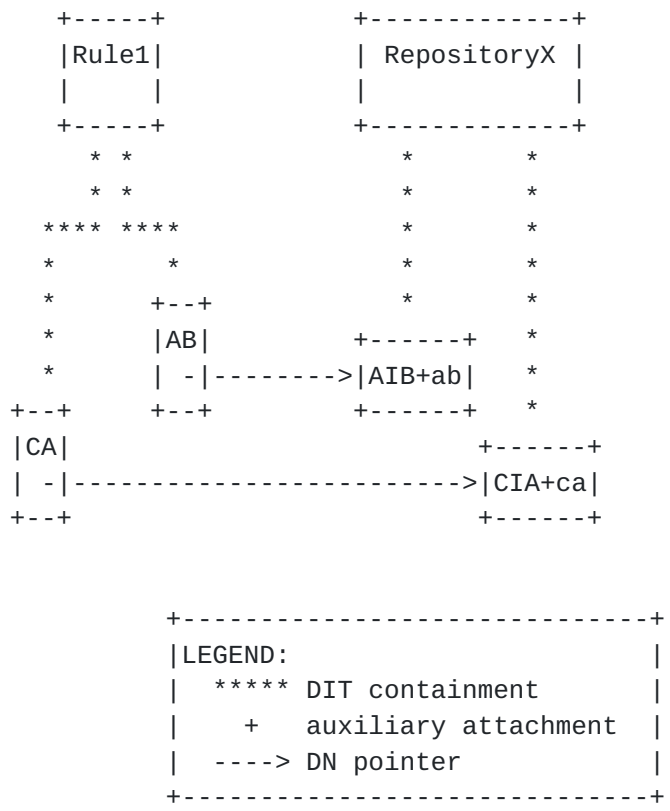

Figure 5. Reusable Policy Conditions and Actions

The classes policyConditionAuxClass and policyActionAuxClass do not themselves represent actual conditions and actions: these are introduced in their subclasses. What policyConditionAuxClass and policyActionAuxClass do introduce 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 representing either a rule-specific or a reusable policy condition or policy action.

In order to preserve the ability to represent either a rule-specific or a reusable condition or action, all the subclasses of policyCondition and policyAction MUST also be auxiliary classes.

4.4. Location and Retrieval of Policy Objects in the Directory

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

o How does it locate and retrieve the directory entries that apply to

its Policy Targets? 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 scalability? 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 Policy Consumer. 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 Policy Consumer SHOULD be able to take advantage of any tools that the policy administrator is able to build into the DIT, but it MUST be 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 Policy Consumer to retrieve all the policy-related objects it needs, and only those objects, using as few LDAP calls as possible. An important assumption underlying this approach is that the policy administrator has sufficient control over the underlying DIT structure to define subtrees for storing policy information. If the policy administrator does not have this level of control over DIT structure, a Policy Consumer can still retrieve the policy-related objects it needs individually. But it will require more LDAP access operations to do the retrieval in this way.

Figure 6 illustrates how LDAP optimization is accomplished.

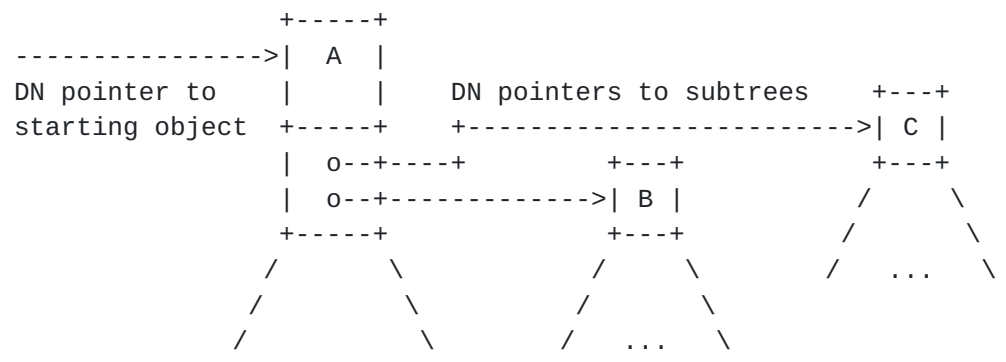

Figure 6. Using a policyContainer Object to Scope Policies

The Policy Consumer is configured initially with a DN pointer to some entry in the DIT. The structural class of this entry is not

important; the Policy Consumer 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

Policy Consumer. Since policySubtreesPtrAuxClass is an auxiliary class, it can be attached to an entry that the Policy Consumer would need to access anyway - perhaps an entry containing initial configuration settings for the Policy Consumer, or for a Policy Target that uses the Policy Consumer.

Once it has retrieved the DN pointers, the Policy Consumer 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 Policy Consumer 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 policyElementAuxClass 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.

If a Policy Consumer 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 Policy Consumer could be configured to request only those policy-related objects containing the keywords "DHCP" and "Eastern US".

To optimize what is expected to be a typical case, the initial request from the client includes not only the object to which its "seed" DN pointer points, but also the subtree contained under this object. The filter for searching this subtree is whatever the client is going to use later to search the other subtrees: "object class = policy", presence of the keyword "POLICY", or presence of a more specific policyKeyword.

Returning to the example in Figure 6, we see that in the best case, a Policy Consumer can get all the policy-related objects it needs, and only these objects, with exactly three LDAP requests: one to its

starting object A to get the pointers to B and C, as well as the policy-related objects it needs from the subtree under A, 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 Policy Consumer can then traverse their various DN pointers locally to understand the semantic relationships among them. The Policy Consumer should 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 6, 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 Policy Consumer has no guarantee that the entity that populates the directory won't use the policySubtreesPtrAuxClass, a Policy Consumer SHOULD understand this class, SHOULD be capable of retrieving and processing the entries in the subtrees it points to, and SHOULD 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 SHOULD also understand and be capable of using the policySubtreesPtrAuxClass. All of these requirements are "SHOULD"s rather than "MUST"s because an LDAP client that doesn't implement them can still access and retrieve the directory entries it needs . The process of doing so will just be less efficient than it would have been if the client had implemented these optimizations.

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

4.4.1. Aliases and Other DIT-Optimization Techniques

Additional flexibility in DIT structure is available to the policy administrator via LDAP aliasing. Figure 7 illustrates this flexibility.

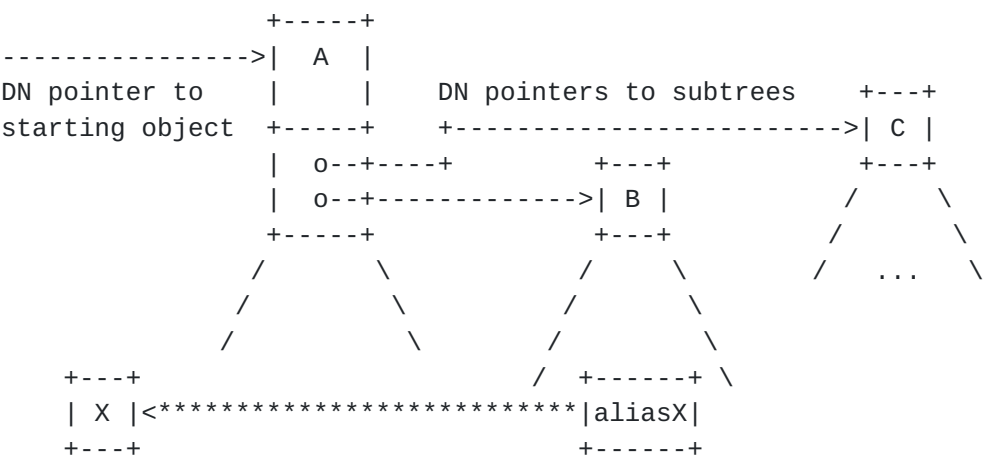

Figure 7. Addition of an Alias Object

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 Policy Consumer 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, Policy Consumers 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 7.

In addition to aliases, there are several other techniques for managing the placement of entries in the DIT and their retrieval by directory clients. Among these other techniques are referrals, LDAP URLs, attributes like seeAlso, and the extensible matching rule for dereferencing DN pointers discussed in reference [16]. Discussion of how these other techniques might be applied to policy-related entries in a directory is outside the scope of this document.

5. Class Definitions

The semantics for the LDAP classes mapped directly from the information model are detailed in reference [10]. Consequently, all that this document presents for these classes is a bare specification of the LDAP classes and attributes. More details are provided for the attributes listed above in Figure 3, which realize in LDAP the relationships defined in the information model. Finally, the classes

that exist only in the LDAP Core Schema are documented fully in this document.

The formal language for specifying the classes, attributes, and DIT structure and content rules is that defined in reference [7].

5.1. The Abstract Class "policy"

The abstract class "policy" is a direct mapping of the abstract class Policy from the Core Information Model. The four properties in Policy map directly to attributes in the class "policy".

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 policyElementAuxClass.

The class definition is as follows:

```
( <oid-oc1> NAME 'policy'
  DESC 'An abstract class with four attributes for describing
       a policy-related instance.'
  SUP top
  ABSTRACT
  MAY (cn $ caption $ description $ policyKeywords)
)
```

The attributes "cn" and "description" are defined in X.520. The remaining two attributes are defined as:

```
( <oid-at1> NAME 'caption'
       DESC 'A one-line description of this policy-related object.'
       SYNTAX IA5String
       EQUALITY caseExactIA5Match
       SINGLE-VALUE
)

( <oid-at3> NAME 'policyKeywords'
       DESC 'A set of keywords to assist directory clients in
            locating the policy objects applicable to them. Each
            value of the multi-valued attribute contains a single
            keyword. Standard keyword values are listed in the
            Policy Core Information Model document.'
       SYNTAX IA5String
       EQUALITY caseExactIA5Match
)
```

5.2. The Class policyGroup

The class definition for policyGroup is as follows. Note that this class definition does not include attributes to realize the PolicyRuleInPolicyGroup and PolicyGroupInPolicyGroup 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.

```
  ( <oid-oc2> NAME 'policyGroup'
        DESC 'A container for either a set of related policyRules or
             a set of related policyGroups.'
        SUP policy
        MUST (policyGroupName)
  )
```

The following DIT content rule indicates that an instance of policyGroup may have attached to it either DN pointers to one or more other policyGroups, or DN pointers to one or more policyRules.

```
  ( <oid-oc2>
        NAME 'policyGroupContentRule'
        DESC 'shows what auxiliary classes go with this object'
        AUX (policyGroupContainmentAuxClass $
             policyRuleContainmentAuxClass)
  )
```

The following DIT structure rules indicate that an instance of policyGroup may be named under any superior, using either the cn or the policyGroupName attribute.

```
  ( <oid-nf1> NAME 'policyGroupNameForm1'
    OC policyGroup
    MUST (cn)
  )

  ( 1 NAME 'policyGroupStructuralRule1'
    FORM policyGroupNameForm1
  )

  ( <oid-nf2> NAME 'policyGroupNameForm2'
    OC policyGroup
    MUST (policyGroupName)
  )


  ( 2 NAME 'policyGroupStructuralRule2'
    FORM policyGroupNameForm2
  )
```

The one attribute of policyGroup is defined as:

```
  ( <oid-at4> NAME 'policyGroupName'
        DESC 'The user-friendly name of this policy group.'
```

```
        SYNTAX IA5String
        EQUALITY caseExactIA5Match
        SINGLE-VALUE
```

)

5.3. The Class policyRule

This class represents the "If Condition then Action" semantics associated with a policy. The conditions and actions associated with a policy rule are modeled, respectively, with auxiliary subclasses of the auxiliary classes policyConditionAuxClass and policyActionAuxClass. Each of these auxiliary subclasses is attached to an instance of one of two structural classes. A subclass of policyConditionAuxClass is attached either to an instance of policyRuleConditionAssociation or to an instance of policyConditionInstance. Similarly, a subclass of policyActionAuxClass is attached either to an instance of policyRuleActionAssociation or to an instance of policyActionInstance.

Of the eight attributes in the policyRule class, seven are mapped directly from corresponding properties in the information model. The eighth attribute, policyRuleValidityPeriodList, realizes the PolicyRuleValidityPeriod association from the information model. Since this association has no "extra" properties (besides those that tie the association to its associated objects), the attribute policyRuleValidityPeriodList is simply a multi-valued DN pointer. (Relationships in the information model can have "extra" properties because CIM represents relationships as classes. See Sections 5.4 and 5.5 for examples of "extra" properties and how they are mapped to LDAP.) This attribute provides an unordered set of DN pointers to one or more instances of the policyTimePeriodConditionAuxClass, indicating when the policy rule is scheduled to be active and when it is scheduled to be inactive. A policy rule is scheduled to be active if it is active according to AT LEAST ONE of the policyTimePeriodConditionAuxClass instances pointed to by this attribute.

The ConditionInPolicyRule and ActionInPolicyRule associations, however, have additional properties: ActionInPolicyRule has an integer to sequence the actions, and ConditionInPolicyRule has an integer to group the conditions, and a Boolean to specify whether a condition is to be negated. In the Core Schema, these extra association properties are represented as attributes of two classes introduced specifically to model the associations: policyRuleConditionAssociation and policyRuleActionAssociation, defined, respectively, in Sections 5.4 and 5.5. Thus they do not appear as attributes of the class policyRule.

The class definition of policyRule is as follows:

(<oid-oc3> NAME 'policyRule'

```
        DESC 'The central class for representing the "If Condition
             then Action" semantics associated with a policy rule.'
        SUP policy
        MUST (policyRuleName)
```

```
          MAY (policyRuleEnabled $ policyRuleConditionListType $
               policyRuleValidityPeriodList $ policyRuleUsage $
               policyRulePriority $ policyRuleMandatory $
               policyRuleSequencedActions)
  )
```

The following DIT structure rules indicate that an instance of policyRule may be named under an instance of policyGroup, where each of these instances may be named using either cn or their respective class-specific naming attributes.

EDITOR'S NOTE: This restriction that an instance of policyRule may be named only under an instance of policyGroup is new -- it came in with Ryan's ABNF. It's easy enough to change it to say that an instance of policyRule can be named under anything, to allow for the case of a deployment with so few rules that the grouping provided by policyGroup is not needed. We just have to decide which way we want it.

```
  ( <oid-nf3> NAME 'policyRuleNameForm1'
        OC policyRule
        MUST (cn)
  )

  ( 3 NAME 'policyRuleStructuralRule1'
      FORM policyRuleNameForm1
      SUP 1 2
  )

  ( <oid-nf4> NAME 'policyRuleNameForm2'
    OC policyRule
    MUST (policyRuleName)
  )

  ( 4 NAME 'policyRuleStructuralRule2'
      FORM policyRuleNameForm2
      SUP 1 2
  )
```

The attributes of policyRule are defined as follows:

```
  ( <oid-at5> NAME 'policyRuleName'
        DESC 'The user-friendly name of this policy rule.'
        SYNTAX IA5String
        EQUALITY caseExactIA5Match
        SINGLE-VALUE
  )

  ( <oid-at6> NAME 'policyRuleEnabled'
```

```
            DESC 'An enumeration indicating whether a policy rule is
                  administratively enabled, administratively disabled, or
                  enabled for debug mode. The defined values for this
```

```
                attribute are enabled(1), disabled(2), and
                enabledForDebug(3).'
          SYNTAX INTEGER
          EQUALITY integerMatch
          SINGLE-VALUE
    )

    ( <oid-at7> NAME 'policyRuleConditionListType'
          DESC '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
          EQUALITY integerMatch
          SINGLE-VALUE
    )

    ( <oid-at8> NAME 'policyRuleValidityPeriodList'
          DESC 'Distinguished names of policyTimePeriodConditions that
               determine when the policyRule is scheduled to be active
               / inactive.  No order is implied.'
          SYNTAX DN
          EQUALITY distinguishedNameMatch
    )

    ( <oid-at9> NAME 'policyRuleUsage'
          DESC 'This attribute is used to provide guidelines on how
                this policy should be used.'
          SYNTAX DirectoryString
          EQUALITY caseIgnoreMatch
          SINGLE-VALUE
    )

    ( <oid-at10> NAME 'policyRulePriority'
          DESC 'A non-negative integer for prioritizing this policyRule
                relative to other policyRules. A larger value indicates
                a higher priority.'
          SYNTAX INTEGER
          EQUALITY integerMatch
          SINGLE-VALUE
    )

    ( <oid-at11> NAME 'policyRuleMandatory'
          DESC 'A flag indicating that the evaluation of the
                policyConditions and execution of policyActions (if the
                condition list evaluates to True) is required.'
          SYNTAX Boolean
          EQUALITY booleanMatch
```

```
        SINGLE-VALUE
    )

    ( <oid-at12> NAME 'policyRuleSequencedActions'
```

```
        DESC 'An enumeration indicating how to interpret the action
              ordering indicated via the policyRuleActionList
              attribute. The defined values for this attribute are
              mandatory(1), recommended(2), and dontCare(3).'
        SYNTAX INTEGER
        EQUALITY integerMatch
  )
```

5.4. The Class policyRuleConditionAssociation

This class contains attributes to represent the "extra" properties of the information model's ConditionInPolicyRule association. Instances of this class are related to an instance of policyRule via DIT containment. The policy conditions themselves are represented by auxiliary subclasses of the auxiliary class policyConditionAuxClass. These auxiliary classes are attached directly to instances of policyRuleConditionAssociation for rule-specific policy conditions. For a reusable policy condition, the auxiliary class is attached to an instance of the class policyConditionInstance, and there is a DN pointer to this instance from the instance of policyRuleConditionAssociation.

The class definition is as follows:

```
  ( <oid-oc4> NAME 'policyRuleConditionAssociation'
        DESC 'The class contains attributes characterizing the
              relationship between a policy rule and one of its
              policy conditions.'
        SUP policy
        MUST (policyConditionGroupNumber $ policyConditionNegated $
              policyConditionName)
        MAY (policyConditionDN)
  )
```

The following DIT content rule indicates that an instance of policyRuleConditionAssociation may have attached to it the auxiliary class policyConditionAuxClass, or one of its subclasses. This combination represents a rule-specific policy condition.

```
  ( <oid-oc4>
        NAME 'policyRuleConditionAssociationContentRule'
        DESC 'shows what auxiliary classes go with this object'
        AUX (policyConditionAuxClass)
  )
```

The following DIT structure rules indicate that an instance of policyRuleConditionAssociation may be named under an instance of policyRule, where each of these instances may be named using either cn

or their respective class-specific naming attributes.

(<oid-nf5> NAME 'policyRuleConditionAssociationNameForm1'

```
        OC policyRuleConditionAssociation
        MUST (cn)
  )

  ( 5 NAME 'policyRuleConditionAssociationStructuralRule1'
        FORM policyRuleConditionAssociationNameForm1
        SUP 3 4
  )

  ( <oid-nf6> NAME 'policyRuleConditionAssociationNameForm2'
        OC policyRuleConditionAssociation
        MUST (policyConditionName)
  )

  ( 6 NAME 'policyRuleConditionAssociationStructuralRule2'
        FORM policyRuleConditionAssociationNameForm2
        SUP 3 4
  )
```

The attributes of policyRuleConditionAssociation are defined as follows. Note that the class-specific naming attribute policyConditionName is also used in the class policyConditionInstance, where it identifies a reusable policy condition.

```
  ( <oid-at13>
        NAME 'policyConditionName'
        DESC 'A user-friendly name for a policy condition.'
        SYNTAX IA5String
        EQUALITY caseExactIA5Match
        SINGLE-VALUE
  )

  ( <oid-at14>
        NAME 'policyConditionGroupNumber'
        DESC 'The number of the group to which a policy condition
              belongs.  These groups are used to form the DNF or
              CNF expression associated with a policy rule.
        SYNTAX INTEGER
        EQUALITY integerMatch
        SINGLE-VALUE
   )

  ( <oid-at15>
        NAME 'policyConditionNegated'
        DESC 'Indicates whether a policy condition is negated in
              the DNF or CNF expression associated with a policy
              rule.  The value TRUE indicates that a condition is
              negated'
```

```
        SYNTAX Boolean
        EQUALITY booleanMatch
        SINGLE-VALUE
    )
```

```
  ( <oid-at16>
        NAME 'policyConditionDN'
        DESC 'A DN pointer to a reusable policy condition.'
        SYNTAX DN
        EQUALITY distinguishedNameMatch
        SINGLE-VALUE
  )
```

5.5. The Class policyRuleActionAssociation

This class contains an attribute to represent the one "extra" property of the information model's ActionInPolicyRule association, which makes it possible to specify an order for executing the actions associated with a policy rule. Instances of this class are related to an instance of policyRule via DIT containment. The actions themselves are represented by auxiliary subclasses of the auxiliary class policyActionAuxClass. These auxiliary classes are attached directly to instances of policyRuleActionAssociation for rule-specific policy actions. For a reusable policy action, the auxiliary class is attached to an instance of the class policyActionInstance, and there is a DN pointer to this instance from the instance of policyRuleActionAssociation.

The class definition is as follows:

```
  ( <oid-oc5> NAME 'policyRuleActionAssociation'
        DESC 'The class contains an attribute that represents an
              execution order for an action in the context of a
              policy rule.'
        SUP policy
        MUST (policyActionOrder $
              policyActionName)
        MAY (policyActionDN)
  )
```

The following DIT content rule indicates that an instance of policyRuleActionAssociation may have attached to it the auxiliary class policyActionAuxClass, or one of its subclasses. This combination represents a rule-specific policy action.

```
  ( <oid-oc5>
        NAME 'policyRuleActionAssociationContentRule'
        DESC 'shows what auxiliary classes go with this object'
        AUX (policyActionAuxClass)
  )
```

The following DIT structure rules indicate that an instance of

policyRuleActionAssociation may be named under an instance of policyRule, where each of these instances may be named using either cn or their respective class-specific naming attributes.

```
    ( <oid-nf7> NAME 'policyRuleActionAssociationNameForm1'
          OC policyRuleActionAssociation
          MUST (cn)
    )

    ( 7 NAME 'policyRuleActionAssociationStructuralRule1'
          FORM policyRuleActionAssociationNameForm1
          SUP 3 4
    )

    ( <oid-nf8> NAME 'policyRuleActionAssociationNameForm2'
          OC policyRuleActionAssociation
          MUST (policyActionName)
        )

    ( 8 NAME 'policyRuleActionAssociationStructuralRule2'
          FORM policyRuleActionAssociationNameForm2
          SUP 3 4
    )
```

The attributes of policyRuleActionAssociation are defined as follows. Note that the class-specific naming attribute policyActionName is also used in the class policyActionInstance, where it identifies a reusable policy action.

```
    ( <oid-at17>
          NAME 'policyActionName'
          DESC 'A user-friendly name for a policy action.'
          SYNTAX IA5String
          EQUALITY caseExactIA5Match
          SINGLE-VALUE
    )

    ( <oid-at33>
          NAME 'policyActionOrder'
          DESC 'An integer indicating the relative order of an action
                in the context of a policy rule.
          SYNTAX INTEGER
          EQUALITY integerMatch
          SINGLE-VALUE
    )

    ( <oid-at34>
          NAME 'policyActionDN'
          DESC 'A DN pointer to a reusable policy action.'
          SYNTAX DN
          EQUALITY distinguishedNameMatch
          SINGLE-VALUE
```

)

5.6. The Class policyConditionAuxClass

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. This auxiliary class can be attached to instances of two other classes in the Core Policy Schema. When it is attached to an instance of policyConditionAssociation, it represents a rule-specific policy condition. When it is attached to an instance of policyConditionInstance, it represents a reusable policy condition.

Since both of the classes to which this auxiliary class may be attached are derived from "policy", the attributes of "policy" will already be defined for the entries to which this class attaches. Thus this class is derived directly from "top".

The class definition is as follows:

```
( <oid-oc6> NAME 'policyConditionAuxClass'
      DESC 'A class representing a condition to be evaluated in
            conjunction with a policy rule.'
      SUP top
      AUXILIARY
)
```

5.7. The Class policyTimePeriodConditionAuxClass

This class provides a means of representing the time periods during which a policy rule is valid, i.e., active. The class definition is as follows. Note that instances of this class are named with the attributes cn and policyConditionName that they inherit, respectively, from policy and from policyCondition.

```
( <oid-oc7> NAME 'policyTimePeriodConditionAuxClass'
      DESC 'A class that provides the capability of enabling /
            disabling a policy rule according to a predetermined
            schedule.'
      SUP policyConditionAuxClass
      AUXILIARY
      MAY (ptpConditionTime $ ptpConditionMonthOfYearMask $
           ptpConditionDayOfMonthMask $ ptpConditionDayOfWeekMask $
           ptpConditionTimeOfDayMask $ ptpConditionTimeZone)
)
```

The attributes of policyTimePeriodConditionAuxClass are defined as follows:

```
( <oid-at19>
```

```
        NAME 'ptpConditionTime'
        DESC 'The range of calendar dates on which a policy rule is
              valid.  The format of the string is
```

```
            [yyyymmddhhmmss]:[yyyymmddhhmmss]'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
        SINGLE-VALUE
    )

    ( <oid-at20>
        NAME 'ptpConditionMonthOfYearMask'
        DESC 'A mask identifying the months of the year in which a
            policy rule is valid.  The format is a string of 12
            ASCII '0's and '1's, representing the months of the
            year from January through December.'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
        SINGLE-VALUE
    )

    ( <oid-at21>
        NAME 'ptpConditionDayOfMonthMask'
        DESC 'A mask identifying the days of the month on which a
            policy rule is valid.  The format is a string of 31
            ASCII '0's and '1's.'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
        SINGLE-VALUE
    )

    ( <oid-at22>
        NAME 'ptpConditionDayOfWeekMask'
        DESC 'A mask identifying the days of the week on which a
            policy rule is valid.  The format is a string of seven
            ASCII '0's and '1's, representing the days of the week
            from Sunday through Saturday.'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
        SINGLE-VALUE
    )

    ( <oid-at23>
        NAME 'ptpConditionTimeOfDayMask'
        DESC '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.  The format of the string is
            hhmmss:hhmmss'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
    )
```

```
    ( <oid-at24>
           NAME 'ptpConditionTimeZone'
           DESC 'The definition of the time zone for the
                 policyTimePeriodConditionAuxClass.  The format of
```

```
            the string is either 'Z' (UTC) or <'+'|'-'><hhmm>'
        SYNTAX PrintableString
        EQUALITY caseIgnoreMatch
        SINGLE-VALUE
    )
```

5.8. The Class vendorPolicyConditionAuxClass

The class definition is as follows:

```
  ( <oid-oc8> NAME 'vendorPolicyConditionAuxClass'
        DESC 'A class that defines a registered means to describe a
              policy condition.'
        SUP policyConditionAuxClass
        AUXILIARY
        MAY (vendorPolicyConstraintData $
             vendorPolicyConstraintEncoding)
  )
```

The attribute definitions for vendorPolicyCondition are as follows:

```
  ( <oid-at25>
        NAME 'vendorPolicyConstraintData'
        DESC '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
        EQUALITY octetStringMatch
  )

  ( <oid-at26>
        NAME 'vendorPolicyConstraintEncoding'
        DESC 'An OID identifying the format and semantics for this
              instance"s vendorPolicyConstraintData attribute.'
        SYNTAX OID
        EQUALITY objectIdentifierMatch
        SINGLE-VALUE
  )
```

5.9. The Class policyActionAuxClass

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 auxiliary class can be attached to instances of two other classes in the Core Policy Schema. When it is attached to an instance of policyActionAssociation, it represents a rule-specific policy action. When it is attached to an

instance of policyActionInstance, it represents a reusable policy action.

Since both of the classes to which this auxiliary class may be attached are derived from "policy", the attributes of "policy" will already be defined for the entries to which this class attaches. Thus this class is derived directly from "top".

The class definition is as follows:

```
( <oid-oc9> NAME 'policyActionAuxClass'
      DESC 'A class representing an action to be performed as a
            result of a policy rule.'
      SUP top
      AUXILIARY
)
```

5.10. The Class vendorPolicyActionAuxClass

The class definition is as follows:

```
( <oid-oc10> NAME 'vendorPolicyActionAuxClass'
      DESC 'A class that defines a registered means to describe a
            policy action.'
      SUP policyActionAuxClass
      AUXILIARY
      MAY (vendorPolicyActionData $ vendorPolicyActionEncoding)
)
```

The attribute definitions for vendorPolicyActionAuxClass are as follows:

```
( <oid-at28>
      NAME 'vendorPolicyActionData'
      DESC '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
      EQUALITY octetStringMatch
)

( <oid-at29>
      NAME 'vendorPolicyActionEncoding'
      DESC 'An OID identifying the format and semantics for this
            instance"s vendorPolicyActionData attribute.'
      SYNTAX OID
      EQUALITY objectIdentifierMatch
      SINGLE-VALUE
)
```

5.11. The Class policyConditionInstance

The role of this class in the Core Schema is to serve as the structural class to which the auxiliary class policyCondition is

attached to form a reusable policy condition. See Section 4.3 for a complete discussion of reusable policy conditions and the role that this class plays in how they are represented.

In addition to the cn attribute it inherits from "policy", this class uses the naming attribute policyConditionName, which was defined above in Section 5.4.

The class definition is as follows:

```
( <oid-oc11> NAME 'policyConditionInstance'
      DESC 'A structural class that contains a reusable policy
            condition.'
      SUP policy
      MUST (policyConditionName)
)
```

The following DIT content rule indicates that an instance of policyConditionInstance may have attached to it an instance of the auxiliary class policyCondition.

```
( <oid-oc11>
      NAME 'policyConditionInstanceContentRule'
      DESC 'shows what auxiliary classes go with this class'
      AUX (policyConditionAuxClass)
)
```

The following DIT structure rules indicate that an instance of policyConditionInstance may be named under an instance of policyRepository, using either cn or its class-specific naming attribute policyConditionName.

```
( <oid-nf9> NAME 'policyConditionInstanceNameForm1'
      OC policyConditionInstance
      MUST (cn)
)

( 9 NAME 'policyConditionInstanceStructuralRule1'
      FORM policyConditionInstanceNameForm1
      SUP 13 14
)

( <oid-nf10> NAME 'policyConditionInstanceNameForm2'
      OC policyConditionInstance
      MUST (policyConditionName)
)
```

```
    ( 10 NAME 'policyConditionInstanceStructuralRule2'
           FORM policyConditionInstanceNameForm2
           SUP 13 14
```

```
  )
```

5.12. The Class policyActionInstance

The role of this class in the Core Schema is to serve as the structural class to which the auxiliary class policyAction is attached to form a reusable policy action. See Section 4.3 for a complete discussion of reusable policy actions and the role that this class plays in how they are represented.

In addition to the cn attribute it inherits from "policy", this class uses the naming attribute policyActionName, which was defined above in Section 5.5.

The class definition is as follows:

```
  ( <oid-oc12> NAME 'policyActionInstance'
        DESC 'A structural class that contains a reusable policy
              action.'
        SUP policy
        MUST (policyActionName)
  )
```

The following DIT content rule indicates that an instance of policyActionInstance may have attached to it an instance of the auxiliary class policyAction.

```
  ( <oid-oc12>
        NAME 'policyActionInstanceContentRule'
        DESC 'shows what auxiliary classes go with this class'
        AUX (policyActionAuxClass)
  )
```

The following DIT structure rules indicate that an instance of policyActionInstance may be named under an instance of policyRepository, using either cn or its class-specific naming attribute policyActionName.

```
  ( <oid-nf11> NAME 'policyActionInstanceNameForm1'
        OC policyActionInstance
        MUST (cn)
  )

  ( 11 NAME 'policyActionInstanceStructuralRule1'
        FORM policyActionInstanceNameForm1
        SUP 13 14
  )
```

```
    ( <oid-nf12> NAME 'policyActionInstanceNameForm2'
           OC policyActionInstance
```

```
        MUST (policyActionName)
    )

    ( 12 NAME 'policyActionInstanceStructuralRule2'
        FORM policyActionInstanceNameForm2
        SUP 13 14
    )
```

5.13. The Auxiliary Class policyElementAuxClass

This class 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:

- Every instance to which policyElementAuxClass is attached becomes an instance of the class "policy", since policyElementAuxClass 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 policyElementAuxClass 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:

```
    ( <oid-oc13> NAME 'policyElementAuxClass'
        DESC 'An auxiliary class used to tag instances of classes
              defined outside the realm of policy as relevant to a
              particular policy specification.'
        SUP policy
        AUXILIARY
    )
```

5.14. The Class policyRepository

This class provides a container for reusable policy information, such as reusable policy conditions and/or reusable policy actions.

The class definition is as follows:

```
    ( <oid-oc17> NAME 'policyRepository'
        DESC 'A container for reusable information.'
```

```
        SUP top
        MUST (policyRepositoryName)
        MAY(cn)
```

```
  )
```

The following DIT structure rules indicate that an instance of policyRepository may be named under any superior, using either the cn or the policyRepositoryName attribute.

```
  ( <oid-nf13> NAME 'policyRepositoryNameForm1'
        OC policyRepository
        MUST (cn)
  )

  ( 13 NAME 'policyRepositoryStructuralRule1'
        FORM policyRepositoryNameForm1
  )

  ( <oid-nf14> NAME 'policyRepositoryNameForm2'
        OC policyRepository
        MUST (policyRepositoryName)
  )

  ( 14 NAME 'policyRepositoryStructuralRule2'
        FORM policyRepositoryNameForm2
  )
```

The one attribute of policyRepository is defined as:

```
  ( <oid-at35> NAME 'policyRepositoryName'
        DESC 'The user-friendly name of this policy repository.'
        SYNTAX IA5String
        EQUALITY caseExactIA5Match
        SINGLE-VALUE
  )
```

5.15. 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.

These entries may be placed in the DIT such that a well-known DN can be used by placing the structural entry (e.g. container) with the policySubtreesPtrAuxClass attached thereto in the root of the directory suffix. In this case, the subtree entry point can contain and/or point to all related policy entries for any well-known policy disciplines. Similarly, the subtree entry point may be placed in the

DIT such that the Policy Consumer's starting point is a subtree with policy-related entries that are dependent on a hierarchically-related set of subtrees (e.g., region, division, corporate). In this latter

case, DNs may be provided to the Policy Consumers via SNMP or other techniques.

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 4.4. 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 will often be included in the DIT subtree beneath an object to which this auxiliary class is attached, a client SHOULD request the policy-related objects from the subtree under the object with these pointers at the same time that it requests the pointers themselves.

Since clients are expected to behave in this way, the policy administrator SHOULD make sure that this subtree does not contain so many objects unrelated to policy that an initial search done in this way results in a performance problem. For example, policySubtreesPtrAuxClass SHOULD NOT be attached to the partition root for a large directory partition containing a relatively few policy-related objects along with a large number of objects unrelated to policy. A better approach would be to introduce a container object immediately below the partition root, attach policySubtreesPtrAuxClass to this container object, and then place the policy-related objects in the subtree under it.

The class definition is as follows:

```
( <oid-oc14> NAME 'policySubtreesPtrAuxClass'
      DESC 'An auxiliary class providing DN pointers to roots of
            DIT subtrees containing policy-related objects.'
      SUP top
      AUXILIARY
      MAY (policySubtreesAuxContainedSet)
)
```

5.15.1. 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:

```
    ( <oid-at30>
           NAME 'policySubtreesAuxContainedSet'
           DESC 'Distinguished names of objects that serve as roots for
```

```
              DIT subtrees containing policy-related objects. No
              order is implied.'
        SYNTAX DN
        EQUALITY distinguishedNameMatch
  )
```

5.16. 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:

```
  ( <oid-oc15> NAME 'policyGroupContainmentAuxClass'
        DESC 'An auxiliary class used to bind policyGroups to an
              appropriate container object.'
        SUP top
        AUXILIARY
        MAY (policyGroupsAuxContainedSet)
  )
```

5.16.1. 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:

```
  ( <oid-at31>
        NAME 'policyGroupsAuxContainedSet'
        DESC 'Distinguished names of policyGroups associated in some
              way with the instance to which this attribute has been
              appended.  No order is implied.'
        SYNTAX DN
        EQUALITY distinguishedNameMatch
  )
```

5.17. 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:

```
( <oid-oc16> NAME 'policyRuleContainmentAuxClass'
       DESC 'An auxiliary class used to bind policyRules to an
             appropriate container object.'
       SUP top
       AUXILIARY
       MAY (policyRulesAuxContainedSet)
)
```

5.17.1. 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:

```
( <oid-at32>
       NAME 'policyRulesAuxContainedSet'
       DESC 'Distinguished names of policyRules associated in some
             way with the instance to which this attribute has been
             appended. No order is implied.'
       SYNTAX DN
       EQUALITY distinguishedNameMatch
)
```

6. 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.

6.1. Subclassing policyCondition and policyAction

In Section 4.3 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 condition or action as either rule-specific or reusable. This flexibility is lost if a condition or action class is defined as structural rather than auxiliary. 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.

6.2. Using the Vendor Policy Encoding Attributes

As discussed Section 5.8 "The Class vendorPolicyConditionAuxClass", the attributes vendorPolicyConstraintData and vendorPolicyConstraintEncoding are included in vendorPolicyConditionAuxClass to provide an escape mechanism for representing "exceptional" policy conditions. The attributes vendorPolicyActionData and vendorPolicyActionEncoding in vendorPolicyActionAuxClass 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.

6.3. 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 association capabilities for policyConditions in policyRules.

7. Security Considerations

- General: See reference [10].

- Users: See reference [10].

- Administrators of Schema: In general, most LDAP-accessible directories do not permit old or out-of-date schemas, or schema elements to be deleted. Instead, they are rendered inactive. This makes it that much more important to get it right the first time on an operational system, in order to avoid complex inactive schema artifacts from lying about in the operational directory. The good news is that it is expected that large network operators will change schema design infrequently, and, when they do, the schema creation changes will be tested on an off-line copy of the directory before the operational directory is updated. Typically, a small group of directory schema administrators will be authorized to make these changes in a service provider or enterprise environment. The ability to maintain audit trails is also required here.

- Administrators of Schema Content (Directory Entries): This group requires authorization to load values (entries) into a policy repository directory schema, i.e. read/write access. An audit trail capability is also required here.

- Applications and Policy Consumers: These entities must be authorized for read-only access to the policy repository directory, so that they may acquire policy for the purposes of passing it to their respective enforcement entities.

- Security Disciplines:

  - Audit Trail (Non-repudiation): In general, standardizing mechanisms for non-repudiation is outside the scope of the IETF; however, we can certainly document the need for this function in systems which maintain and distribute policy. The dependency for support of this function is on the implementers of these systems, and not on any specific standards for implementation. The requirement for a policy system is that a minimum level of auditing via an auditing facility must be provided. Logging should be enabled. This working group will not specify what this minimal auditing function consists of.

- Access Control/Authorization: Access Control List (ACL) functionality must be provided. Standards for directories which use LDAPv3 as an access mechanism are still being worked on in

the LDAPext working group, as of this writing. The two administrative sets of users documented above will form the basis for two administrative use cases which require support.

- o Authentication: In the LDAP-accessible directory case, both TLS and Kerboros are acceptable for authentication. Existing LDAP implementations provide these functions within the context of the BIND request, which is adequate. We advise against using weaker mechanisms, such as clear text and HTTP Digest. Mutual authentication is recommended. The LDAPv3 protocol supports this, but implementations vary in the functionality that they support.

- o Integrity/Privacy: In the LDAP-accessible directory case, TLS is acceptable for encryption and data integrity on the wire. If physical or virtual access to the policy repository is in question, it may also be necessary to encrypt the policy data as it is stored on the file system; however, specification of mechanisms for this purpose are outside the scope of this working group. In any case, we recommend that the physical server be located in a physically secure environment.

In the case of Policy Consumer-to-Policy Target communications, the use of IPSEC is recommended for providing confidentiality, data origin authentication, integrity and replay prevention. See reference [11].

- o Denial of Service: We recommend the use of multiple policy repository directories, such that a denial of service attack on any one directory server will not make all policy data inaccessible to legitimate users. However, this still leaves a denial of service attack exposure. Our belief is that the use of a policy schema, in a centrally administered but physically distributed policy directory, does not increase the risk of denial of service attacks; however, such attacks are still possible. If executed successfully, such an attack could prevent Policy ConsumerÆs from accessing a policy repository, and thus prevent them from acquiring new policy. In such a case, the Policy Consumers, and associated Policy Targets would continue operating under the policies in force before the denial of service attack was launched. Note that exposure of policy systems to denial of service attacks is not any greater than the exposure of DNS with DNSSEC in place.

- o Other LDAP-accessible Directory Schema Considerations:

  - o Replication: Replication among directory copies across servers should also be protected. Replicating over connections secured by SSL or IPSEC is recommended.

8. 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.

9. Acknowledgments

This document is closely aligned with the work being done in the Distributed 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. Special thanks also to Ryan Moats, for providing the ABNF representation of the Core Policy Schema.

10. References

[1] Strassner, J., and E. Ellesson, "Terminology for describing network policy and services", draft-ietf-policy-terms-00.txt, June 1999.

[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] Strassner, J. 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 A. Coulbeck, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

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

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

[10] Moore, B., and E. Ellesson, J. Strassner, "Policy Framework Core Information Model", draft-ietf-policy-core-info-model-01.txt, October 1999.

[11] Yavatkar, R., and R. Guerin, D. Pendarakis, "A Framework for Policy-based Admission Control", draft-ietf-rap-framework-03.txt, June 1999.

[12] Stevens, M., and W. Weiss, H. Mahon, B. Moore, J. Strassner, G. Waters, A. Westerinen, J. Wheeler, "Policy Framework", draft-ietf-policy-framework-00.txt, September, 1999.

[13] Mahon, H., "Requirements for a Policy Management System", draft-ietf-policy-req-00.txt, September 1999.

[14] Snir, Y., and Y. Ramberg, J. Strassner, "QoS Policy Framework Information Model", draft-ietf-qos-policy-schema-01.txt, September 1999.

[15] Weiss, W., and J. Strassner, A. Westerinen, "Terminology for describing network policy and services", draft-weiss-policy-device-qos-model-00.txt, June 1999.

[16] Moats, R., and J. Maziarski, J. Strassner, "Extensible Match Rule to Dereference Pointers", draft-moats-ldap-dereference-match-00.txt June 1999.

11. Authors' Addresses

John Strassner

Cisco Systems, Bldg 15
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

12. Full Copyright Statement