Internet Engineering Task Force John Strassner
INTERNET DRAFT Stephen Schleimer
**17 November 1998** Cisco Systems

# Policy Framework Definition Language

draft-ietf-policy-framework-pfdl-00.txt

## Status of Memo



## Abstract

Recently, the IETF has developed protocols that classify packets in order to treat certain classes or flows of packets in a particular way compared to other classes or flows of packets. The successful wide-scale deployment of these protocols depends on the ability to administer and distribute consistent policy information to different types of network devices as well as hosts and servers that participate in policy decision making, administration, distribution and control. There is a clear need to develop a scalable framework for policy administration and distribution that will enable interoperability among multiple devices and device types that must work together to achieve a consistent implementation of policy.
This document defines a language, called the Policy Framework Definition Language (PFDL), that maps requirements for services to be provided by the network as defined in a business specification (e.g., an SLA) to a common vendor- and device-independent intermediate form. This enables policy information and specifications to be shared among the heterogeneous components that comprise the policy framework, and allows multiple vendors to use multiple devices to implement that framework. The PFDL is the common 'currency' that is exchanged between these heterogeneous components to enable them all to perform their function in providing, securing, distributing, and administering policy. The PFDL becomes the way to ensure that multiple vendors interpret the policy the same way while enabling vendors to provide value-added services.

Definition of Key Word Usage

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119 [TERMS]. These words will be capitalized to ensure that their intent in this context is easily discernible.

Table of Contents

Status of this memo
Abstract

**1. Introduction**

**2. Difference Between the PFDL and Other Languages**

**3. The Structure of Policies in the PFDL**
**3.1 Policy Component Hierarchy**
**3.1.1 The ComplexPolicy class**
**3.1.2 The SimplePolicy class**
**3.1.3 The PolicyRule class**
**3.1.4 The PolicyCondition class hierarchy**
**3.1.5 The PolicyAction class**
**3.2 Policy Conflict Detection**
**3.2.1 Types of Policy Conflicts**
**3.2.2 Intra-Policy Conflicts**
**3.2.3 Inter-Policy Conflicts**
**3.3 Service and Usage Policies**
**3.3.1 Service Policies**
**3.3.2 Usage Policies**
**3.4 Collective Aspects of Policy**
**3.4.1 Roles**
**3.4.2 Groups**

**4. The Grammar of the PFDL**
<lots of additional sections here>

**5. Examples of Using the PFDL**

**6. Security Considerations**

**7. Acknowledgements**

**8. References**

**9. Author's Addresses**

APPENDICES
Appendix A: PFDL Syntax in BNF

## 1. Introduction and Motivation

A policy is a named object that represents an aggregation of Policy Rules. The policy describes the overall business function(s) to be accomplished, while the set of policy rules defines how those business functions will be met. A policy rule defines a sequence of actions to be initiated when a corresponding set of conditions is satisfied. It should be noted that a condition may be negated (e.g., NOT (user IN GoldSLAGroup) ). Therefore, this policy condition is satisfied when the specified user is NOT a member of the GoldSLAGroup.

Policies are the link between a high-level business specification of desired services and low-level device configurations that provide those services. The Policy Framework Working Group will define a secure framework for policy administration, representation and distribution for multiple devices and device types. This includes devices whose primary job is to enforce the policy (e.g., a network element) as well as other devices that have one or more of the following roles: policy storage, distribution, decision-making, conflict detection, conflict resolution, administration, and management.

A framework that is comprised of heterogeneous components requires a common definition of policy. Furthermore, policy MUST be able to be represented and managed in an unambiguous, interoperable manner within a single network administrator's domain. Here, interoperability means providing a vendor- and device-independent specification of what the device configuration must do in order to provide the desired services. The Policy Framework Definition Language (PFDL) quantifies this link as a formal grammar. The purpose of the language is to translate from a business specification, such as those found in a Service Level Agreement, to a common vendor- and device-independent intermediate form. This enables policy information and specifications to be shared among the heterogeneous components that comprise the policy framework, and allows multiple vendors to use multiple devices to implement that framework. The PFDL is the common "currency" that is exchanged between these heterogeneous components to enable them all to perform their function in providing, securing, distributing, and administering policy. The PFDL becomes the way to ensure that multiple vendors interpret the policy the same way while enabling vendors to provide value-added services.

The PFDL will first address the needs of expressing QoS policies for the Differentiated Services, Integrated Services, RAP, and ISSLL Working Groups. The design will be iterated as necessary while the Policy Framework itself matures, and then further iterated to ensure that the needs of detailed schemata are fulfilled.
This draft represents the initial design of the PFDL.

## 2. Difference Between the PFDL and Other Languages

< to be supplied >

## 3. The Structure of Policies in the PFDL

In addition to the requirements listed in Section 1 above, the Policy Framework architecture must be scalable, interoperable and reusable. Here, scalability refers to the ability to build reusable arbitrarily complex policies from a set of simple ones. Interoperability means that the PFDL must be expressed in a way that enables essential information to be communicated between different vendors' policy management systems and the devices of different vendors that are controlled by the policy management systems of different vendors. Specifically, a policy management system should be able to be constructed from multiple vendors, and it should be able to convey policy information to multiple vendors' devices.

This requires a canonical definition of the structure of a policy. Reusability means that each component of a policy should be reusable by other policies.

### 3.1 Policy Component Hierarchy

The design of the PFDL is based on satisfying the Policy Information Model being designed in the DMTF [PLYMDL]. The class and relationship hierarchies of the Policy Information Model help define the structure of the PFDL grammar. Accordingly, this section will provide a brief overview of the components that define a policy in the Policy Information Model that require specific constructs to be defined in the PFDL.

#### 3.1.1 The ComplexPolicy class

A ComplexPolicy can contain one or more SimplePolicies. This enables reusability at the policy level. For example, when a user logs on to the network, many different policies must be combined. Some of the different policies that could be assigned to a user include:

- Security policies to enable proper access to network resources, as well as possible accounting and auditing functions
- A DHCP lease policy that allocates an IP Address depending on who the user is, what services the user is contracted for (which in turn can depend on the role of the user), how the user is logging in (e.g., Ethernet vs. PPP), and other factors
- QoS policies that get activated when the user logs on as well as other QoS policies that get activated on demand

A ComplexPolicy is modeled as an aggregation of SimplePolicies. This represents reusability at the highest (policy) level. The PFDL MUST be able to express the aggregation of simple policies into a more complex policy. Both of these concepts must be able to be identified as named elements.

### 3.1.2 The SimplePolicy class

A SimplePolicy can contain one or more PolicyRules. Each SimplePolicy (a.k.a., a PolicyRule class) contains a SET of PolicyConditions and a SET of PolicyActions.

The reason for this is that a PolicyRule is itself domain-specific. There are PolicyRules that define security mechanisms (e.g., authentication methods and ACL settings), QoS mechanisms (e.g., PHBs that will manage traffic), and other mechanisms for other related services. Each of these has its own definitions that are specific to its domain of knowledge, requiring a set of domain-specific conditions and actions. However, at a more abstract level, certain policies need one or more of these policy rules to be aggregated in order to provide the service(s) requested by a client.

For example, when a user logs on, the Policy Information Model can describe the binding between the user and the services that a network provides by aggregating a set of policy rules. This might consist of an overall Security Policy that governs how the user will be authenticated and what services the user is authorized to utilize based on a number of factors (time of day, method of login, etc.). There might be a set of QoS policies that get activated based on the user's SLA. There might be a DHCP Lease Policy that controls what dynamic IP address the user is assigned. There might be separate accounting and auditing policies that get triggered on an as-needed basis.

It is essential that these diverse policy rules get aggregated into a single consistent policy that can be associated with the user. The PFDL MUST be able to specify and support the aggregation of multiple policy rules into a single policy class. Furthermore, it must be able to treat policy rules and policy classes as discrete named objects.

The concepts of order and priority are also needed. For example, assume that a Policy is comprised of multiple PolicyRules, where one of the PolicyRules expresses security restrictions. If the conditions of the security policy rule are not satisfied, then it is irrelevant if the conditions of the other policy rules are satisfied - the actions of all policy rules will not be executed. Thus, the concept of rule ordering is needed. Visually:

```
  IF SecurityPolicyRule is satisfied THEN
     IF DHCPLeasePolicy is satisfied THEN
        Execute QoSPolicies
        Execute AuditingPolicies
     ENDIF
  ENDIF
```

Similarly, rule priority is a useful concept. For example, this provides one mechanism for resolving policy conflicts. Therefore, the

PFDL MUST be able to express relational and logical operators, along with priority and ordering concepts.

### 3.1.3 The PolicyRule class

The PolicyRule class expresses a set of conditions that, when satisfied, triggers a set of actions to be performed. A set of conditions is used as part of a canonical template within each policy knowledge domain. That is, security, QoS, DHCP, IPSEC, and other policy knowledge domains will in general have their own specific templates that define a particular set of conditions and an associated actions that must be executed if those conditions are satisfied. A "canonical template" is prescribed for interoperability reasons. It is a fact that multiple vendors will be supplying multiple components as part of any general policy architecture. Therefore, a common communication methodology is needed. An extensible, canonical definition of what constitutes the conditions and actions of a given knowledge domain is offered as the solution to this interoperability requirement. Therefore, the PFDL MUST support modeling a set of conditions and a corresponding set of actions as an extensible template.

Note that in addition to the above, a set of actions could be executed if the set of conditions are not satisfied. However, this can be modeled as a separate policy rule. This is preferred to complicating the model and the resulting grammar. The PFDL MUST be able to model a policy rule as an extensible template. The template consists of a set of conditions that, when satisfied, triggers the execution of an associated set of actions.

### 3.1.4 The PolicyCondition class hierarchy

A policy condition, in the context of the Policy Framework Working Group, is defined as testing one or more aspects of users of the network and/or traffic flowing through the network, in order to see if either the correct policy state can be maintained, or if a newly desired policy state can be achieved.

A PolicyRule is comprised of one or more policy conditions. Each policy condition is represented by a PolicyConditionList. For simplicity, if there are multiple PolicyConditionLists within a given PolicyRule, this version of the PFDL requires that the PolicyConditionLists are logically ORed together.

A PolicyConditionList aggregates one or more PolicyConditionStatements. For simplicity, this version of this draft requires that each of the PolicyConditionStatements within a given PolicyConditionList are logically ANDed together.

A PolicyConditionStatement consists of two parts, called a PolicyConditionCategory and a PolicyConditionValue. A PolicyConditionStatement defines a relation between a

PolicyConditionCategory and a PolicyConditionValue.

PolicyConditionCategories are predefined elements that are specific to a particular knowledge domain. They form the structure of the condition half of the policy rule template. For example, concepts like user, location, application type, and service are all categories of conditions that can be tested for a QoS policy, and hence are QoS PolicyConditionCategories.

PolicyConditionValues represent the value that a given PolicyConditionCategory can take. Some of the limits and other general parameters of a PolicyConditionValue are pre-defied (e.g., a valid network address must be entered), but only to enforce validation of a specific value to be tested. PolicyConditionValues are the means to represent user- and application-specific values of the category of elements that are to be tested as part of this policy rule.
The PFDL MUST support the representation and structuring of policy conditions as defined in this section.

### 3.1.5 The PolicyAction class hierarchy

The PolicyAction class is in reality a list of one or more policy action statements. These policy action statement may be executed either in any order (default case) or in a prescribed order, through the use of specially defined attributes.

A policy action, in the context of the Policy Framework Working Group, is defined as the changing of the configuration of one or more network elements in order to achieve a desired policy state. This (new) policy state provides one or more (new) behaviors.

A PolicyRule is comprised of one or more policy actions. Each policy action is represented by a PolicyActionList. In contrast with PolicyConditionLists, the PolicyActionLists are not by default ORed together. In fact, the default relationship between a set of PolicyActionLists is that they are all executed in ANY order. To accommodate special situations, facilities exist for imposing an ordering of the PolicyActionLists as well as conditional execution of one or more PolicyActionLists based on the outcome (e.g., success or failure) of a previously executed PolicyActionList.

A PolicyActionList aggregates one or more PolicyActionStatements. The PolicyActionStatements within a given PolicyActionList are by default executed in ANY order. However, facilities exist for imposing an ordering of the PolicyActionStatements within a given PolicyActionList as well as conditional execution of one or more PolicyActionStatements based on the outcome (e.g., success or failure) of a previously executed PolicyActionStatement.

A PolicyActionStatement is comprised of two parts, a PolicyActionCategory and a PolicyActionValue. A PolicyActionStatement defines a relation between a PolicyActionCategory and a PolicyActionValue. However, the PolicyActionValue is dependent on the type of PolicyActionCategory. Therefore, the PFDL combines these.

The PolicyActionCategory defines a canonical set of operations or treatments that can be given to traffic flowing into and out of a network element (e.g., deny, change code point, etc.) in a vendor- and device-independent manner. The PolicyActionValue specifies the type of mechanism to be used to provide the specified operation or treatment. PolicyActionCategories are predefined elements that are specific to a particular knowledge domain. They form the structure of the action half of the policy rule template. For example, concepts like providing a specific service for a particular application or class of traffic are categories of actions that can be performed for a QoS policy, and hence are QoS PolicyActionCategories.

A PolicyActionValue represents the value that a given PolicyActionCategory can take. Some of the limits and other general parameters of a PolicyActionValue are pre-defied (e.g., a valid network address must be entered), but only to enforce validation of a specific action to be performed. PolicyActionValues are the means to represent user- and application-specific values of the category of services that are to be provided by a specified network element as part of this policy rule.

The PFDL MUST support the representation and structuring of policy actions as defined in this section.

## 3.2 Policy Conflict Detection

Policy conflict detection is crucial to the design of a scalable and deployable policy framework. The concept of policy detection is actually fairly straightforward. However, the implementation of policy conflict detection can be quite complicated in the general case. Therefore, this proposal explicitly limits the power and flexibility of the PFDL in order to ensure that conflict detection and resolution is both doable and implementable.

### 3.2.1 Types of Policy Conflicts

There are two fundamental types of policy conflicts: those caused by within a policy (either a SimplePolicy or a ComplexPolicy) and those that cause a conflicting action to be taken in the network. These are referred to as Intra- and Inter-Policy conflicts.

### 3.2.2 Intra-Policy Conflicts

An intra-policy conflict occurs when the conditions of two or more policies can be simultaneously satisfied, but the actions of at least one of the policies can not be simultaneously executed. This implies several things:

- one or more policy rules of each of the policies is satisfied by the same request

- each condition of each of the conflicting policy rules is satisfied by the same request

- one or more actions specified by one policy conflict with one or more actions specified by the other policy

Intra-policy conflicts can be resolved in a number of different ways. The simplest is to change the conditions and/or actions of one of the policies so that it no longer conflicts with the other policies. However, if the policies must remain inherently conflicting, then there are a number of ways to resolve the conflict on an individual event basis, including the following:

- apply a "match-first" criteria, wherein conflicts are resolved by matching the first policy that is found
- apply a priority order criteria, wherein conflicts are resolved by finding all policy rules which match a given event and selecting only the rule with the highest priority
- use additional metadata to determine which rule or rules should be applied. The difference between this and straight priority is that priority is inherently linear, whereas metadata enables non-linear solutions, such as branching, to be used.

The PFDL MUST support facilities to resolve conflicts.

#### 3.2.3 Inter-Policy Conflicts

An inter-policy conflict is defined as two or more policies that, when applied to the network, result in conflicting configuration commands and/or mechanisms to be specified for one or more network devices. It is important to note that in this case, the two (or more) conflicting policies do not conflict when compared to each other, but do conflict when applied to a specific network device or devices. For example, two policies could specify conflicting configurations on a given interface, or specify that a certain number of queues be used in one network device and a different number of queues be used in another network device for the same traffic flow.

The PFDL must support constructs that enable such conflicts to be resolved.

### 3.3 Service and Usage Policies

There are two different types of policies. They are called Service Policies and Usage Policies. It is not mandatory for the PFDL to support explicit identification of whether a policy is categorized as a service or a usage policy - this is really more for the convenience of an associated UI and for the implementation of a Policy Server. However, the PFDL should not prevent the optional specification of whether a policy is categorized as a service or usage policy.

### 3.3.1 Service Policies

Service policies describe the creation of services in the network. They organize the facilities that the network provides into services that may be used later to satisfy the requirements of users of the network. For example, creating various QoS service classes (VoiceTransport, VideoTransport, ..., BestEffort) is done using service policies. This is accomplished, for example, by establishing the PHBs needed on backbone interfaces over which the traffic to be afforded the specific service is to flow.

The application of a service policy results in the creation of one or more named objects that represents network services that are available for usage policies.

### 3.3.2 Usage Policies

Usage policies describe how to allocate the services created by Service policies to requestors of services. Usage policies describe particular mechanism(s) employed to either maintain the current state of the object, or to transition an object from one state to a new state, in order to utilize the specified services.

Usage policies associate services that are provided by the network to clients of the network (e.g., users and applications). This can include services provided indirectly (e.g., mark packets of this type with this DS value) as well as directly (use this set of reservation parameters for this class of traffic).

Usage policy actions are specifically limited to associating a service with a particular PolicyActionCategory. For example, all users in the QuarterEndFinance group are assigned Gold Service for the SAP application under the following conditions.

Put another way, service policies describe what the network is capable of providing, and usage policies describe how to configure the network in order to take advantage of one or more services that the network provides.

## 3.4 Collective Aspects of Policy

An essential attribute of policy is its collective nature.  That is, policies are created in order to control many possibly heterogeneous objects of the system that are under the control of a policy management infrastructure. For example, when we create a particular PHB, we want to be able to apply it to all the interfaces to which it is appropriate.

Two mechanisms are provided for the collective aspects of policy: roles and groups.

### 3.4.1 Roles

A role is a label. A role indicates a function that an interface or device in the network serves. For example, an interface in the core of the network connecting to another interface in the core of the network provides "backbone" services (it aggregates a large number of flows; it is very high speed; etc). We assign to this interface the role "BackBone". When we create a policy appropriate to interfaces serving a backbone role, we assign the role "BackBone" to that policy.

Another important use of roles is to explicitly differentiate the functions provided by a device. Continuing the above example, an "Edge" role could be defined that describes the various services provided by interfaces at the edge of the network (e.g., filtering, traffic shaping, rate limiting, etc.). These services together describe the functionality of an edge interface, but also serve to differentiate it from the functionality of a core interface.

Roles enable an administrator to group the interfaces of multiple devices into common groups. This enables the administrator to apply the same policy to each of these interfaces collectively, as opposed to individually.

A role may be associated with, at most, a single Policy (where a single Policy may be as complex as necessary.  Policies associated with a role are specifically intended to apply to interfaces or devices that are assigned the role.

An interface may be assigned multiple roles. When an interface is assigned multiple roles, the policies designated by those roles are intended to apply to that interface. Roles may be parameterized. This is to enable the definition of more complex policies that define configurations that contain values that depend on the combination of type of interface, media being used, protocol being used, and other factors. For example, a policy might be applied to all Frame Relay interfaces. Depending on the actual network topology, the types of congestion experienced in these interfaces might be different. The parameterized policy offers a way to group these Frame Relay interfaces together (to simplify administration) while providing an inherent flexibility to accommodate the particular semantics of interfaces that are identified by this role.

For example, some kind of conditional expression associated with a Role which partitions the space of interfaces.  Those interfaces included by the conditional would take the role; those interfaces not included by the conditional would not take the role.

When there are several roles assigned to an interface, inter-policy conflicts may occur.  They are resolved as described in section 3.2.2.

### 3.4.2 Groups

Grouping is a fundamental concept of aggregation. In most systems, a group is a named object that contains individual objects as well as possibly additional (sub-)groups of objects. The key point here is that group membership is statically determined.

Unfortunately, networks are inherently dynamic in nature. This means that the ability to "group" objects whose membership is determined dynamically (e.g., a s a function of the state of the system) is required. We will call this a "Dynamic Group". The most important characteristic of a Dynamic Group is that its membership is determined dynamically when a reference to the Dynamic Group is evaluated.

Dynamic Group objects can therefore represent either individual objects or other groupings of objects.  For example, a Dynamic Group ranging over users may be used to limit which users on a certain set of interfaces have a certain kind of QoS assigned to packets they generate.

A Dynamic Group is only used as a value part of a condition of a policy. If a PolicyConditionValue uses a grouping construct, then that policy is satisfied if the object specified in the PolicyConditionCategory satisfies the relationship specified in the policy between the object in the PolicyConditionCatgegory and the membership of the Dynamic Group specified in the PolicyConditionValue. For example, suppose that a logon is in progress. Further suppose that a policy tests for users who are engineers belonging to a group that is to receive Gold QoS. Then, at the instant of the testing of the condition, the set of user objects that the Dynamic Group specifies (people that are entitled to receive Gold QoS) is determined.
If the object representing the user logging in is in the set specified by the Dynamic Group, then this condition of the policy is satisfied.

The objects of a Dynamic Group are always determined by the range of objects that the PolicyConditionCategory can specify.  For example, there are several ways in which the type <user_category> can specify the objects that it contains. These same methods can be used by other types of PolicyConditionCategories, but the different semantics of each type of PolicyConditionCategory result in different memberships being built for PolicyConditionCategory.

There are several ways in which a Dynamic Group can specify the objects It designates: by characterization, by enumeration, or by a combination of the two. A Dynamic Group uses characterization in order to be able to explicitly define acceptable values for the attributes of the object. For example, suppose that a <user_category> object has the attributes name, job title, and supervisor.  A group, LucasEngineers, might be characterized as follows:

```
  Group over <user_category> named LucasEngineers:
     Job Title = "engineer",
     Supervisor = "Luca".
```

Then, a condition might be written as follows:

```
        <user_category> IN LucasEngineers
```

This condition would be satisfied in the user designated by the object being tested in the <user_category> above had a supervisor attribute named "Luca" and job title attribute named "engineer".

A Dynamic Group may also specify the objects that are its members using enumeration.  That is, the individual objects that the Dynamic Group is to specify may simply be explicitly listed. For example, a Dynamic Group of type <user_category> can refer to the objects it contains, (e.g., LucasEngineers) by explicitly listing its members as opposed to characterizing attributes or behaviors of those objects. If the membership of the LucasEngineers Dynamic Group was explicitly defined, it might look as follows:

```
    {<Cal>, <Tom>, <Steve>, <Edwin>}
```

where each of the bracketed entities refers to a user object.

Finally, a group may specify the object it designates by both characterization and enumeration. The above examples could easily be combined to yield such a group.

## 4. The Grammar of the PFDL

<This section will be finalized shortly>

## 5. Examples of Using the PFDL

<This section will be finalized shortly>

## 6. 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.

## 7. Acknowledegments

Many thanks to the useful discussions and suggestions from the Internet
Community at large but especially to <to be supplied>.

## 8. References

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

<more to be supplied>

## 9. Authors' Addresses

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

Stephen Schleimer
Cisco Systems
170 West Tasman Drive, Building D
San Jose, CA 95134
Phone:  +1-408-527-3291
Fax:    +1-408-???-????
E-mail:  sschlmr@cisco.com

Appendix A: BNF of the PFDL

```
<PFDL_Program>
        ::=     [<policy_definition>]+

<policy_definition>
        ::=     [<policy_rule>]+
        |       <role_identifier> ':' [<policy_rule>]+

<policy_rule>
        ::=     'IF'   <policy_condition_OR_list>
                'THEN' <policy_action_priority_list>

<policy_condition_OR_list>      /* still have to put in priority */
        ::=     <policy_condition_AND_list ['OR' <policy_condition_AND_list>]*

<policy_condition_AND_list>
        ::=     <policy_condition_statement>
        ['AND' <policy_condition_statement>]*

<policy_condition_statement>
        ::=     <policy_condition_expr>
        |       'NOT' <policy_condition_expr>

<policy_condition_expr>
        ::=     <user_category_statement>
        |       <application_category_statement>
        |       <device_category_statement>
        |       <interface_category_statement>
        |       <protocol_category_statement>
        |       <address_category_statement>
        |       <traffic_category_statement>
        |       <SLA_category_statement>
        |       <time_category_statement>

<user_category_statement>
        ::=     USER_CATEGORY_REFERENCE <rel_op>
                <user_category_value>

/* the values for <user_category_value> are all either email addresses
   or a Distinguished Name */
<user_category_value>   /* the following are all DNs */
        ::=     <user_name>
        |       <user_group_name>
        |       <organizational_unit_name>              /* "division" */
        |       <organization_name>

<application_category_statement>
        ::=     APPLICATION_CATEGORY_REFERENCE <rel_op>
                <application_category_value>
```

```
<application_category_value>
        ::=     <application_name>
        |       'SOURCE:' <application_port_number>
        |       'DESTINATION:' <application_port_number>

<device_category_statement>
        ::=     DEVICE_CATEGORY_REFERENCE <rel_op>
                <device_category_value>

<device_category>
        ::=     <device_name>
        |       <address_spec>
        |       <role_identifier>

<interface_category_statement>
        ::=     INTERFACE_CATEGORY_REFERENCE <rel_op>
                <interface_category_value>

<interface_category_value>
        ::=     <role_identifier>

<protocol_category_statement>
        ::=     PROTOCOL_CATEGORY_REFERENCE <rel_op>
                <protocol_category_value>

<protocol_category_value>
        ::=     <protocol_name> [<protocol_options>]*
        |       <protocol_number> [<protocol_options>]*

<address_category_statement>
        ::=     ADDRESS_CATEGORY_REFERENCE <rel_op>
                <address_category_value>

<address_category_value>
        ::=     'SOURCE:' <address_spec>
        |       'DESTINATION:' <address_spec>
        |       'BOTH:' <address_spec> <address_spec>

<traffic_category_statement>
        ::=     TRAFFIC_CATEGORY_REFERENCE <rel_op>
                <traffic_category_value>

<traffic_category_value>
        ::=     <ingress_DSV>
        |       <ingress_ToS_byte_value>
        |       <ingress_802.1_value>
        |       <ingress_ATM_QoS>
        |       <ingress_FR_QoS>

<SLA_category_statement>
```

```
        ::=     SLA_CATEGORY_REFERENCE <rel_op>
                <sla_category_value>
```

```
<SLA_category_value>
        ::=     'SLA NAME:' SLA_VALUE [<SLA_name_options>]*

<time_category_statement>
        ::=     TIME_CATEGORY_REFERENCE <rel_op>
                <time_category_value>

<time_category_value>
        ::=     <absolute_time_category_value>
        |       <relative_time_category_value>

<absolute_time_category_value>
        ::=     <valid_time_period>
        |       <valid_time_period> <time_period_mask>

<relational_operator>
        ::=     'IN'
        |       'NOT IN'
        |       'EQUALS'
        |       'NOT EQUALS'
        |       'GREATER THAN'
        |       'GREATER THAN OR EQUAL TO'
        |       'LESS THAN'
        |       'LESS THAN OR EQUAL TO'

<policy_action_priority_list>
        ::=     [<policy_action_statement>]+

<policy_action_statement>
        ::=     <policy_action_category> [<apply_spec>] [<seq_num>]

<policy_action_category>
        ::=     'PERMIT' <permit_deny_category>
        |       'DENY' <permit_deny_category>
        |       'REMARK PACKET' <remark_spec>
        |       'START USING' <service_policy>
        |       'STOP USING' <service_policy>
        |       'Table Definition' <table_def>

<table_def>
        ::=     '{' <table_type> <table_shape> <table_content> "}"

<table_type>
        ::=     'RED_THRESHOLDS'
        |       'TAIL_DROP'
        |       ...

<table_shape>
        ::=     '{' <table_dimensions> <table_axis_sizes> '}'
```

```
<table_dimensions>
        ::=     INTEGER
```

```
<table_axis_sizes>
        ::=     '{' INTEGER* '}'

<table_content>
        ::=     '{' <table_row>* '}'

<table_row>
        ::=     '{' <table_column_entry>* '}'
        |       <row_index> '{' <table_column_entry>* '}'

<table_column_entry>
        ::=     INTEGER
        |       <column_index> ':' INTEGER

<row_index>
        ::=     INTEGER

<column_index>
        ::=     INTEGER
```