Simplifying Network Administration Using Policy-Based Management

Dinesh C. Verma,
IBM Thomas J Watson Research Center

Abstract

The management of network infrastructure in an enterprise is a complex and daunting affair. In an era of increasing technical complexity, it is becoming difficult to find trained personnel who can manage the new features introduced into the various servers, routers, and switches. Policy-based network management provides a means by which the administration process can be simplified and largely automated. In this article we look at a general policy-based architecture that can be used to simplify several new technologies emerging in the context of IP networks. We explain how network administration can be simplified by defining two levels of policies, a business level and a technology level. We discuss how business-level policies are validated and transformed into technology-level policies, and present some algorithms that can be used to check for policy conflicts and unreachable policies. We then show how to apply this architecture to two areas: managing performance service level agreements, and supporting enterprise extranets using IPSec communication.

Present-day IP networks are large complex systems consisting of many different devices. Ensuring that all these devices interoperate smoothly is not a trivial task. New technologies that have emerged to address some of the limitations of the traditional IP protocols have added to the complexity of the network infrastructure. There is an acute shortage of experts who understand the new technologies and are able to manage and deploy them on a large network. For many emerging technologies, the management costs associated with deploying the technology outweighs the advantage conferred by that technology. As an example, many pragmatic network operators choose to overengineer their networks to address any performance concerns rather than deploy bandwidth-saving quality of service techniques. This is because the manpower cost associated with learning the new technologies and managing them is much higher than the savings in bandwidth-related costs that would result from deploying these technologies.

In this environment there is a clear need to make new and emerging technologies easier to manage. The policy framework being standardized within the Internet Engineering Task Force (IETF) [1] holds the promise to deliver this ease of management. The simplification and automation of the network management process is one of the key applications of the policy framework. This article explains how the policy management framework can help network administrators attain this simplification, and demonstrates its applicability in three different disciplines.

The following section provides an overview of the general policy-based administration architecture. The next section discusses structure of a management tool that can apply to many policy disciplines, and the algorithms needed within the management tool, as well as some policy validation algorithms that can be used to check for consistency and usability of network policies. After that is shown how the generic framework can be applied to the areas of service level agreement enforcement and secure IP communications. Finally, summaries and conclusions are presented.

General Policy-Based Administration Architecture

The general policy-based administration framework we present can be considered an adaptation of the IETF policy framework to apply to the area of network provisioning and configuration. The IETF/Distributed Management Task Force (DMTF) policy framework is shown in Fig. 1, and consists of four elements: the policy management tool, policy repository, policy decision point, and policy enforcement point.

An administrator uses the *policy management tool* to define the policies to be enforced within the network. A device that can apply and execute the different policies is known as the *policy enforcement point* (PEP). The *policy repository* is used to store the policies generated by the management tool. In order to ensure interoperability across products from different vendors, information stored in the repository must correspond to an information model specified by the Policy Framework Working Group. A policy enforcement point uses an intermediary known as the *policy decision point* (PDP) to communicate with the repository. The PDP is responsible for interpreting the policies stored in the repository and communicating them to the PEP. The PEP or PDP may be in a single device or different physical devices. Different protocols are to be used for various parts of the architecture (e.g., COPS or SNMP can be used for PDP–PEP communication). A repository could be a network directory server accessed using LDAP.

0890-8044/02/$17.00 © 2002 IEEE

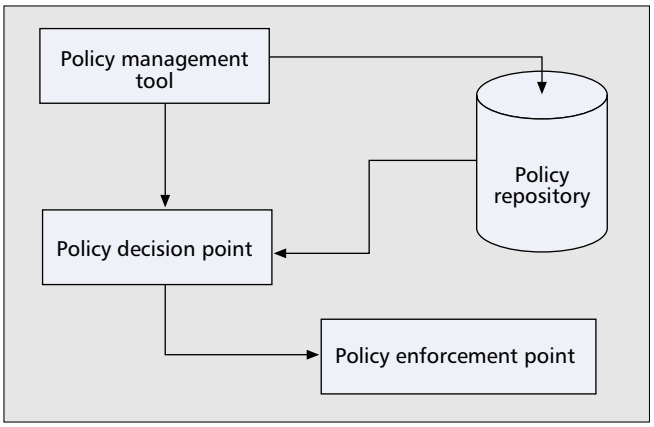

Figure 1. *The IETF/DMTF policy framework.*

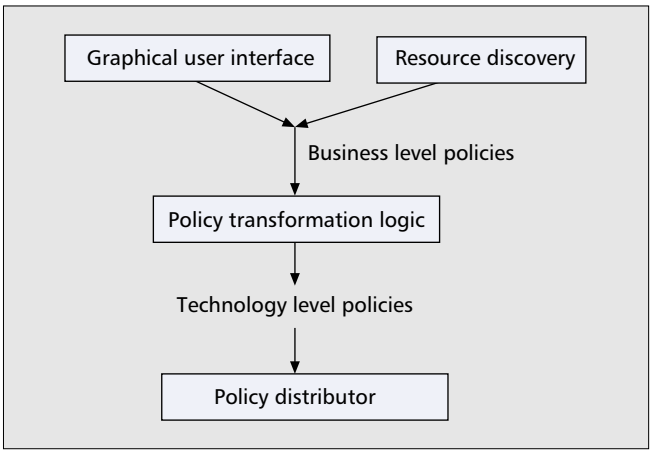

Figure 2. *A generic policy management tool.*

The structure of the management tool is not defined by the IETF standards since a standard format is not needed for interoperability between different machines. In this article we focus on the policy management tool and how it can leverage the power of policies to simplify the provisioning and configuration of the different devices within the network. This simplification of the management functions is obtained via two elements of the policy management tool and the policy architecture: centralization and business-level abstractions.

Centralization refers to the process of defining all the device provisioning and configuration at a single point (the management tool) rather than provisioning and configuring each device itself. In a system with a large number of machines, the centralization of configuration at a single node reduces the manual effort required of an administrator. The administrator inputs the policies needed for network operation into the management tool that populates the repository. The information in the repository is specified in terms of the technology being deployed within the network, for example, using terms and concepts available for managing performance using differentiated services (DiffServ) technology. The PDPs retrieve the policy defined in the technology-specific notation, and convert it into the appropriate configuration of the PEP that can enforce the desired policies.

The benefits of centralization in reducing manual tedium can easily be seen. In a network of 1000 machines needing 10 min of configuration per machine, an administrator would need to work over a week to configure the machines manually. With a policy-based solution, the administrator needs to spend about 15 min populating the repository with the appropriate policies, and the PEPs/PDPs would take care of the rest.

Business-level abstractions make the job of policy administrator simpler by defining the policies in terms of a language closer to the business needs of an organization rather than in terms of the specific technology needed to deploy it. The administrator need not be very conversant with the details of the technology that supports the desired business need.

As an example, let us consider the case of a network operator that needs to define two levels (premium and normal) of customers. It is fairly simple for an administrator to identify each customer and to define which level they may map to. One way to support the business need for bilevel customer support is to use IP DiffServ technology within the network. If the administrator wants to define policies using DiffServ concepts, he needs to be familiar with the technical details, for example, be aware that differentiation is obtained by assigning traffic to different per-hop behaviors (PHBs) and that the premium customer is mapped to a specific PHB (e.g., EF-PHB, expedited forwarding), and then define some parameters for this PHB. The jargon of the technology (PHB, EF-PHB, configuration parameter details) requires special knowledge that is harder to find than familiarity with the business needs of the organization.

The business-level abstractions depend on the business needs and the technology that the policies are being defined for. The business needs of an organization may be satisfied by many different technologies. A *policy discipline* is a two-tuple consisting of a business need and the technology that supports it. A business need, such as supporting performance SLAs on a network, may be satisfied by technologies such as capacity planning [2], integrated services (IntServ), DiffServ, or content distribution. A business need such as establishing a secure virtual private network may be satisfied using IP Security (IPSec) [3] or TLS protocol [4]. While each discipline would have some aspects of policies that are specific to it, many operations related to policies can be performed in a generic manner.

The Policy Management Tool

As discussed above, the policy management tool needs to support the notion of policies specified as business level abstractions as opposed to the policies specified as technology-level abstractions. A generic policy management tool that will support these two levels of policies can be constructed out of the four basic components, as shown in Fig. 2.

The *user interface* is the means by which an administrator can input the business-level policies within the network. The interface may consist of command lines or a graphical tool that can be used by the administrator. Command lines permit programmatic manipulation of the policy management tools.

The *resource discovery* component determines the topology of the network, the users, and applications operational in the network. In order to generate the configuration for the various devices in the network, the capabilities and topology of the network must be known.

The *policy transformation logic* component is responsible for ensuring that the high-level policies specified by the network administrator are mutually consistent, correct, and feasible with the existing capacity and topology of the network. It also translates the business-level policies into technology-level policies that can be distributed to the different devices in the network.

The *policy distributor* is responsible for ensuring that the technology-level policies are distributed to the various devices in the network. If the network devices support the IETF policy architecture, policy distribution consists simply of writing the technology-level policies (low-level policies) to the repository. If some devices do not conform to the IETF architecture, the distributor has to use alternatives, for example, convert the low-level policies into the appropriate device configuration, and configure the device over a command line interface.

The user interface, resource discovery, and policy distributor are components that have a relatively straightforward design. This is not to discount their importance; a good user interface can make the difference between an unusable tool and a great one. However, the heart of policy management lies in the policy translation logic: how the policies will be represented and managed.

The Policy Translation Logic

The policy transformation logic module validates the information provided in the high-level policies and transforms them into the configuration of devices in the network. The validation process must incorporate syntactical checks as well as semantic checks. The semantic validation of high-level policies consists of various types of checks:

- *Bounds checks*: validate that values taken by an attribute in the policy specification are within specific limits determined by the network administrator
- *Relation checks*: validate that the value taken by any two parameters in the policy specification satisfy a relationship determined by the specific technology
- *Consistency checks*: validate that any two policies defined by the administrator do not conflict with each other
- *Dominance checks*: check for "unreachable policies": policies defined by an administrator that will never become active in the network because they are rendered ineffective by the definition of other policies
- *Feasibility checks*: ensure that the set of policies desired by an administrator for a network are feasible in the operating environment provided by the network

Some of these checks can be made generically independent of the policy discipline. The translation process and feasibility checks depend on the policy discipline being supported. However, given a suitable policy representation, it is possible to perform the bounds, relations, consistency, and dominance checks in a discipline-independent manner.

Policy Representation

The high-level and low-level policies required for network management can be specified in many different ways. Among the researchers who are involved in specifying policies, multiple approaches for policy specification have been proposed. These approaches range from an interpretation of policies as programs to an interpretation of policies as simple entries in a directory or database.

From a human input standpoint, the best way to specify a high-level policy would be in terms of a natural-language input. Although these policies are very easy to specify, the current state of natural-language processing, a special area within the field of artificial intelligence, needs to improve significantly before such policies can be expressed in this manner.

The next approach is to specify policies in a special language that can be processed and interpreted by a computer [5, 6]. This maps a policy to a piece of software that can be executed by a computer under certain conditions. Another approach is to specify the policy using a formal specification language [7, 8]. When policies are specified as a computer-interpretable program, it is possible to execute them. However, in general it is quite difficult to determine if the policies specified by two different programs are mutually consistent.

A simpler approach is to interpret the policy as a sequence of rules, in which each rule is in the form of a simple condition-action pair (in an if-then-else format). The rules are evaluated on specific triggers, such as the passage of time or the arrival of a new packet within the network. If a rule's condition is true, the action is executed. A sample specification of the policy in this format would be "If the packet's source or destination IP address belongs to the research or engineering subnet, encrypt the packet." Policies specified in this fashion are easier to analyze than policies specified as full-blown computer programs or by a formal specification language.

Representing policies using if-then-else semantics may lead to the conclusion that policy representations should be evaluated using an expert system or theorem-proving approach. Although such an approach toward network policies is feasible, this may not be the appropriate approach for the problem. For the most important problems that are of relevance in an application of policies within the network, expert-systems-based approaches have several limitations. As an example, checking the mutual consistency of a set of policies using the theorem-proving approach requires exponential running time in many instances.

An alternative specification of policies is to represent them simply as entries in a table. The table consists of multiple attributes. Some of these attributes constitute the condition part, and others constitute the action part. Different types of tables need to be specified if the condition components or action components of different rules vary. Such a tabular representation is rich enough to express most of the policies that can be specified with a rule-based notation. Furthermore, it is easier to analyze for dominance and consistency.

The IETF [1] has chosen a rule-based policy representation in its specification. However, due to the need to store this representation in an LDAP directory or database, this representation essentially follows the tabular specification just described. For a variety of policy disciplines that arise in the field of TCP/IP networks, we have been able to use such a tabular specification of policies to capture most of the practical scenarios one may encounter.

Policy Validation Algorithms

When we use a tabular representation for the policies, each policy discipline can be characterized by a description of the set of tables in the policy definition for the discipline, and the set of columns that make up each table. We refer to this description as a *policy schema*. A column of a schema defines an attribute of the policy that could be a simple (textual or numerical) attribute, a structured attribute consisting of multiple attributes (e.g., an IP Subnet attribute consists of the simple attributes of SubnetAddress and Prefix Length), or a nested table (e.g., the table of interfaces at a computer). Different types of validation criteria can be associated with each table and column of the attribute.

The validation criteria enable some of the checks to be performed in a very simple way. By associating a limit checking criteria with each column, the bounds checks can be performed rather trivially. Similarly, the relations checks can be performed by defining a relationship criteria associated with a table. Each row in the table is validated against the relationship criteria.

The checking for policy conflicts and dominance needs to be performed across all rows of a table. The consistency criterion is that if two rules can both apply under some conditions, the actions to be performed must be uniquely identified and doable simultaneously.

Conflict Resolution — As an example of conflicts among different policies, consider a simple example where two classes of service (Gold and Silver) are defined within the network. An application called WebServer is defined to operate on TCP protocol and the port number of 80. A set of users with IP addresses in subnet 9.2.34/24 is defined as HighPowerUsers. Two policies are defined as follows:

P1: *Any access to WebServer gets Silver service.*

P2: *Any use of the network by HighPowerUsers gets Gold service.*

Both of these rules are perfectly okay by themselves, but there's a conflict when the two are taken together. In the case of a HighPowerUser who is trying to access WebServer, it is unclear whether the Gold or Silver service should be provided.

One approach to detecting conflicts among the different rules is to look on each policy rule as consisting of multiple independent terms and one or more derived terms. A policy rule consists of the generic form if-condition-then-action. Let us impose the restriction that the terms defining the condition be distinct from the terms defining the action portion of the rule. Business SLA policies as well as security policies are often defined in terms of classes of service (dealing with performance or security). Each class of service combines several actions that can be taken together.

Consider the different terms that make up the condition part of a policy expressed in the format if-condition-then-action. Each independent term can be looked on as an independent axis in a hyperdimensional space. Each rule defines a region in the hyperdimensional space. Each such region can be associated with a dependent term (e.g., the service class) identified by the rule. If any point in space has multiple dependent terms that conflict with each other, you have a potential conflict. For example, consider the case of policy definitions that have two independent terms. Each of the policy definitions would carve out a two-dimensional space. If the regions defined by two policies do not overlap, they do not conflict. If two regions overlap, the corresponding policies have a potential conflict if the dependent terms in the policy definition can't be done together.

For a more specific example, consider the simple example previously mentioned about HighPowerUsers and Webserver. The two independent axes in this case are the applications (identified by their port numbers) and the users, identified by their IP address. These two rules define regions in a two-dimensional space, with the first dimension being IP address and the second the port numbers. The first rule carves out a region defined by the line obtained by keeping the port dimension fixed at 80. The second rule is the square region carved out by subnet 9.2.34/24 (with a lower of range 9.2.34.0 and upper range of 9.2.34.255). The line intersects with the region, and the common region contains a conflict with two different classes of service.

More realistic cases of policy definition would tend to have more independent terms and corresponding dimensions defining the policies. However, if one defines the dependent and independent terms for each policy table, along with a function for each independent term that checks whether there is an overlap between two values of that term, the algorithm can be implemented in a very simple fashion with a running time of $O(n^2)$ where n is the number of policies. The use of computational geometry algorithms [9] may enable algorithms with better runtime complexity. If a conflict is found, one way to resolve it is to assign them different priorities. Since the priority can be considered an independent term for conflict resolution, policies with different priorities will not result in overlapping regions in the hyperdimensional space.

Dominance Checks — The dominance criterion checks whether a policy is actually applicable in some condition that can arise during operation. Dominance checks are also designed around the concept of the hyperdimensional space. To check for this, we map each policy into the independent and dependent terms as before. We also assume there is a function that takes two policies and determines which will dominate in a region of the overlap. With this information, the dominance check for a single rule consists of comparing it against all the other policies that overlap with it and dominate it.

We start with a list of hyperdimensional regions initially consisting of only one hyperdimensional region defined by the policy rule we are checking for dominance. Then we remove the region described by each dominating and overlapping policy from all the regions in the list successively. After all the policies have been compared, we examine the resulting list. If the list of hyperdimensional regions is empty, the policy is unreachable and can be removed without any penalties. Otherwise, the final regions represent the combination of independent terms where the policy would remain applicable.

The worst case running time of this algorithm is $O(n^{k+1})$ where n is the number of policies to be compared, and k is the types of independent terms that are used to define the hyperdimensional space. While this may appear rather inefficient algorithm, practical management systems are not likely to find it a problem. The number of independent terms is usually in single digits, and the algorithm is thus polynomial. In the context of QoS networking policies, k is usually 5 (corresponding to the 5-tuple in the network header). The number of policies in the system would be approximately 50 on the average, and the resulting comparisons can be completed in a few minutes by current processors. Another factor that helps considerably is the fact that the running time for a single policy dominance is $O(n_2 + n_1{}^K)$ where n_1 is the number of policies that overlap with the given policy and n_2 is the number of policies that do not overlap with the given policy. Since the number of overlapping policies is only a small fraction of the total number of policies, the expected time for checking the dominance of all policies is $O(n^2)$.

Discipline-Specific Procedures — The translation of business-level policies to a technology-level policy and the feasibility checks are discipline-specific procedures. The exact method to translate the business-level abstractions to a specific technology has to be defined on a per-discipline basis. However, the policy management tool provides a common framework within which the translation procedure can be performed. The common framework consists of defining the rules that provide the mapping from the tables defined as business-level policies to the tables that define the technology-level policies. Once these rules are defined for a discipline, the policy management tool can use them to perform the translation in a generic fashion.

As an example, let us assume that the business-level tables as well as technology-level translation are represented in XML. A representation of the table-driven policies is straightforward to do in XML. A different XML rule specification is needed for the business-level and technology-level abstractions. The translation rules consist of XSLT mappings that can be applied by the policy management tool. While the XSLT rules have to be defined on a discipline-specific basis, the management tool performs the translation in a generic fashion.

The feasibility checks also need to be performed in a discipline-specific manner. For the case of performance-related SLAs, the feasibility check require ensuring that the current network topology and the features of the technology being used can meet the desired performance goals. This may require the use of performance evaluation schemes. For the case of secure communications, the feasibility checks would require checking the compatibility of IPSec capabilities at two devices.

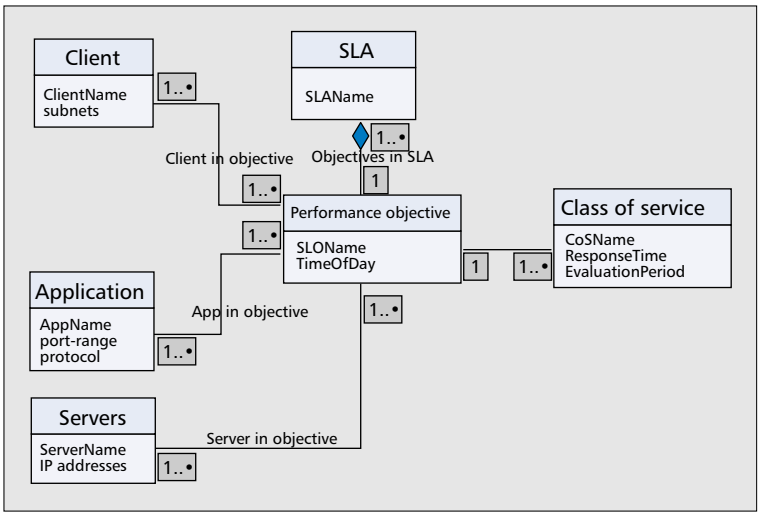

Figure 3. *An object model of business-level SLAs in an enterprise environment.*

Some Example Policy Disciplines

Having provided an overview of the generic policy management, we will now consider the policy management tool for a couple of disciplines. The two policy disciplines we will consider are:

- The support of performance-based SLAs using the IP DiffServ technology
- The support of enterprise extranets using IPSec protocol suite

For each of the policy disciplines we want to apply to the generic management architecture, we need to do the following tasks:

- Define the policy schema for the business-level policies
- Define the policy schema for the technology-level policies
- Define the discipline-specific translation rules
- Define the nature of any discipline-specific feasibility tests

For demonstrating the examples with policy discipline, we would consider the environment that of an enterprise network consisting of several campus networks connected together by means of wide-area links. We would use some simple policy schemas for both the business- and technology-level policies, illustrating them in UML.

The policy management architecture can be used for many other disciplines and in business environments such as a network services provider or an application services provider [10].

Service Level Agreement Using Differentiated Services

Within an enterprise environment, one of the components of the business SLAs enforced on the IT department specifies desired objectives for *application performance*, for example, "Mail messages less than 100 kbytes should be retrieved in less than a second." Figure 3 shows a very simple object model of the business-level performance SLA policies in an enterprise environment.

As shown in Fig. 3, an SLA[1] is an aggregation of several performance objectives. Each objective defines a association between a client, an application, a server, and a class of service. Semantically, a performance objective states that traffic flows used by a client to access an application running on a specific server be mapped to one of many classes of service. As an example, consider a client called Accounting, which is accessing an application called SAP running on a server called business server. A performance objective may state that accounting's access to SAP running on business server be given Gold class of service.

In Fig. 3, a *client* represents users within the network. Each client has two properties: a name (clientName) that identifies it uniquely, and the subnet address, identifying the location of the machines used by these users. A *server* is a machine where applications run, and its properties include a name and IP address(es) of its interfaces. We assume that applications run on well-known port numbers or a range of port numbers on each of the servers. Thus, an *application* has the properties of name, port range, and protocol. A *class of service* defines a level of performance. Its properties include a name, a response time, and an evaluation period. The response time is the expected application response time for any traffic flows that map into this class of service. The evaluation period states how long measurements must be taken in order to determine the response time. As an example, the Gold class of service may have a response time of 500 ms for an evaluation period of 1 h. Any traffic flow that maps into the Gold class of service is required to have a response time of 500 ms or less when averaged over intervals of an hour or more.

The *performance objective* provides an association between a client, an application, a server, and a class of service. An objective has an association with exactly one class of service, but the association could be with more than one client, application, or server. An objective could only be valid at specific times of day, which is one of the attributes shown for the performance objective in Fig. 3.

The low-level policies for the mapping would consist of the policy schemas defined for the DiffServ technology. An example of such a specification for DiffServ is shown in Fig. 4. Multiple devices with the same set of policies are mapped into a device role. For a device role, multiple network levels are defined, each network level corresponding to one of the many DiffServ PHBs that can be used within the network. The DiffServ policies map the traditional IP 5-tuple (consisting of the source and destination addresses, ports, and protocol) to one at the network level. While this set of DiffServ policies is very different than the framework defined by the IETF, the set of policies conformant to the object model shown in Fig. 4 can easily be mapped into the notation of the IETF workgroup [11].

Let us assume an expert user has defined the rules that specify the mapping of the classes of services defined as per Fig. 3 into the network levels defined as per Fig. 4. The expert user determines the PHB and the amount of bandwidth to be allocated to each class. Thus, an expert user may define that one should use the class selector PHBs within the network, with the Gold class of service to correspond to the highest priority service, the Silver class of service to correspond to the medium priority service, and the Bronze class of service to correspond to the default service, with a maximum bandwidth limit of 80 percent of a link's capacity to be used by applications in the default class of service.

The policy tool uses the network topology to determine the set of access routers and servers that are relevant for each business-level policy. For each device, the relevant rules are collected together. Devices with the same set of policies are collected together in a common device role. The management tool then uses the translation tables provided by the expert user to determine the correct marking behavior for all the

[1] *Some authors refer to the aggregation of performance objectives as service level specifications (SLS), and use the term SLA for business contracts incorporating SLS.*

servers and access routers. For the core routers themselves, the policy management tool must generate a consistent mapping of the ToS encoding to the right priority level along the forwarding paths of the routers.

Supporting Enterprise Extranets using IP-security

Enterprises establish extranets in order to automate their business processes with other enterprises(e.g., with their contractors and suppliers). An extranet allows a business partner to access part of the enterprise infrastructure. We assume that the following entities are involved in establishing the extranets we are examining here:

- *An extranet client application*: This client application runs in the demilitarized zone (DMZ) of a business partner, which is a supplier to the enterprise in this example. It runs on a machine that has support for IPSec. These types of machines are the only entities in the supplier's environment that are allowed to communicate to a set of servers within the enterprise.
- *An extranet server application*: This application runs in the DMZ of the enterprise, and communicates with the extranet client applications that are operational in the supplier's DMZ. The extranet server application runs on extranet server machines.

Furthermore, a policy management tool and a policy repository are required in compliance with the IETF policy architecture. It is assumed that the machines running the extranet client application and the extranet server application implement the IETF PEP and PDP functionality.

The UML diagram illustrating the business-level policy schema for enterprise extranets is shown in Fig. 5. An *extranet* definition allows a set of business partners to access a set of applications that are running on some servers within the enterprise. A *business partner* may have more than one machine at its site where the extranet client application is operational. Thus, the business partner contains an association to multiple *extranet client* machines. Each extranet client houses the extranet client application, and is identified by its name and IP address. The extranet definition allows some machines to become accessible to external business partners. These machines (the *extranet servers*) have a name and an IP address as their attributes. Only some *applications* running on the servers may be made accessible to external business partners. These applications are identified by their name, and their other attributes include the ports they run on and the protocol these applications use for communication. An extranet is access associated with one or more business partners, one or more extranet servers, and one or more applications. Each extranet definition allows extranet clients belonging to associated business partners to access associated applications executing on associated extranet servers.

Each extranet is associated with exactly one security class. The security class defines the type of security that needs to be provided to the traffic flows that form part of the extranet definition. The details of the security class are provided within the low-level technology-specific definitions, and are described in more detail in the following subsection.

All the extranet definitions taken together constitute the high-level policy in this environment.

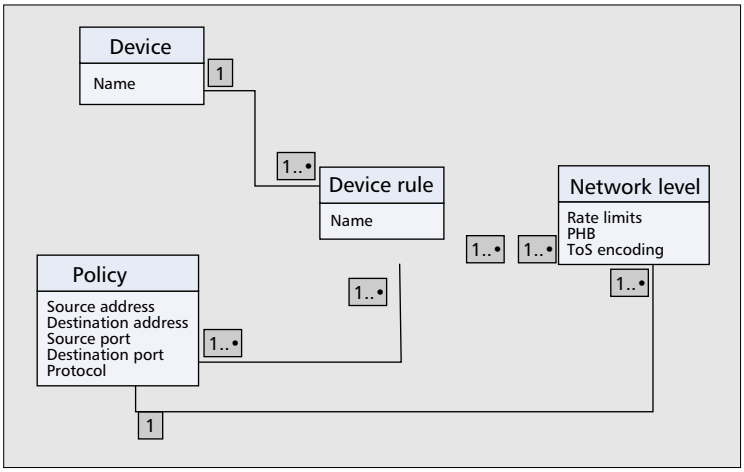

Figure 4. *A DiffServ policy schema.*

Figure 6 shows an object model to represent the technology-level (low-level) policies for IPSec. An instance of *security policy rule* maps an instance of *communication tunnel* to an instance of *security class*. Using the "if condition then action" representation of the policy rule, the security policy rule uses the associated communication tunnel as the condition part, and the associated security class as the action part.

Each instance of a security class is associated with one instance of *phase one parameters*, and one or more instances of *phase one transform*. Similarly, it is associated with one instance of phase two parameters and one or more instances of *phase two transform*. Each instance of a phase one transform defines a set of acceptable encryption/authentication algorithms that can be used for phase one communication within IPSec. Each instance of phase one parameters contains values of various parameters such as the duration after which keys for phase one communication must be renegotiated. An analogous explanation holds for the phase two counterpart of the transforms and parameters. Depending on the IPSec phase [3] of communication with a remote party, the associated instances of transforms and parameters dictate the operation of the IPSec protocol engine.

Each security policy rule is associated with only one security class, and each communication tunnel is associated with only one security policy rule. However, a security class may

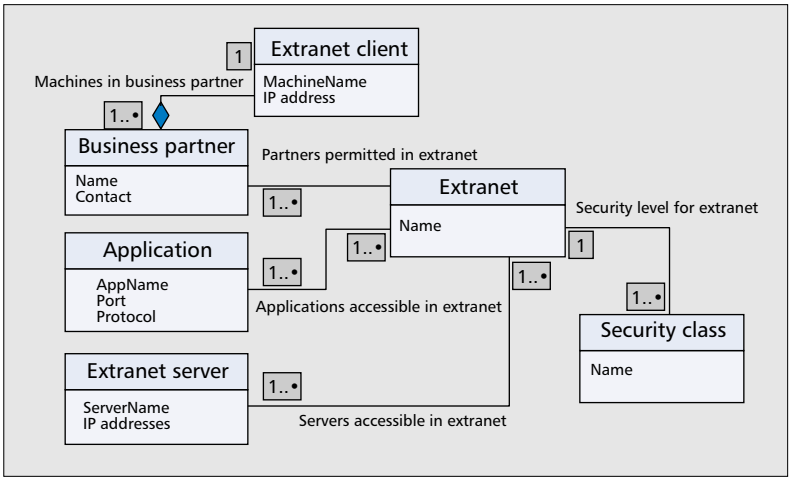

Figure 5. *A UML diagram of the business-level policy schema for enterprise extranets.*

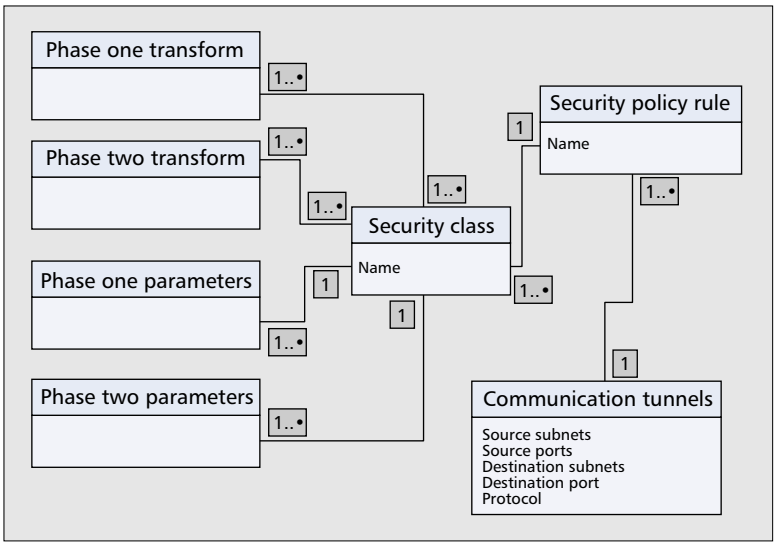

Figure 6. *An object model representing the technology-level policies for IPSec.*

be associated with more than one security policy. Similarly, instances of phase one transforms, phase one parameters, phase two transforms, and phase two parameters can be shared across multiple instances of security classes. Please note that the IPSec object model shown in Fig. 6 is a simple model intended for illustrative use within this article, and can be mapped to the standard representation used by the IETF [11], but does not follow the standard definitions verbatim.

In order to translate the high-level policies shown in Fig. 5 to the low-level policies shown in Fig. 6, we need to map the definitions of the extranets to a set of secure communication tunnels, and then generate the right associations between the communication tunnels, and the phase one and phase two parameters and transforms.

As in the case of the enterprise SLA, we presume that an expert user (e.g., the chief security officer of an enterprise) would determine an appropriate definition for a security class. As an example, a security class named *secure* might be defined as using the IPSec Authentication header protocol without encryption of packets, while a security class named *ultrasecure* might be defined as using IPSec Encapsulating Security Payload protocol with both authentication and encryption. These definitions have to be based on an object model as well. This description will essentially map a security class to a set of policies used for the low-level policy definition.

In order to translate the definition of extranets into secure communication tunnels, the policy translation tool creates one secure tunnel between each participating extranet client application machine and the named participating extranet server application machine. It then determines the appropriate mode in which the security transformation must take place. If the point where IPSec transformations happen is the same machine as the endpoints of the communication, one can use the transport mode. Otherwise, tunnel mode needs to be used.

Once the set of secure communication tunnels to be established has been determined, we can proceed with determining the relevant set of tunnels for each firewall/machine involved in the extranet. From the set of relevant tunnels at each device, one could determine the right set of phase one and phase two tunnel descriptions to be used for IPSec policies that will then be populated into the policy repository. The different firewalls involved in the process can then reconfigure themselves.

References

[1] The IETF Policy Framework Working Group: Charter available at http://www.ietf.org/html.charters/policy-charter.html
[2] R. S. Cahn, *Wide Area Network Design*, Morgan Kaufmann, 1998.
[3] D. Maughan *et al.*, "Internet Security Association and Key Management Protocol (ISAKMP)," Internet RFC 2408, Nov. 1998.
[4] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," Internet RFC 2246, Jan. 1999.
[5] J. Fritz Barnes and R. Pandey. "CacheL: Language Support for Customizable Caching Policies," *Proc. 4th Int'l. Web Caching Wksp.*, San Diego, CA, Mar. 1999.
[6] J. Hoagland, "Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on Objects." Ph.D. dissertation, UC Davis, Mar. 2000.
[7] R. Darimont *et al.*, "GRAIL/KAOS: An Environment for Goal Driven Requirements Engineering," *Proc. 20th Int'l. Conf. Soft. Eng.*, Kyoto, Japan, Apr. 1998, pp. 58–62.
[8] N. Damianou *et al.*, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems," Imperial College, UK, res. rep. DoC 2001, Jan. 2000.
[9] F. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer Verlag, 1998.
[10] D. Verma, *Policy Enabled Networking*, New Riders Publications, 2000.
[11] B. Moore *et al.*, "Policy Core Information Model — Version 1 Specification," RFC 3060, Feb. 2001.

Biographies

DINESH C. VERMA [M '92] (dverma@us.ibm.com) received a B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, in 1987, and a Ph.D. degree in computer science from the University of California, Berkeley in 1992. Since then, he has worked at the IBM Watson Research Center and Philips Research Laboratories. He is currently a research manager at the IBM Watson Research Center, and oversees research in the area of edge networking. His current research interests include content distribution networks, policy-based networking, and performance management in networked systems.