INTERNET-DRAFT
Expires: April 2004

A. Westerinen
Cisco Systems
Editor
October 2003

Policy Framework
<draft-ggf-policy-framework-00.txt>

Status of this Memo

This document provides information to the community regarding the functional components of a policy-based management framework for grid environments. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

This document articulates the requirements and basic framework of a policy-based management system for grid environments. It focuses on the storage and retrieval of Policy Rules from a repository, for use in the management and operation of a grid. This framework document describes functional components and operational characteristics of a system that is intended to be device, resource, service and vendor independent, interoperable and scalable.

There are three basic sections of this draft, addressing:

- the motivation for policy-based management that briefly describes the requirements for implementing policy in a grid;
- a reference model that defines a first-level functional decomposition of such a framework, and captures the key concepts in defining policy tools, Policy Rules, the use of a repository and schema, and the mechanisms underlying the definition, storage and retrieval of policies; and
- a description of each of the functional components, as well as a narrative about how a policy system can implement prescribed policies.

Full Copyright Notice

Copyright © Global Grid Forum (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property Statement

The GGF 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. 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 GGF Secretariat.

The GGF 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 recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

Table of Contents

1. Introduction

The purpose of a policy system is to manage and consistently control a grid environment, so that its operations conform to the business goals of the organization(s) that operate and utilize it. Ultimately, achieving such control requires altering the behavior of the individual entities that comprise the network, systems and/or grid. One approach is to alter the behavior of these entities individually by using a centralized management application. Iterating through a list of entities, a management application achieves control of the grid by manipulating the operational parameters of each entity separately.

Taking this approach places a disproportionate burden upon management applications. To effectively control a grid environment, management software must have explicit knowledge of the interfaces of each entity that it endeavors to control, as well as knowledge of the capabilities of each of these entities. As a result, management software is often forced to manage only those features controlled by the interfaces common to the majority of the entities in the grid. Implementing policy in this way remains piecemeal and proprietary.

The policy framework described in this document represents an alternative approach to controlling the operational characteristics of a grid. Unlike traditional management approaches, the systems developed within the policy framework implement policy by managing the storage and deployment of prescribed rules, instead of implementing policy by centralizing control functions into a single software application. A policy system devised under this framework shifts the focus from configuring individual resources to setting policy for the grid in aggregate, and controlling element behavior through grid policy.

At the center of such a policy systems is the Policy Rule. Policy Rules may be general and abstract, or specific and concrete. In either case, Policy Rules represent a pairing of conditions and actions that are intended to be device-, resource-, service- and vendor-independent.

The Policy Rule serves as the point of interoperability between entities participating in any policy system developed within this framework. To make the Policy Rule into the main point of interoperability, the following must be described:

- Composition and declaration of Policy Rules
- Characteristics of the entities that are being controlled by Policy Rules
- Relationships and interactions among the objects being managed

The composition of Policy Rules, along with some of the characteristics of the elements that are being controlled by Policy Rules, are described by a model or language. This defines the format and the organization of the storage for Policy Rules, as well as the data that characterize the elements being controlled by Policy Rules. Other characteristics of devices, resources and services, used to capture the semantics and relationships between different objects being managed, define how the conditions and actions represented in a Policy Rule are interpreted and what effect they have on the functions of the grid. These are described in an information model for the grid. This document presents a context for the schema and semantic definitions, and enumerates the functional elements that may be required to realize a complete policy system.

A policy system built upon the expression of rules must demonstrate at least three abilities:

1. The ability to enable a user to define and update Policy Rules.
2. The ability to store and retrieve Policy Rules.
3. The ability to interpret, implement and enforce Policy Rules.

To better understand the ramifications of the list above, we can recast it as a list of the functional elements of a policy system. A possible breakdown follows:

1. A Policy Management Tool, to enable an entity (e.g.: person, application) to define and update Policy Rules and optionally, monitor their deployment. For example: a graphical or command line/script interface.
2. A Policy Repository, for persistent storage and retrieval of Policy Rules. (Note: that the repository simply stores data, it does not in general process or act on it).
3. A Policy Consumer (a convenient grouping of functions) is responsible for acquiring Policy Rules, deploying Policy Rules, and optionally translating Policy Rules into a form useable by Policy Targets.
4. A Policy Target's (a functional element) behavior is dictated by Policy Rules. The Policy Target carries out the action indicated by the Policy Rule.

Policy Consumers and Policy Targets are logical entities and represent interfaces, not necessarily physical entities. Consequently, Policy Consumers, and Policy Targets can be realized in a number of combinations. A Policy Consumer can be realized in software running on a general-purpose computer separate from the Policy Target. Alternatively, a Policy Consumer can be coupled with a Policy Target and realized in software running on a specialized device like a router or a switch, or on a general-purpose computer.

Regardless of where the Policy Consumer software executes, its purpose is to acquire, optionally translate and deploy Policy Rules. Functionally, translating rules is separate from the implementation of the rule, which is the evaluation of conditions

and the execution of actions. Although a single software or system entity may be responsible for both the acquisition and deployment of Policy Rules, Policy Consumers can be functionally distinct from the targets of the Policy Rules.

For example, a Policy Rule may state that a certain group of users are to be given priority service. A device may be able to make a decision based on criteria similar to that expressed in the Policy Rule. A Policy Consumer may be employed to interpret the Policy Rule and create an analogous but more device-specific form. For example, the Policy Consumer might translate a condition expressed in terms of resource names into one containing network addresses. In such a case, a network device is the Policy Target.

Policy Rules may also contain references to time in their conditions. Some Policy Targets may be incapable of evaluating conditions containing time. In such a case, a Policy Consumer may decompose the Policy Rule and distribute the decision process between itself and the Policy Target.

In some situations, a physical device can be involved in affecting policy in a grid, while not being the Policy Target. In such a situation, the Policy Consumer and Policy Target functions are combined and realized in a software application, and the physical device is simply manipulated by the software in which the Policy Consumer and Policy Target are realized. Examples include elements that have no facility to interpret policy, but can be used to affect policy.

A router capable of enabling and disabling its ports, but incapable of interpreting standardized policy expressions stored in a repository, can serve as another example. Suppose an organization has a set of game servers, and wants to limit access to these servers to periods of the day outside normal working hours. A Policy Rule governing access to the servers could be written in two ways.

- It could specify time conditions, and an action indicating that access to the servers should be enabled or disabled.
- It could specify the same time conditions, but the action could contain directives specific to the element where the policy is to be deployed.

In either case, the aforementioned router can be used to affect policy. Both rules require the development of software to interpret the Policy Rule on behalf of the router. The first option can be standardized, and although the element cannot evaluate the Policy Rule, an application can be created to function as the Policy Consumer and Policy Target. The latter form of the Policy Rule is more element-specific. In both cases, the Target of the rule is the router.

Another motivation for the functional split occurs when policy condition(s) cannot be evaluated by the same entity that executes the action(s). So, the information stored in the policy action must, for certain cases, be element-specific. This framework accommodates both element-specific as well as element-independent policies.

The purpose of discerning a difference between Policy Consumers and Policy Targets is to make it easier to understand Policy Rule semantics, and develop the building blocks for standard policy expressions. In an effort to devise examples of Policy Rules, people often express rules that imply two distinct subjects within the same rule. The result is either a rule that makes no sense to others, or one that leads us to the development of element-specific rules.

It is important to note the steps in "implementing a Policy Rule". Policy Consumers acquire and optionally translate Policy Rules. Policy Targets implement Policy Rules in a much more constrained fashion. Two choices are possible:

- Behaving according to contents of the Policy Rule as a result of treating the behavioral specification as a set of direct commands, or
- Operating in a manner consistent with configuration parameters received from a Policy Consumer that has interpreted Policy Rules.

Implementers may also choose to add mechanisms to measure the effectiveness of Policy Rules, to establish feedback loops, and to ensure synchronization between functional elements. . . . [Need details]

2. Terminology

See RFC3198, Terminology for Policy-Based Management
[Are there other terms to define?]

3. Policy Framework

[Throughout Sections 3 and 4, add grid examples]

3.1. The Conceptual Model

This section introduces a policy framework, provides a first-level functional decomposition of it, and describes the role of the functional elements in the framework. This is a conceptual model, and not intended as a specification of components that must be present in a policy system. Such issues as communication between multiple Policy Consumers will be covered later in this document.

This framework is based on the four functional elements described in Table 1. The policy framework does not require that all functional elements be implemented nor does it specify implementation packaging of functional elements.

Functional Element	**Functions Performed**
Policy Management Tool	Policy editing, presentation, rule translation, rule validation, global conflict resolution, other functions
Policy Repository	Storage, search and retrieval of policies
Policy Consumer	Rule locator, element adapter, state resource validation, rule translation, rule transformation, other functions
Policy Target	Operation as specified by the actions of the Policy Rule / MAY perform rule validation, execution feedback and other functions

Table 1. Functional Elements of a Policy Framework

The implementation of three different abstractions of policies is permitted. The three levels supported by this model are:

1. The administrators' view of policy is to abstract general configuration and operational characteristics of the resources in a policy domain and the management of service-level objectives (SLOs) for these resources. SLOs are frequently derived from contractual service-level agreements (SLAs) and may be probabilistic in nature. The Policy Management Tool may provide significant value-add in the level of abstraction and degree of sophistication of the GUI presentation of SLAs and SLOs, and in the mapping between these and the lower-level Policy Rules.

2. The Policy Rules, as stored in the Policy Repository according to a defined information model or text form (which may also take the form of separate schemata that are derived from it), provide a deterministic set of policies for managing the behavior of resources in the policy domain. These Policy Rules are usually produced by the Policy Management Tool, stored in the Policy Repository and consumed by the Policy Consumer. However, note that in some cases, it is desirable to ship the Policy Rules directly to the Policy Consumers (without first storing them in the Policy Repository), to allow for further processing before they are stored. For example, a Policy Rule could be specified in the Policy Management Tool, but a feasibility check, as well as conflict resolution, may first be performed before storing the Policy Rule in the Policy Repository.
3. The policy mechanisms are policy discipline-specific and may include implementation-specific mechanisms and representations of the Policy Rules. They are the APIs, methods, protocols and other constructs used to forward and evaluate policies and perform actions on grid components. Ultimately, policy mechanisms result in Policy Targets taking action to deliver the services as prescribed at the administrative interface of the Policy Management Tool.

[Include additional images, for example, from SNIA policy discussions]

3.2. Policy Specification

The administrative user of the Policy Management Tool functional component specifies abstract policies that have meaning to the administrator and, indirectly, the end user or application for whom the policy is prescribed. Although specific enough to implement service level objectives via the Policy Management Tools' abstraction of the Policy Rules and mechanisms, these administratively specified Policy Rules may not be specific enough to allow for direct mapping to grid equipment configurations for deployment. It would be unusual (but not impossible) for an administrator or software automating administrative function to specify policy for a specific network traffic filter or job queuing parameter.

The Policy Management Tool also provides the mapping of the prescribed policies to a set of Policy Rules. Policy Management Tools should implement consistency checking of the Policy Rules to verify that the Policy Rules are consistent prior to placing them into the Policy Repository (see section 5).

It is not necessary to employ all functional elements as distinct physical entities. It is also possible to implement the Policy Rules and policy mechanism layers without implementing a Policy Management Tool. In fact, the central purpose of this framework is to enable interoperable implementations of Policy Consumers and Targets using common schema in a Policy Repository.

3.3. Policy Rules

Administrators manually create and/or manipulate Policy Rules. Also, a Policy Management Tool can produce the Policy Rules, which the Policy Consumers then use to appropriately influence the behavior of the Policy Targets. The Policy Rules specify the logic used to deliver the prescribed service and service-levels. Policy Consumers interpret and may further validate the Policy Rules and then map these rules to the underlying policy mechanisms of the Policy Targets. The Policy Consumers may also transform the Policy Rules into forms that Policy Targets can use directly. Policy rules are of the form: <trigger> if <condition> then <action>.

[Need more detail] <Trigger> indicates . . . The <condition> expression may be a compound expression and it may be related to entities
such as hosts, applications, protocols, users, other system sub-components, etc. The <action> may be a set of actions that specify services to grant or deny or other parameters to be input to the provision of one or more services. The set of actions associated with a Policy Rule may be ordered or unordered.

The policy information model, as described in [MODEL], is a platform- and technology-independent object-oriented model that describes not only the structural characteristics of a set of managed objects (e.g., users, network devices, and services) but also describes the relationships between those objects. This information is then mapped to a form that is suitable for storage in a particular repository, such as a directory.

Policy Consumers may detect changes in Policy Rules by periodic polling of the repository, by use of event notification mechanisms when changes occur, or by some other schedule or 'push' mechanism.

3.4. Policy Mechanisms

Policy Mechanisms are defined as the underlying methods, protocols, and tools used to perform the actual implementation (evaluation and action execution) of the Policy Rules. Usually, the Policy Consumer translates the Policy Rules and generates appropriate instructions for the Policy Target. Sometimes, the Policy Consumer simply identifies the appropriate Policy Rules for a given flow or environment and passes them to the appropriate Policy Targets. These policies are then evaluated and enforced by the Policy Targets as appropriate for a given event. In either case, these policies may be discipline-specific and, perhaps, element-specific. Typical uses are [grid examples], or a QoS policy.

It is not in the scope of this framework to specify actual mechanisms, but to provide a common interface through Policy Rule abstraction for access to the actual mechanisms.

3.5. Options for Packaging

As indicated earlier, Table 1 represents the functional elements of a policy framework, not actual products. A policy product may implement exactly one of the functional elements, more than one functional element, or even a part of one of the functional elements. Figure 1, below, shows a multi-role policy server that includes both a Policy Management Tool and a Policy Consumer. The implementation details of these two elements are hidden inside the server's boundaries. The only interfaces visible outside the server are the Policy Management Tool's user interface, the Policy Consumer's protocol for communicating with the Policy Targets, and the interfaces via which the Policy Management Tool and the Policy Consumer communicate with the Policy Repository.

Note that a given product may need multiple repositories to efficiently store and retrieve data that is used to make policy decisions. For example, if the main repository is a directory, and SNMP information needs to be used as part of a decision, then it would be a bad idea to store the values of (for example) SNMP counters in a directory. However, the directory could be used to specify the location and access method (for example, via a URL) of other data.

The line formed by asterisks in Figure 1 illustrates how the manufacturer of a multi-role policy server might add additional communication paths for transfer of policy information within the server. This is not meant to imply that not having this line of communication is better or worse than not having it. It simply describes a manufacturer's packaging option. In such a case, the line formed by asterisks represents a path by which policy information input at the user interface can be sent directly to the embedded Policy Consumer, without first having to be placed in the Policy Repository. Since this line is within the server, it has no bearing on interoperability.

In the future, it may be determined that this communication path should be standardized between separately packaged Policy Management Tools and Policy Consumers. Currently, the line represents an internal product interface, for the simplicity of standardization, and to gain experience to provide the above interfaces.

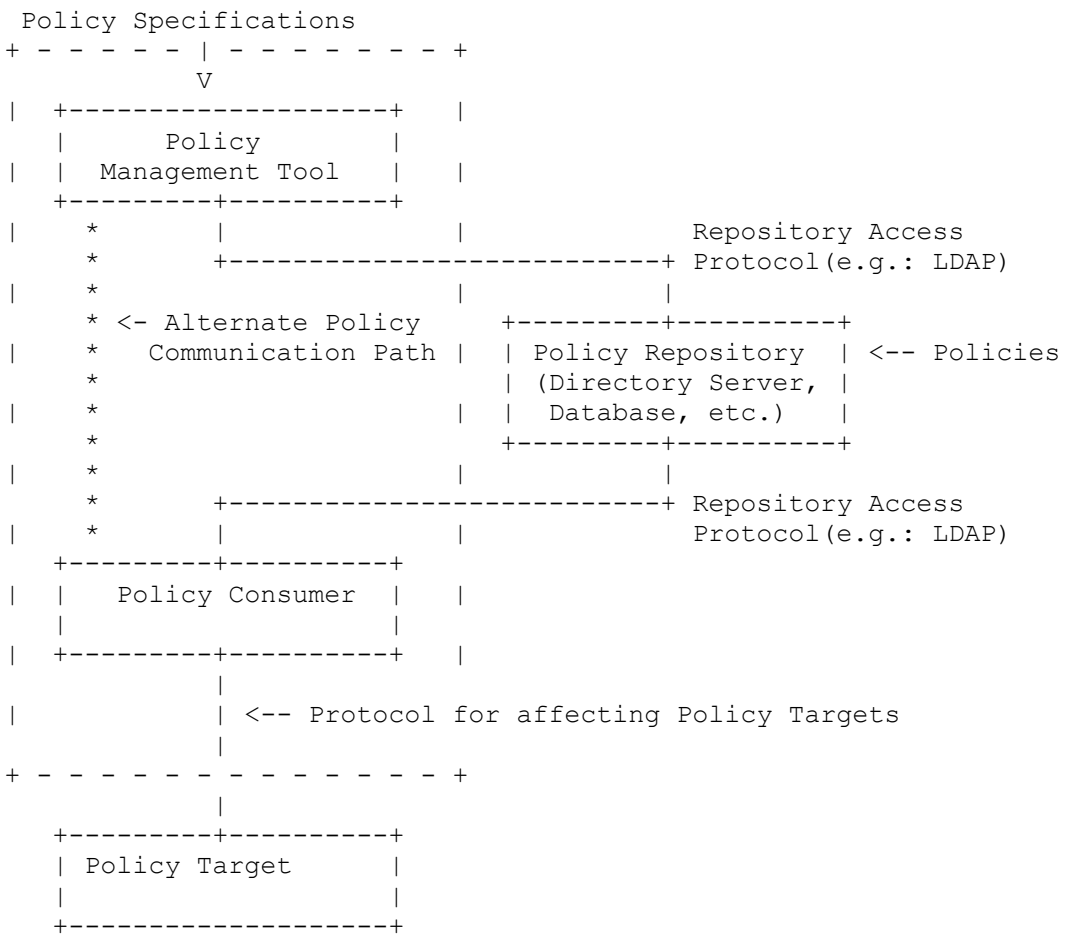

Figure 1. A Packaging Option for the Functional Components

4. Functional Groupings

A policy system implementation can be composed of the four functional entities, shown in Table 1. Some of these components have previously appeared in IETF drafts in different shapes and forms, as a Policy Server [IPSEC] [DIAMETER], as RAP's PDP and PEP [RAPFRAME], and as bandwidth brokers. It is important to separate the functional components and describe the relationships between them. That is the purpose of this section.

Note: This is an enumeration of policy functions and, as such, it does not describe any implementation details such as distribution, platform, or language. It merely shows an example of convenient groups of functions within a policy system.

4.1. Policy Infrastructure

Likely a sub-component of the Policy Management Tool, the Policy Editor provides the policy editing, policy presentation, rule translation, and rule validation functions. In this component, many rules are translated from abstract or human understandable forms to the syntax of the policy information model of the repository. Basic syntactic and semantic validation is also possible.

The Policy Consumer is responsible for Policy Rule interpretation and initiating deployment. Its responsibilities may include trigger detection and handling, rule location and applicability analysis, network-, grid- and resource-specific rule validation, and element adaptation functions. In certain cases, it transforms and/or passes the Policy Rule and data into a form and syntax that the Policy Target can accept, leaving the implementation of the Policy Rule to the Policy Targets.

Policy Targets are responsible for the evaluation of policy conditions and Policy Targets handle the execution of actions. These entities may also perform related element-specific functions, such as Policy Rule validation and policy conflict detection.

These functions are reviewed in the following sections.

4.1.1. Policy Editor

The Policy Editor is a mechanism for entering, viewing and editing Policy Rules in the Policy Repository. Grid and policy administrators would use the Policy Editor. It could be implemented as a full-featured graphical user interface, a simple Web-based form, and/or support a command line or scripting interface.

Policy Rules may be of several forms. Probabilistic rules may be of the form: "with 99% probability, provide sub-second response-time for department D using application A." Other general specifications may be of the form: "I want 100Mb/s of premium traffic going from point A to point B" or "I want a video conference between the sales team managers". Alternately, more specific rules may be defined, such as "For NetworkDevice1 and 3, configure drop queues as follows ..." or "From source=10.56.43.x to destination=10.56.66.x, enable Premium Service" for these types of traffic. There is wide latitude in the level of abstraction and function of the policy editor UI. However, it is beyond the scope of this document to specify such functions.

[Discussion of other grid examples, policy forms and languages]

Once a Policy Rule has been entered into the Editor and before it is stored in the repository, simple validation should be performed. This is described in 4.1.3. The Policy Editor should provide feedback to the administrator of the validation results. At the simplest level, this could result in a "Valid/Invalid Policy" message. More useful, however, would be for the Editor to indicate the erroneous rule conditions or actions, or display the pre-existing Policy Rules with which the new rule conflicts. Further rule definition or update would then be the responsibility of the administrator. However, it is beyond the scope of this document to specify such functions.

4.1.2. SLO Translation

Translation of general policy specifications or SLOs into Policy Rules that the Policy Consumer can interpret is performed by the Rule Translation function. This function maps a high level (e.g., business-oriented) specification of a Policy Rule, with its associated parameters, to a more specific Policy Rule format pertaining to that service.

It is expected that further processing for grid environments will be defined. Using QoS as an example, the Rule Translator would take general Policy Rules related to the specification of "normal" or "premium" service, and translate these to the specific format defined for the grid. This format is element- and platform-independent and could be performed by the Policy Management Tool. Note, however, that another level of translation is usually necessary to enable an element to interpret and execute the policy. The Policy Consumer does this form of translation.

As another example, the following Policy Rule could be defined: "Traffic between Point A and Point B should receive Expedited Forwarding". This could be translated into the following two Policy Rules:

- source = 10.24.195.x, destination = 10.101.227.x, any protocol, provide Expedited Forwarding

- source = 10.101.227.x, destination =10.24.195.x, any protocol, provide Expedited Forwarding

where the action to perform Expedited Forwarding is enabled through the marking of packets with the Differentiated Service Code Point (DSCP) of 101110. In this example, the network has been configured to treat packets with a DSCP 101110 as packets that receive Expedited Forwarding treatment. Thus these rules apply to the ingress interface for the network, on either an end system or a router, where packets will be marked.

4.1.3. Rule Validation

The Rule Validation function performs checking of a policy prescription and/or rule, and returns the results of this checking. Two kinds of checking should be performed:

- Validation of the data types of the terms of the specified Policy Rule. For example, if a policy term calls for the input of an IP address, then the system should ensure that a valid IP address and mask are specified (as opposed to, for example, an integer).
- Validation of the semantics of the Policy Rule. This has to do with ensuring that the construction of a Policy Rule, and its conditions and actions, from a set of pre-defined building blocks, actually makes sense. Policy rules can be syntactically correct yet make no sense. For example, a rule may be defined stating that "Traffic at 50 Mb/s should receive Expedited Forwarding treatment and run only between 10.23.24.56 and 10.23.24.56". This is syntactically valid but semantically wrong since it specifies the same source and destination address.

4.1.4. Global Conflict Detection

The Global Conflict Detection function checks to see whether or not a newly entered policy conflicts with other policies. It is called "global" in order to connote that this type of conflict detection is not bound to any specific device, subnet, network or grid.

The Global Conflict Detection component checks for static conflicts derived from Policy Rules whose conditions are simultaneously satisfied, but whose actions conflict with those of currently existing rules. For example, an administrator may define two rules stating that "A maximum of 10 video conference channels are allowed on NetworkA", and that "eight video conference lines are dedicated to Finance on Tuesdays from 9-10am". If a third rule provisioning 3 video conference lines for Legal every day at 9-10am were to be added to the rule set, a conflict should be detected. The administrator is attempting to provision for 11 video channels (versus the maximum of 10 channels allowed).

Not all policy conflicts can be detected by the Global Conflict Detection function. Rules may be "time based" (specifying an effective validity period in the future) or based on dynamic state information. These rules may indeed conflict with others. But, these conflicts may only be detected at the time that the rule becomes valid and enforcement actions are attempted. For example, one may have Policy Rules that apply in normal, congested, and business-critical (e.g., financial crisis, take away all bandwidth from everywhere to support this) conditions. On the surface, they appear to conflict with each other. However, in reality, they don't, since they are meant to apply in non-overlapping time periods and conditions.

The validation performed in this component is also called off-line validation, meaning that it is not performed at the same time as the execution of the policy. On-line validation occurs within the Policy Assessment component, discussed in Section 4.4.

4.2. Rule Storage and Retrieval

Once a Policy Rule has been translated and verified, its storage in one or more Policy Repositories is required. This may be done before or after the Policy Consumer starts processing the Policy Rule. Utilization of Policy Rules to maintain or change system/device state requires retrieval of these rules from the Policy Repositories. In addition, the repositories are accessed during the rule validation process discussed above.

4.3. Policy Consumer Functions

4.3.1. Changing Policy

Data in the Policy Repository will change from time to time and Policy Consumers need to be informed of these changes as they happen. This framework does not specify the means for notifying Policy Consumers of changes. There are a number of possible ways to make this notification (e.g., polling, LDAP change notification, using SNMP traps, etc.).

4.3.2. Evaluation of Policy Conditions

Evaluation of policy conditions may involve the Policy Consumer, the Policy Target, or both the Policy Consumer and the Policy Target. When evaluation applies to a single element, or when it applies to detailed conditions that only the Policy Target (as opposed to a physically separate Policy Consumer) can understand, then condition evaluation will typically occur only in the Policy Target. At the other extreme, if only global conditions such as time of day or the overall state of the grid are being evaluated,

then the condition evaluation may take place entirely in the Policy Consumer. In many cases, though, the evaluation of policy conditions may be shared between the Policy Consumer and the Policy Target.

An example will show clearly how a Policy Consumer and a Policy Target might share policy condition evaluation. The Policy Consumer in this example is one that translates policy rules into configuration settings, and then downloads these configuration settings to its Policy Targets. Such a Policy Consumer might have retrieved from the Policy Repository the following two rules for one of its Policy Targets:

- Rule 1: If there is overall congestion in the network, then drop packets received from subnet-1.
- Rule-2: In there is not overall congestion in the network, then accept and process packets received from subnet-1.

"Overall network congestion" in these conditions does not indicate a single interface's or single device's understanding of the current state of the network. Instead, it refers to an understanding of the state of the network as a whole, which might involve a management application (the "congestion application") that interacts with various probes in the network, and/or introduces artificial traffic into the network and measures the progress of this traffic. In this simplified example, this application would need to provide a binary answer ("Yes, the network is congested" or "No, the network is not congested") to the Policy Consumer.

Based on whether or not the network is currently experiencing congestion, the Policy Consumer acts. If the network is congested, then the Policy Consumer downloads to the Policy Target a set of configuration parameters that will cause it to drop packets from subnet-1. If the network is not congested, then the Policy Consumer downloads a different set of configuration parameters, that cause the Policy Target to process packets from subnet-1.

This initial configuration download is not the end of the Policy Consumer's responsibilities in this case. After this, it must continue to interact with the congestion application, and be ready to download new configuration parameters to the Policy Target if, in the opinion of this application, the network becomes congested, or ceases to be congested.

As an implementation option, a Policy Consumer may elect to cache the congestion application's opinion about whether the network is congested, so that it can quickly determine which configuration settings to download if a new Policy Target contacts it. This sort of cached data would typically be stored locally by the Policy Consumer, as opposed to being stored in the Policy Repository.

4.3.3. Element Adapter and Execution of Policy Actions

The Element Adapter function has two distinct purposes. One purpose is to take the canonical representations of Policy Rules (as stored in the Policy Repository) and interpret them on behalf of elements not equipped to interpret them directly. In this case, the element adapter function can be realized as a Policy Consumer which is effectively a proxy, enabling legacy objects to participate in the implementation of policy within a given grid environment, without having to retrofit the legacy elements.

The second use of the Element Adapter function is to relieve the Policy Consumer of having to know all of the intimate details of the Policy Targets that it controls. The problem is that a given grid may contain many different types of elements, each with different capabilities. One example is that a single configuration can not be given to different elements. Instead, the element configuration will vary as a function of vendor, element type, protocol used, and other factors.

The problem is that many vendors make so many products, that it becomes impossible for a single Policy Consumer to be able to control all of them, due to their differing interfaces and capabilities. Operating in a multi-vendor grid exacerbates this problem. The solution is to develop a set of extensions to the Policy Consumer that are able to individually translate the policy generated by the Policy Consumer to an equivalent form that is usable by a specific set of elements.

4.3.4. Transformation

There are in general five models for sending a policy from the Policy Consumer to a Policy Target. These models support the different types of grid devices described in section 4.5 below. They are:

- Pass-Through. Simply pass the policy retrieved from the Policy Repository to the Policy Target directly, and let the Policy Target interpret, evaluate and execute it.
- Modify-Transform-Send. The Policy Consumer interprets and evaluates the policy (possibly adding some data or changing some parameters in the process) and then ships the modified form of the policy to the Policy Target, which then evaluates and executes the modified policy.
- Command-Transform-Send. The Policy Consumer interprets and evaluates the policy, and then generates a set of commands that the Policy Target can use to implement the policy.
- Proxy. The Policy Consumer must use a Policy Proxy to be able to communicate to the Policy Target.
- Co-Location. The Policy Consumer and the Policy Target are co-located in the same physical platform, and internal, programmatic interfaces are used.

4.3.5. Local Conflict Detection

The Local Conflict Detection (LCD) component is an integral part of the Policy Consumer. Whereas the Global Conflict Detection components check for policy conflicts that do not apply to any specific grid element, the LCD checks for policy conflicts that apply to all elements that are controlled by a given Policy Consumer.

The LCD detects local conflicts and checks that the requirements of the policies can be satisfied and assesses the feasibility of a policy (new, changed, or deleted) in which this Policy Consumer has interest. The types of checks performed include:

- Conflict Detection. This entails checking that the new, modified, or deleted policy does not conflict with any existing local policy.
- Requirements Checking. This is a set of checks to ensure that the resources needed by a policy, in isolation from all other local policies, are available in the elements to which this policy applies. For example, suppose that a policy requires that a certain set of paths through the network provide a specific queuing behavior. Suppose further that on one of the paths at one of the interfaces, no advanced queuing mechanisms are available. This would mean that the needs of the policy are not satisfied. Thus, the policy itself is not satisfied, implying that this policy cannot be implemented in these devices.
- Feasibility. This compares the available services of the grid with respect to the full set of policies that want to use those services. Feasibility checking will most likely require post-policy deployment checking that is sensitized to the particular grid elements involved as well as the nature and effects of the deployed policies. This is beyond the scope of this document.

4.4. Policy Assessment

4.4.1. Assessment of the Feasibility of Policy Rules

A set of Policy Rules may be infeasible for reasons other than being in conflict. Resource availability and the state of the network may render Policy Rules impracticable.

Such assessment of Policy Rules may involve multiple components (e.g., the Policy Consumer and the element). When assessment applies to a single object, or when it applies to detailed state or operational conditions that only the Policy Target (as opposed to a potentially, physically separate Policy Consumer) can understand, then communication is required, or this function must execute within the Policy Target.

The Validation components gather, (optionally) store, and monitor grid state and resource information. Upon a request to evaluate a Policy Rule set, the Validation function uses this information and returns a determination as to the feasibility if the Policy Rule.

Often, authentication and authorization checking are required of the Validation components. Examples include checking the current time of day against the authorized times that a user or application can access certain resources, or checking against the level of service that a user or application can request.

State and resource validation is also concerned with the current availability of grid resources. In other words, the services and/or resources requested must exist in the quantity required. If requested resources are available, then the actions of a Policy Rule may be executed. The notion of current resource availability is dynamic and depends on how resources are currently provisioned in the grid, and what resources are presently in use by, or reserved for, other uses.

4.5 Policy Execution

It is important to understand that the critical point of interoperability with regard to grid policy resides in a realized information model, rather than in a transport protocol and its message semantics. Instances of the classes described in the IETF's Core Policy Information Model [PCIM] contain data that describe operational policies. To affect policies in the grid, entities within the grid must interpret prescribed policies. Not all entities necessarily possess the ability to interpret policies directly. Such entities may require translation and transformation assistance with respect to the Policy Rules.

Grid devices, resources and services be categorized into groups: policy-aware and policy-unaware. Policy-unaware elements are unable to interpret any portion of the policy information model or schema. However, they can still participate in a policy-based network if an application can translate the policies into a form that the policy-unaware element can implement. It is irrelevant that the policy-unaware object doesn't know it is executing policies; what is relevant is that it is participating in a policy solution.

Policy-aware elements fall into four groups: policy-interpretive, policy-compliant, policy-capable, and policy-proxied. Policy-interpretive elements have the capability to interpret expressions of policy as represented in a repository, conforming to the core policy schema. For example, an interpretive device that possesses the capability of delivering specified quality of service may also understand a QoS policy schema, and interpret and enforce those policies without the aid of an external application.

Policy-compliant elements can interpret portions of the policy schema. In the case of QoS policy schema, a policy-aware device possesses the capability to interpret the classes WITHIN the policy schema that describe vendor- and implementation-independent expressions of QoS. Policy-compliant elements cannot interpret the QoS rules, but can interpret QoS parameters as defined by QoS classes within the schema.

Both policy-capable as well as policy-proxied elements are those that can not directly interpret policy as defined in a policy schema or information model. An intermediate process must be used in both cases to transform the policy as stored in the Policy Repository to a form that can be executed by the Policy Target. The difference is that a policy-capable element can communicate directly with the Policy Consumer, whereas the policy-proxied element requires a proxy to communicate with the Policy Consumer.

Both policy-unaware and policy-aware elements require assistance in interpreting policies. A Policy Consumer is an example of an application that can assist both policy-unaware and policy-aware elements by interpreting policies. In the case of policy-unaware elements, a Policy Consumer may find it necessary to transfer policy and other related information to and from an intermediate process (which then talks directly to the entity), in order to affect policy. Typically, the Policy Consumer may need to read or write configuration and read state information associated with a given entity, as it prepares that entity to implement specific policies. Recommending a specific protocol or mechanism for the purpose of establishing communication from Policy Consumers to policy-unaware elements is beyond the scope of this document. It is recognized that a number of options exist.

A MIB is an instance of a data structure that describes configuration and state information of a device or service. Therefore, a Policy Consumer could use a MIB for configuration and for discerning state of the policy-unaware devices or services with which the MIB is associated. Another option may be to use a command-line interface. In such cases, a Policy Consumer can use the command-line interface to configure a device as needed. Note that in both of these cases, control may be effected through the use of an intermediate process. In this case, the role of the Policy Consumer changes to communicating its requests to the intermediate application, which is then responsible for communicating with and controlling the appropriate elements on behalf of the Policy Consumer.

In the case of policy-aware elements, a Policy Consumer still must transfer policy information to and from elements. Since the element is capable of interpreting certain classes defined within the policy schema, the Policy Consumer may not need to use a MIB or command-line interface to configure a device. Instead, a Policy Consumer could use any protocol to transmit instances of the policy schema classes that represent the desired operation, and let the Policy Target perform the actual configuration. Examples of protocols include COPS, SNMP, and Telnet.

It is possible that the Policy Consumer resides with the Policy Target, such that there is no network connection between the Policy Consumer and Policy Target. In such a case, the function of communication between the Policy Consumer and Policy Target is completely implementation dependent since there is no interface that must be exposed. In such an implementation where the Policy Consumer and Policy Target reside within the same system (e.g., a network element such as a router), the Policy Consumer and Policy Target may even be simply different objects or functions within a single process.

As we develop policy systems in grids, we must be careful to distinguish protocol- and element-specific components from information model components. At the level of the information model, the schema reflects abstract, general information. The core policy information model is designed to enable interoperability, and a proprietary device-, resource- or service-specific data structure reduces interoperability. Any mapping between standardized expressions of policy and the parameters of proprietary algorithms should take place in the application responsible for and capable of interpreting policy (e.g., a Policy Consumer).

Furthermore, more than one Policy Consumer may need to share information represented in the standard schema. Using an appropriate policy protocol, policy-aware entities may express objects of the information model in a variety of agreed-upon formats (yet to be defined) and transmit them as necessary. [Need details]

4.6. Applying and Deploying Policy Rules via Roles

The specific policy to apply to an element may depend on many factors, including the physical and/or logical characteristics of the element, its status, user configuration parameters, or other parameters such as time of day, geographical location, and function in the grid. Rather than specifically tying policies to an element, policy applicability can be specified indirectly, via "roles".

Roles provide a powerful means of indirection:

- New or modified policies are associated with a role, instead of having to associate these policies individually to each and every element in a grid
- Existing policies are applied to newly-installed elements by assigning the relevant roles to the element, rather than copying policies from existing, "similar" objects
- Roles enable administrators to generate grid-wide policies, rather than having to remember all the individual components to which policies should be applied
- Neither the permanently-stored policy data, nor the Policy Consumer, needs to have intimate knowledge of each and every element in a grid; rather, each element can inform the Policy Consumer of the roles for which it needs policy

Roles are labels that are used to pass policy information between the Policy Consumer and the Policy Target. Roles abstract element capabilities, and are useful for aggregating entities to apply a common set of changes to without having to name specific entities. For example, this enables "all Frame Relay Edge interfaces" to be provisioned in one operation, instead of individually.

A given element may have multiple roles associated with it. This simply means that an element performs many separately identifiable functions in the grid.
[Need details from current models and thinking, compare and contrast IETF, DMTF and SNIA]

When the Policy Consumer and the Policy Target first connect, the Policy Target could report the roles that it supports to the Policy Consumer. This enables the Policy Consumer to determine which policies are applicable to the Target. For example, if a device has five interfaces with roles A and B, and four interfaces with roles A and C, then it must request policy data for two roles: A+B and A+C. The Policy Target also reports changes to its roles to the Policy Consumer.

4.7. Interfacing with Components Outside the Policy Infrastructure

4.7.1. Grid Management Products

Existing management products can play an integral role in comprehensive policy systems. A management product can be used to configure grid elements based on the definition of Policy Rules. In such a case, a management product can become a Policy Consumer or provide services to one. It can be used by a Policy Consumer to install element-specific mechanisms that implement Policy Rules. A monitoring application provided by the management product could be used for independent policy verification.

A management product would also be useful for accessing the grid inventory and topology. This information is critical for making certain types of policy decisions.

5. Policy Conflicts

A policy conflict occurs when the conditions of two or more Policy Rules are concurrently satisfied but the actions that they mandate produce inconsistent results with each other. For example, a Policy Rule specifying that "all engineers get bronze service" is in conflict with another rule defining that "the lead engineer gets gold service". This is a direct conflict, since there are directly identifiable terms in each Policy Rule that conflict. However, there are also indirect conflicts, such as with this third rule: "all ftp traffic gets best effort". This conflicts only if an engineer decides to send FTP traffic.

A conflict may be determined before execution of the policy is attempted. This function is represented in policy systems via two different mechanisms: (1) "Global Conflict Detection" (section 4.1.4) and (2) "Local Conflict Detection" (section 4.3.5). For example, the conflict may be detected by the Global Conflict Detection component when the policy is entered into the Policy Editor. Alternatively, the conflict may go unnoticed until the Policy Target tries to validate or implement it. A different type of conflict may also be determined when the policy is processed at the Policy Consumer. An example of this type of conflict is when one Policy Consumer loads a policy into a Policy Target and a second Policy Consumer attempts to load a conflicting policy.

Conflict detection is an important aspect of a policy infrastructure. Various mechanisms and degrees of sophistication exist in implementations.
[Need details]

6. Interoperability

The framework outlined in this memo defines two types of entities that access the data repository: the administrative tools and the policy consumers. Both of these entities require interoperability with the data repository on at least two levels.

The first level of interoperability is on the data model level. The entities require knowledge of the structure, syntax, and semantics of the data in order to be interoperable. Failure to fully comply with any of the data definitions will cause an entity to produce incorrect results.

The second level of interoperability is on the data access level. For the specific case of a directory used as the repository, the policy framework would implement LDAPv3 (or higher) as the protocol to access the repository. Assuming a compliant LDAPv3 implementation, data access should be interoperable. Other access protocols are suited for alternative repositories.

It should be noted that various repository infrastructures may not be sufficient to ensure data repository interoperability. The following features are required for data interoperability:

- Change notification: the ability to notify data access entities when data changes and how the data changed;
- Transactional integrity: the ability of the repository to ensure that a set of related operations are completed as a set; and,
- Referential integrity: the ability of the repository to ensure that a given operation applied to one object affects related objects in the appropriate way

Interoperability problems will occur if implementations choose to use proprietary change notification mechanisms or implement notification in a non-consistent fashion. Lack of transactional and referential integrity will result in interoperability problems since implementations may update objects in different orders, or fail to apply certain operations to all objects. This could cause data repository corruption.

7. Future: Inter-Network and Inter-Domain Communication

The inter-domain communication interface of a policy management system is concerned with communication with other policy systems in adjacent domains. This communication may be across enterprise-carrier or carrier-carrier boundaries. The primary purpose of inter-domain exchanges is to negotiate SLAs with adjacent networks to establish policy services within the adjacent network. Ideally, the adjacent grids and networks should have sufficient SLAs in place with their downstream neighbors to support the requested service end-to-end.

Adjusting provisioning at domain boundaries entails re-negotiation of SLAs with adjacent domains. Linking provisioning with policy management makes it possible to manage how provisioning is performed.

The area of inter-domain communication for policy service requests is an ongoing research topic. Protocol requirements, message contents, etc. are still under study within several IETF working groups including RAP, DiffServ, Policy, and AAA working groups.

8. Application/Mapping of Grid Policy Use Cases to the Framework

[Insert details]

9. Acknowledgements

The original authors of the first draft of this document are acknowledged both here and in the Author's Section: M. Stevens, W. Weiss, H. Mahon, B. Moore, John Strassner, and Glenn Waters. The first version of this document was an Internet-Draft, published in the Policy Framework Working Group of the IETF. Although that draft expired, work has resumed in the grid community of GGF.

10. Security Considerations

The implementation of a policy infrastructure 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 (Policy Management Tools, the Policy Repository, Policy Consumers, and Policy Targets) involved in policy-based management can verify each other's identity and establish necessary trust before communicating.

[Need more discussion]

11. References

[Need normative references]

12. Authors' Information

[Insert contact info]