IBM PRESS

Policy Technologies for Self-Managing Systems

Dakshi Agrawal, Seraphin Calo,
Kang-Won Lee, Jorge Lobo, Dinesh Verma

Foreword by Alan Ganek, Chief Technology Officer of
Tivoli Software and Vice President of Autonomic Computing, IBM

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

IBM Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

> U. S. Corporate and Government Sales
> 1-800-382-3419
> corpsales@pearsontechgroup.com.

For sales outside the U. S., please contact:

> International Sales
> international@pearsoned.com.

Foreword

It is a great pleasure to write the foreword to this IBM Press book on *Policy Technologies for Self-Managing Systems*.

Self-management of IT systems and processes is a stated goal of the IBM Autonomic Computing initiative, which I have led for the past six years. Simply put, autonomic systems should shift more of the burden of operational decision making from people to the technology, allowing people more time for truly creative activities. In this model, IT systems sense and respond to changing conditions resulting from component failures, workload changes, environmental changes, and so on. Obviously, such characteristics offer the ability to reduce labor costs, improve reliability, utilize equipment more effectively, and many other benefits. However, implementing such a model raises the following question: How does the technology know what we want it to do? This challenge, a fundamental issue in the grand challenge of autonomic computing, is exactly what policy technologies are all about: Policies that allow humans to specify their objectives for self-management—bridging the gap between people and technology—are at the heart of achieving autonomic computing. In its Autonomic Computing initiative, IBM has developed many advances in policy technology that will assist in self-management of systems. This book by the IBM research team provides a good survey of the state of the art of policy technology and discusses the different ways in which they can be exploited to build self-managing systems.

From an adoption perspective, policy technologies in systems management are at a relatively early stage. As a result, many IT administrators are not aware of the benefits that can be attained using policy-based management and the different ways to apply policies within their environment. This book does an excellent job of providing an overview of the technology area and showing how they can be applied for tasks such as configuration management, fault management, and access controls.

The authors are among the best and most successful group of policy researchers who have contributed to key standards, developed theoretical frameworks, and, more importantly, applied policy technology to build pragmatic self-management solutions in networking, storage systems, and security systems.

This is an important topic described by experts in the field, and I am excited that this book is coming to the marketplace through IBM Press.

Alan Ganek

Chief Technology Officer, Tivoli Software, and Vice President, Autonomic Computing

Preface

Enterprise computer systems have become increasingly complex over the course of the last couple of decades, and a significant percentage of the Information Technology budget of each enterprise is spent on the management and operations of these computer systems. As technology evolves and changes, an increased level of expertise is required in the management and support of the systems. Not only the individual configurations of the computer systems vary widely across organizations, different applications, devices, and users in the enterprise also interact with each other in a complex and subtle manner. These interactions are highly unique to each enterprise environment, and it takes a significant effort to customize management systems for each enterprise environment.

In an ideal world, each enterprise computer system would have a management software that would analyze the different interactions, and require the intervention of a human administrator only on an occasional basis. Development of such a system is the goal of research into autonomic computing underway at different research laboratories and universities. This book describes how to build such a system based on policy technology, where a policy is a mechanism to put constraints on the behavior of enterprise systems.

The primary thesis of this book is to show how self-management can be enabled using policies. The book does this by presenting a general framework for self-management using policies, which is followed by examples of the application of this framework to the areas of storage systems, computer networks based on Internet Protocol, and security.

Who Will Benefit from This Book?

This book is intended for operators and architects of enterprise computing systems who want to understand how policy technology can be used to simplify the operations and management of their networks.

If you are a management software developer working on a policy enabled product, this book will help you understand the different algorithms and techniques which can be used to efficiently implement the different components of a policy-based solution. Algorithms which are useful for policy management software as well as algorithms needed at routers and servers implementing policy support are included in this book.

This book will also be beneficial to management consultants whose clients include operators of enterprise IT systems. If you are a management consultant and would like to understand the technical issues associated with the use of policies in the enterprise, this is the right book for you.

If you are a technical professional involved in areas of network or systems management in an enterprise, and want to understand how network policies can simplify your task, you will find this book to be very useful.

Who Is This Book Not For?

If you are looking for a book describing government or legal policies related to enterprise networks, this book is not for you. This book describes the technical applications of policies regarding configuration and operation of enterprise computer systems, and does not address any legal or business issues.

If you are looking for an introduction to systems or network management, this book is not intended for you. This book only addresses those aspects of systems management which relate to policies.

Finally, if you are looking for specific details on how to deploy policies in an enterprise using a specific vendor product, this book is not for you. The intention of the book is to describe the general techniques in this field and it does not describe any specific product.

The Organization of This Book

This book has been organized into nine chapters, which can be logically grouped into two parts. Chapters 1 through 5 provide an overview of policy technologies, describing the generic algorithms and techniques that can be used for defining, managing, and using policies. Chapters 6 through 8 demonstrate how policy-based technologies can be used in specific areas of systems management.

Chapter 1 introduces the concept of policies, and provides a formal definition of the same. It presents an architecture for building policy-based systems, and discusses the use of policies to achieve the goal of self-management in computer systems—such as the use of policies for developing systems that are self-configuring, self-optimizing, self-healing, and self-protecting.

Chapter 2 provides a review of the lifecycle through which policies are used to create a policy-based self-managing system, discussing the approaches that can be used to define policies and distribute them among the different components of a self-managing system.

Chapter 3 describes the information model associated with policies—that is, the logical structure of a policy and what it contains. It explores different alternate ways to represent the information model and provides an overview of the Common Information Model representation of policies, a standard specification from the Distributed Management Task Force.

Chapter 4 describes how the information models described in Chapter 3 can be represented in a concrete format using a policy definition language. A policy language is used to define policies in a manner that can be exchanged and communicated among different components of a computer system. The chapter looks at the details of various policy languages used in different policy management systems.

Although policies allow the development of self-managing systems, one needs to validate that the policies used for self-management are not inconsistent or inappropriate. A self-managing system with inconsistent policies can be disastrous. Chapter 5 describes how the different policies specified by an administrator can be validated and checked for inconsistencies, mistakes, or impossible targets. It also describes how policies entered by a human administrator can be transformed into policies that are precise and enforceable by a computer element.

Chapter 6, 7, and 8 discuss how to apply policy-based technologies to the different areas of computer systems management, and to develop systems that can automatically react to different issues arising in those areas of management.

Chapter 6 describes how the policies can be used to simplify the task of systems configuration management. In addition to discussing the different applications of policies into simplifying systems configuration and determining policy-based inconsistencies in system configuration, it provides a detailed example of using policies to validate the configuration of a storage area network.

Chapter 7 discusses the applications of policy technologies to the task of computer systems fault management. It describes how policies can be used to build a system that can automatically react to the faults and error conditions that can arise in a computing environment.

Chapter 8 describes the different applications of policies in managing security of computer systems. It discusses architectures for policy-based self-protecting systems, and provides an example of using policies to support business-level secure communications requirements using network communications technology such as the IP security protocol.

Chapter 9 discusses some advanced topics that are related to the concept of policies including a discussion of production rules, service level agreements, IP processes, and business process management.

Policy Definition and Usage Scenarios

Managing IT (information technology) infrastructure is hard. From Fortune 500 enterprises to small businesses and from nationwide data centers to personal computers in homes, an inordinate amount of time and effort is spent in managing IT. Management and operational expenses are taking an increasingly larger share of the IT budget in many organizations, with a major part of it attributed to the complexity of the systems that need to be managed.

IT management is a labor-intensive task, and skilled administrators need to intervene frequently to keep the IT infrastructure running. The exponential increase in the size of IT infrastructures coupled with increasing technical complexity has led to a situation where, despite automation, remote management, and off-shoring, the fundamental problem—there are not enough skilled people to ensure seamless operation of IT systems—remains untamed. This has driven research and industry to look for management frameworks that go beyond the direct human manipulation of network devices and systems [AUTO]. One approach toward this aim is to build policy-based management systems (PBMS). Policy-based management refers to a software paradigm developed around the concept of building autonomous systems or systems that manage themselves with minimum input from human administrators. This paradigm provides system administrators and decision makers with interfaces that let them set general guiding principles and policies to govern the behavior and interactions of the managed systems. Although large portions of the IT

management chores are still carried out manually and in an *ad hoc* manner, policy-based management systems are maturing and can be found in areas such as data center management, privacy, security and access management, and the management of quality of service and service level agreements in networks. The main objective of this book is to provide the reader with a firm understanding of what policy-based management systems are, how they can be used to reduce the cost of IT administration, and the state of the art in policy-based management in real life.

1.1. Formal Definition of Policy

The word "policy" has its origins in government and regulations and its source is Middle English and Middle French. If we open a dictionary and look for the word "policy" we may find the following definitions [MERR]:

1. A definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions.

2. A high-level overall plan embracing the general goals and acceptable procedures especially of a governmental body.

The reader may notice that even though both the definitions convey a very similar idea, that is, a policy is a plan or course of action, the distinction between the two comes from the specificity of the plan. In the first case the plan is definite and concrete, whereas the second definition refers to a high-level plan. In many occasions, the term "policy" is used interchangeably with "regulation" although regulations have more emphasis on enforcement, usually describing authoritative rules dealing with details or procedures [MERR]. In general, the word "policy" is used in a broad spectrum of situations in common English.

The use of the word "policy" in computer science, networking, and information technology has experienced a similar phenomenon. It has been used to describe among other things: regulations, general goals for systems management, or prescriptive plans of action. A few examples of where the term has been applied are access control policies, load balancing policies, security policies, back-up policies, firewall policies, and so on. We also find references to policy in high-level programming languages and systems generally referred to as Business Rules Systems [CJDA].

In many cases policies are equated with system configuration. Take, for example, policies in Microsoft® 2000 Exchange servers. In Exchange, a policy is "*a collection of configuration settings that are applied to one or more Exchange configuration objects…. You can define a policy that controls the configuration of some or all settings across a server or other objects in an Exchange organization….*"

Given the variety of usage of the word "policy," we first need to precisely define what we mean by policy. Heuristically, a policy is a set of considerations designed to guide decisions on courses of actions. Policies usually start as natural language statements. From these descriptions, many details need to be sorted out before policies can be implemented. Consider the following statement usually implemented as a default policy in Apache Web servers:

```
Do not allow the execution of CGI scripts.
```

The policy is activated by setting the value of an appropriate variable in a configuration file. During initialization the Web server reads the configuration file and adjusts its behavior in a way that when interpreting and serving documents to Web clients it will throw an exception if it encounters a CGI script as the source of the document to be rendered into the Web client.

Compare this policy to the following example from banking regulations:

```
A currency transaction report (CTR) must be filed with the federal
government for any deposit of $10,000 or more into a bank account.
```

This statement, extracted from the Money Laundering Suppression Act enacted by the U.S. Congress in 1994, is a typical policy regulation that banks must implement. In modern bank systems, the implementation will probably be done using database triggers.

The implementation of these two policies has little in common. However, there is significant commonality in the *specification*. First, both policies identify a target system: the computer where the Web server is running and the bank information system. Second, both policies express constraints over the behavior of the target system.

From the point of view of high-level policy specification, what the system is or how the system is implemented is not relevant.[1] The policy merely indicates how to regulate the behavior of a system by indicating the states that the system can or cannot take. In the Web server example, if we can take a snapshot of the state of the server at any moment in time, the policy indicates that we

should never find a process associated with a CGI script that was started by the Web server. In the banking example, if we take a snapshot of the system and find a transaction containing a transfer of $10,000 or more, the snapshot must also contain the generation of a CTR. Accordingly, for us to specify a policy we need first to identify three things:

1. The target of the policy, which we will call the **target system**. A target system may be a single device such as a notebook computer or workstation, or it can be a complex system such as a data center or a bank information system consisting of multiple servers and storage systems.

2. A set of **attributes** associated with the target system. The value of an attribute can be a simple number or text string, or it can be as complex as a structured object containing other attributes. At this moment we do not need to define a data model for attributes; we need to know only that these attributes are identifiable and accessible and that they take values from a predefined set of types.

3. The **states** that the target system can take at any given time, which are defined by an assignment of values to the system attributes.

In practice, there are many alternatives for the definition and identification of target systems. For example, the computer system where the Web server is running could be identified by an IP address; or we can group subsystems and identify the group with a unique logical name, for example, all the computers on the second floor of an office building. There are also many ways to define and get the values of system attributes. For example, an attribute of a computer system could be a set of objects representing the processes running in the computer system at a given time. These objects could be complex objects with testable properties that identify whether the object represents a process that has been started by the Web server, and whether it is a CGI script.[2] However, the behavior of a system is not completely characterized by the set of states it is in. A definition of "behavior" needs to take into consideration how the system moves through these states. Given that policies constrain the behavior, it is not surprising to find policies that constrain these state transitions. Consider the following example:

```
If a credit card authorized for a single person has been used within 1
hour in different cities that are at least 500 miles apart, reject the
charge and suspend the credit card immediately.
```

This policy is also a constraint. But in contrast to our previous policies, the constraint is not imposed on a single state of the system, but on at least three states: the state of the system at the time the credit card is first used; the state at the time when a second use of the credit card is detected and the transaction needs to be rejected; and any state in the future where credit card transactions must be rejected.

Thus, we will define the **behavior** of a system to be a continuous ordered set of states, where the order is imposed by time. Consider a system *S* that may behave in many ways. Let *B(S)* be the set of all possible behaviors the system *S* can exhibit (that is, any possible continuous ordered set of states).

Definition: A policy is a set of constraints on the possible behaviors *B(S)* of a target system *S*; that is, it defines a subset of *B(S)* of acceptable behaviors for *S*.

We note that this is a very generic definition, and it does not say how policies can be implemented or enforced. Implementations will require systems to provide operations that can affect their behavior. If there is no way to affect the behavior of the system, we will not be able to implement policies. These operations are special attributes of the system that policies can use. We will generically refer to these operations as **actions**. Note also that even though the system states can change continuously, implementations will be able to observe only discrete changes.

In many real-life systems, the state of the system may not be completely defined or known. Note that the determination of the full state of a system is not necessary to use a policy based approach. Policies can be defined using only a small number of attributes of system state and do not require the determination of the complete state *a priori*.

Let us return to our Web server example. We have noticed that activating the policy to restrict the execution of all CGI scripts is straightforward—we set the appropriate variable in the configuration file of the Apache server, the server will be restarted and it will take care of the rest by itself. Now let's take a more interesting policy. We can create policies that will allow different sets of users to execute different sets of CGI scripts. The implementation of a policy like this in a standard Apache server is not that obvious. One could try to implement this policy by creating a directory structure that reflects the different sets of scripts with links to the scripts from the appropriate directories, and creating access control files for each directory with the different sets of users that have

access to the scripts. Thus the policy would be enforced by giving user names and passwords to the users and forcing users to authenticate themselves before executing any of the scripts. A severe inconvenience with this implementation is that changes in either the set of users or scripts may require reshuffling of the directories and changes in different access control files. The difficulty arises because there is no obvious connection between *what* the policy wants to enforce and *how* it is enforced. In the simple CGI policy, *how* the policy is implemented is hidden inside the implementation of the Web server and the implementer needs merely to set the policy on or off. For the second case, having only the possibility of setting the CGI script execution policy on or off is too restrictive because an essential component of *what* the policy wants to constrain is conveyed by the different sets of users and scripts.

A policy-based management system aims to provide an environment to policy authors and implementers where they can concentrate their efforts on describing *what* the policy restricts and thereby alleviating the burden created by having to describe *how* the policy will be enforced. This separation of *what* from *how* varies widely among different systems and applications, and in practice most policy authors are still required to have at least partial understanding of policy implementation.

1.1.1. Types, Nature, and Usage of Policies

As defined earlier, policies are constraints on the behavior of a system, and system behavior is a sequence of system states. In turn, each state of a system can be characterized by the values that a collection of system attributes takes. In this section, we enumerate some of the common types of constraints specified on the system behavior, and discuss how they result in different types of policies.

The attributes of a state can be divided into three groups—a set of fixed attributes, a set of directly modifiable attributes, and other observable but not directly modifiable attributes. The fixed attributes of a system cannot be modified directly or indirectly. As an example, a server in a data center has a state characterized by attributes such as maximum number of processes, size of virtual memory, size of physical memory, amount of buffer space for network communication, processor utilization, disk space utilization, memory utilization, time taken to respond to a user command, and so on. Among these, the size of physical memory is a fixed attribute for the purposes of systems management—it

cannot be changed until the hardware of the server itself is modified. Some of these attributes, such as the maximum number of processes, size of virtual memory, or the amount of buffer space, can be modified directly by changing some values in a configuration file. Other attributes, such as processor or memory utilization, cannot be modified directly. They can be manipulated only by modifying the direct parameters or taking some other action—for example, by killing a running process. We define the set of directly modifiable attributes of a system as its *configuration attributes*. Furthermore, the set of attributes that are not directly modifiable, but can be observed or computed from observation of system attributes, are defined as *system metrics*. The configuration of a system, using these conventions, is the collection of the configuration attributes and the assignment of values to them.

The simplest policy type specifies an explicit constraint on the attributes of the state that the system can take, thereby limiting system behavior:

Configuration Constraint Policy: This type of policy specifies constraints that must be satisfied by the configuration of the system in all possible states. These may include allowable values for an individual configuration attribute, minimum and maximum bounds on the value of an individual configuration attribute, relationships that must be satisfied among different configuration attributes, or allowable values for a function defined over the configuration attributes. Some examples of configuration constraint policies are as follows:

- ```
  Do not set the maximum threads attribute on an application server
  over 50.
  ```
- ```
  The size of virtual memory in the system should be less than two
  times the size of physical memory.
  ```
- ```
  Only users in the administration group have access to the system
  configuration files.
  ```

Configuration constraint policies are often used to ensure correct configuration of a system, to self-protect the system from operator errors, and to prevent the system from entering the operational modes that are known to be harmful.

Metric Constraint Policy: This type of policy specifies constraints that must be satisfied by the system metrics at all times. Unlike configuration attributes, the metrics of a system cannot be manipulated directly. The system needs to determine in an automated manner how to manipulate the configuration of

the system, or to take appropriate actions such that the constraints on the metrics are satisfied. The constraints on the metrics may include bounds on any observable metrics, or relationships that may be satisfied among a set of system attributes including at least one metric attribute. Metric constraint policies that specify an upper or lower bound on a metric are also known as *goal policies* because they provide a goal for that metric, which the system should strive to achieve. Examples of metric constraint policies include the following:

- ```
  Keep the CPU utilization of the system below 50%.
  ```
- ```
  All directory lookups on the name of a person should be completed
  in less than a second.
  ```
- ```
  The end-to-end network latency should be kept below 100
  milliseconds.
  ```

Metric constraint policies are often used to enable self-configuration of systems in order to meet specific performance requirements or objectives.

Action Policy: In the preceding examples, the two types of policies described specify constraints on a single system state. In many cases, a policy may require explicit actions to be taken when the state of a target system satisfies some constraints. These types of policies are called *action policies* because they require the system to take a specific set of actions. Action policies constrain a sequence of states. That is, when a particular state is observed then certain actions must be taken at a later point so that the target system will be in some other state. In most cases, the action policy would modify the configuration of the system in response to some condition being true. Action policies essentially provide a plan according to which the system should operate when it encounters a certain condition specified in the policy. Examples of action policies include the following:

- ```
  If the CPU utilization of a server in a data center exceeds 70%,
  allocate a new server to balance the workload.
  ```
- ```
  If the temperature of the system exceeds 95 degrees Celsius, then
  shut-down the system.
  ```
- ```
  If the number of bytes used by a hosted site exceeds 1 Gbyte in a
  month, then shut down access to the site.
  ```
- ```
  If the inbound packet has a code-point for expedited forwarding
  (EF) per-hop behavior (PHB) in the packet header, then put it in
  the high priority queue.
  ```

In these examples, action policies have been used to manage the performance of computer servers and networks, for managing the effect of environmental conditions, for limiting resource utilization, and for providing different Qualities of Service in communications networks.

Not all action policies specify an action that can be directly executed on a system. One important type of action policy is the alert policy, which is commonly used to flag any conditions that may require operator intervention.

Alert Policy: An alert policy is an action policy where the action consists of a notification sent out to another entity. A notification is an action that does not modify the configuration of the system itself. Instead it can take one or more of the following forms: sending an email or an SMS message, making a phone call, logging a message in a file, or displaying an alert visually on a display. Some examples of alert policies are as follows:

- Notify all users who have not accessed their account for three months by email to warn of possible account deletion.
- If a system has not installed the latest version of anti-virus software, send an email to the employee and his/her manager.
- If a system has gone down, send a message to the administrator's pager.

Although real-life policies, as shown here, are specified in various different styles, all of these policies can be restructured using a common pattern or a model. Formally, this model is called the *policy information model*. One of the most widely used policy information models describes a policy using a *condition-action* rule, which means if the condition is true then perform the action. A more specific version of the condition-action rule is the *event-condition-action* (ECA) rule, which means upon occurrence of the event, if the condition is true then perform the action. It is not difficult to see that the preceding policy rules can be transformed into some version of ECA rules. For example, the metric constraint "The end-to-end network latency should be kept below 100 milliseconds" can be rewritten as "Upon completion of measurement, if the end-to-end network latency is above 100 milliseconds, then record the violation in the system log file." The policy information model is a useful framework to describe, compare, and analyze various different policy rules. In Chapter 3, "Policy Information Model," we will review some of the popular policy information models that are being widely used.

Having defined policies as constraints on the operation of a system, let us examine how the specification of such constraints can help in the management of IT systems. The specification of constraints on the state of the system can be used for several purposes, such as

- When the demand or workload on a system changes requiring a recon-figuration of the system, the constraints can be used to determine a desirable new configuration.

- When there is a contention for resources in the system, the constraints can be used to determine the manner in which to resolve that contention.

- When any external entity tries to access the resources in the system, the constraints can be used to determine whether that access ought to be permitted.

- When a system violates certain constraints, it can determine and exe-cute a set of actions that will allow it to remove that violation.

Policies can be used to build systems that are autonomic—that is, exhibit the properties of self-configuration, self-protection, self-optimization, and self-healing. A self-configuring system would configure itself according to its intended function. A self-protecting system would identify threats to itself and take corrective actions. A self-optimizing system would modify its configura-tion according to the current workload to maximize its performance. A self-healing system would automatically repair any damage done to its components. The manner in which policy technology can be used to enable the development of such systems is described in the next few subsections.

1.2. Policy-Based Self-Configuration

One of the most time-consuming operations in the management of any system is the initial configuration of a new installation, or the reconfiguration that needs to be performed when new requirements are received. The basic approach in self-configuration is to offer a simplified set of abstractions that the administrator needs to manipulate, while the detailed configuration of a myriad of parameters in the system are hidden to a large extent. Although there are several instances in which policy-based self-configuration mecha-nisms can be used, we will use the example of a hosting service provider for illustration purposes.

For the sake of simplicity of illustration, let us assume that all the customers of this service provider run their Web sites only within the premises of the service provider, and each Web site is supported by one or more instances of a Web

server such as Apache or IIS. The service provider has a pool of stand-by servers that can be deployed for any customers after a proper installation of applications. Some of the customers whose Web sites draw heavy traffic may need multiple servers at the site with a load-balancer in front of them, whereas customers whose sites are not as popular may be sharing a single server with other customers. The service provider can set up a hosting Web site with a set of routers, virtual LAN switches, load-balancers, and server blades that can enable this service. All of these devices make up the *target system* of the policies. Because most hosting service providers will have system administrators who can write scripts to automate common processes, we further assume that they will have developed a series of scripts so that they can automatically allocate a server from a shared pool to a specific customer, and conversely return a server back to the pool once the busy period is over or the contract with a corresponding customer expires. A similar script can be developed for automating the addition or removal of a new virtual server for smaller customers. A more detailed discussion about how such a system can be developed can be found in [APPL].

When a new customer is added or an existing customer removed, the configuration of the site needs to be changed according to the change in the set of customers being supported. If there are mechanisms available for servers to be assigned in an automated manner to different customers from a shared pool, then the number of servers or processors assigned to a specific customer may change depending on the intensity of traffic to that site. Sometimes the hosting site may want to enforce limits on how much bandwidth a customer's site can use in a month, and may want to reconfigure the site to restrict the throughput available to a hosted site if the traffic to that site exceeds predetermined thresholds. Let us assume that the service provider characterizes its customers into two groups: large and small. It may instantiate policies for self-configuration of its site, which may look like the following:

```
If a large customer has 75% or more utilization of all its servers, and
has less than its maximum allowed number of servers, then allocate an
additional server from the free pool to it.
```

```
If a large customer has 30% utilization or less on all its servers, and
has more than its minimum allowed number of servers, then remove a
server from it to the free pool.
```

```
If a small customer has reached 125% of the monthly bandwidth allowed
to its site, then disallow further access to the site for the rest of
the month.
```

```
If the addition of a new small customer causes the number of small
customers at a server to exceed a threshold, allocate a new server from
the free pool and migrate half of the existing small customers to that
new server.
```

In the preceding example, we can identify several attributes: utilization rate, number of servers, numbers of free servers, monthly bandwidth, and so on. Each of the policies provides a constraint on the new configuration of the system. The behavior of the system (allocation of servers between customers and the free pool) is constrained to conform to the guidelines set earlier. The guidelines may change based on the experience of the service provider—instead of using server utilization, it may opt to use the bandwidth consumed as a trigger for reallocation of servers, or it may use a combination of both. Also it may choose to not block small customers that exceed their throughput limits, opting instead to charge them an additional amount of money. Looking back at the discussion of policy types, we can recognize all of these policies as instances of action policies.

These sets of policies allow the administrator to manage the customers using attributes (utilization, number of servers, bandwidth rate, and so on) that are decoupled from the details of actual server configuration (their IP addresses, commands to control bandwidth, their operating system version, and so forth). Thus, the goal is to allow administrators to view system management in terms of the abstracted attributes that lets them specify *what* needs to be done, leaving the details of *how* it can be done to the underlying mechanisms that support a policy-based management system. Policies do not describe the mechanisms for the reallocation of the servers, the migration of customers to the new server, or disabling access to any site. However, assuming that appropriate scripts to do these tasks exist, the ability to specify the policies and invoke the right scripts for the required actions would enable the system to self-configure itself in accordance with the wishes of the service provider.

Building a policy enabled management system for this scenario would involve three steps:

1. Determining a way to specify the policies.

2. Enabling support within the system to interpret and enforce the policies.

3. Invoking a mechanism to distribute policies from the entity specifying them to the entities interpreting and enforcing them.

To specify policies, a language that can capture the semantics of policies needs to be selected and a tool to specify the policies needs to be developed. Later we will discuss that having an information model is also an important aspect of the specification process in addition to selecting a policy language. The system management software that allocates and reallocates servers needs to understand policies specified in this language so that it can enforce the policies by transferring the servers under various operation conditions. Finally, it is important that the policy specification from a system administrator is distributed to a system that can enforce the policies. In this particular case, if there is only one instance of the system management software, the third problem of distribution is trivial because there is no need for synchronizing among multiple copies of the policy.

1.3. Policy-Based Self-Protection in Computer Networks

Policy-based management in network administration has been practiced for more than a decade and has been used successfully in many application areas. A common usage of policies in network management is to regulate the traffic in the network, especially dealing with the security, access control, and quality of service of different traffic streams. Some examples of network traffic security policies include the following:

- `Allow telnet connections out of the local network.`
- `Block telnet connections into the sites on the local network.`
- `Allow only secure HTTP traffic, and block any other traffic into the local network.`
- `If a UDP packet on an illegal port is received from an external computer, disallow any communication to that computer.`

The enforcement of these policies within the network, which is the *target system* in this case, needs to be managed in the context of the configuration of a specific network. If we consider a simple model of network access protection, in which access to the local network is secured and protected by means of one or more perimeter firewalls, then the policies can be readily seen as configuration constraint policies as described in the previous section. The enforcement of these policies requires that the configuration of the firewall be done in a manner that is consistent with these policies. In this case the *attributes* are source and destination IP address and ports.

As in the previous example, we need to have means to specify the policies, interpret and enforce the policies, and distribute policies from the entity specifying them to the entities interpreting and enforcing them.

For the purpose of defining policies, a machine-readable language needs to be developed for their specification. For access control and security policies, the language could be a standard access control policy language such as the "eXtensible Access Control Markup Language" (XACML) [OASI] from the Organization for the Advancement of Structured Information Standards (OASIS) or some other policy languages with similar capabilities.

The firewalls in the system need to implement and enforce the access control policies. If they are able to interpret the language selected for specifying policies, they can take the policy as it is specified for enforcement. Otherwise, a translation of the policies to a format that the firewall can interpret needs to be done. In some cases, the policies might not be able to be translated easily into a firewall configuration. An example would be the presence of a policy requiring that a notification should be sent out to the administrator whenever an access attempt to a forbidden site is made. Because the firewall is not capable of sending notifications—it merely records passively the sites that users are trying to access—an additional mechanism is needed. In this case, an independent software package would be required to periodically collect the records of which sites were accessed by each user from the firewall, and then process them to check whether a notification needs to be generated.

If there is more than one firewall in the system, we need to have a mechanism that will keep the policies specified in different firewalls consistent. Although this can be done manually, it would be more convenient to have an automated mechanism to dispense and distribute the policies. Various alternative approaches to the distribution of policies can be developed and they are discussed in detail in Chapter 5, "Policy Transformation and Analysis."

In another variation of self-protection, policies may be defined that indicate how the set of applicable policies ought to be changed. As an example, if a system detects that the system is under attack by a newly identified infected host, the applicable policy rules may be augmented to prevent traffic from that infected host to reach the rest of the network.

There are several other uses of policies in managing computer networks. The application of policy-based fault management in computer networks is discussed in detail in Chapter 7, "Policy-Based Fault Management."

1.4. Policy-Based Self-Optimization in Computer Systems

A computer system can be called self-optimizing if it can reconfigure itself so as to best satisfy the needs and requirements imposed upon it. In the case of enterprise systems, the optimization requirements imposed on a computer system are usually driven by the needs of a business. The business that owns the computer system could have some contractual obligations that it may have signed up for. For example, the hosting services provider may have signed a service level agreement (SLA) promising a target system response time to a customer, and it would like to provision, configure, and operate its systems so that it is in the best situation to meet its obligations.

Self-optimization in computer systems may be specified by means of metric constraints policies or goal policies. These policies would require a bound on a metric that the system may or may not directly control. When the metric constraint policies are specified, the system is expected to try to do its best to meet them.

To be able to match these policies, the system must translate them into a set of action policies—that is, a set of actions that can be invoked when some conditions about the system behavior are satisfied. The translation of metric constraint policies into action policies is the process of policy transformation, which is described in Chapter 5.

An example of this type of policy is the support of service level agreements. An SLA might read as follows:

```
The service provider must be able to process at least 1000 transactions
per second under the condition that the system is operating under normal
load 70% of the time. Otherwise the provider is not obliged to fulfill
the requirement.
```

The policy must also define normal and overload conditions (but they are not shown here for simplicity). An implementation will provide a client of the service provider with access to the system attributes so that client or sometimes a third party can verify that the agreement has been fulfilled. If policies are violated, the policies themselves may include penalties or compensations as actions to encourage the system to conform its behavior to the contract. To implement self-optimizing policy, the system can try to predict when it is expected to fail the requirements and take corrective actions before any constraint is violated.

1.5. Policy-Based Self-Healing

Sometimes the primary role of policies is to make sure that the operational state of the system satisfies the policies that are defined within the system. If the system is not satisfying those constraints, it should take corrective actions or create an alert. Thus, both action and alert policies can be used to implement self-healing systems, although one may take the position that the alert policy simply allows the system to call for assistance when it sees an issue rather than healing itself.

An example of this is the following security policy:

```
The temperature in the blade center must be maintained at less than 65
degrees.
```

For example, if one of the fans in the blade center breaks, this policy can trigger an action to put some of the blades to sleep in order to reduce the temperature. If the action does not sufficiently reduce the temperature, an alert policy can be triggered to request the attention of the system administrator.

Another good example can be found in storage area network (SAN) configuration management. SAN configuration is a complex problem because of the interaction between many different devices and software systems. Configurations need to make sure proper device drivers are installed, incompatible devices are not connected or are not configured in the same zone[3], redundancy requirements are fulfilled, and so on. To cope with the complexity, experts come up with various sets of policies that represent best practices for interoperability and reliability. An example of a policy in this set would be the following:

```
The same host bus adapter (HBA) cannot be used to access both tape and
disk devices.
```

This policy can be verified automatically if there is an appropriate software module installed at different computers and devices in the storage area network. Data about the system configuration is collected and policies are evaluated against the data. If the configuration is not compliant, the policy management service reports the violations and may isolate parts of the system to avoid errors or failures. In some cases, it may even be able to modify the system automatically for compliance, thereby achieving the goal of self-healing.

As previously mentioned, a machine-readable language needs to be developed for the purpose of defining policies, and an information model is an important aspect of the specification process. The software module that validates compliance needs to be a component of the management software for the system that collects the state information and checks it for violation of policies.

1.6. Building a Policy-Based Management System

The reader must have noticed from the description of the various aspects of policy-based management for different scenarios that there are many similarities in the design of the underlying capabilities. In this section, we present the architecture of a generic policy-based management system and describe the functions needed to build it.

We start as a reference point with what is usually called the *IETF/DMTF Policy Architecture*. The IETF/DMTF Policy Architecture is found in many Request for Comments (RFCs) published regarding the use of policies in computer communication networks. (See the sidebar titled "IETF and DMTF" for more details.) Despite being cited as the IETF Policy Architecture or the DMTF Policy Architecture, it is worth pointing out that neither of the two organizations has actually standardized policy architectures. Thus, the architecture is not a standard formally defined by either the IETF or the DMTF. It is more akin to a folk architecture that is usually ascribed to the IETF and/or DMTF.

IETF and DMTF

The IETF (http://www.ietf.org) and DMTF (http://www.dmtf.org) are two standards organizations that have been at the forefront of policy standardization. IETF is an acronym for the Internet Engineering Task Force. It is the organization that defines the standards governing the protocols, management and other issues related to the operation of the Internet. IETF standards are published as RFCs that are publicly available on the Internet. Several policy-related efforts were carried out within the IETF as part of the initiatives needed to ensure Quality of Service and security within the Internet. In particular, RFC 2748 defined the COPS (Common Open Policy Service) protocol, RFC 2749 defined COPS usage for RSVP (Resource ReSerVation Protocol), and RFC 2753 defined a framework for policy-based admission control. All three RFCs mentioned were published in early 2000.

> A working group within the IETF defined a policy common infor-
> mation model, which was published as RFC 3060 in 2001. Work
> on defining policy standards was subsequently moved over to the
> DMTF (Distributed Management Task Force), which is an indus-
> trial organization that develops standards required for managing
> different types of IT systems. The DMTF's key technical contribu-
> tion has been the development of a common information model—
> that is, the definition of a standard set of information that all IT
> systems need to provide in order to be managed in a uniform
> manner. After the standardization of policy efforts moved to DMTF,
> further enhancements to the policy information model were pub-
> lished as RFC 3460 in 2003. Although RFC 3060 and 3460 take a
> big step toward standardization of policy efforts, they are still in the
> realm of information models and they fall short of concrete archi-
> tectural prescriptions.
>
> Other standards organizations have also defined standards related
> to policies, and they are discussed in more detail in Chapter 9,
> "Related Topics," of this book. However, the work of the DMTF and
> IETF on policies forms the basis from which many of those stan-
> dards have been derived.

The reason the folk architecture is cited so frequently in academic circles is that
it captures the driving principles behind the derivation and definition of many
of the standard RFCs and drafts submitted to the IETF that deal with policies.
Thus, even though this architecture is not put out in any of the official docu-
ments from the two organizations, it represents the guiding principles behind
much of the standards work, and it is appropriate to refer to it as the
IETF/DMTF architecture.

This policy architecture consists of four components as shown in Figure 1.1: a
policy management tool, a policy repository, a policy decision point (PDP), and
a policy enforcement point (PEP). The policy management tool provides a user
interface for the creation and definition of policies. The policy repository pro-
vides mechanisms for storing policies and retrieving them as needed by the
decision points. The policy decision points are modules in the system respon-
sible for making policy decisions—that is, they examine the policies stored in
the repository that would be applicable under a given circumstance and deter-
mine what needs to be done to comply with those policies. Policy enforcement
points are elements that are responsible for enforcing the outcome of those
policy decisions.
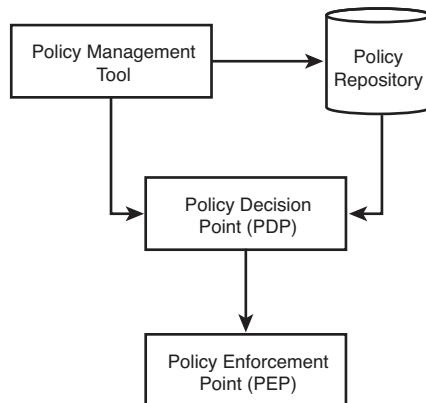
Figure 1.1   The IETF/DMTF Policy Architecture

Although the formal definition of policies, their types and usage, and the bare-bones IETF/DMTF policy architecture just described captures basic concepts used in all policy-based management systems for self-management, they hardly convey the complexity faced in implementing a PBMS. When building a PBMS, an architect needs to make several decisions regarding policy lifecycle: when and where policies are created, modified, and transformed from higher levels of abstraction to lower levels of abstraction; how they are stored, distributed, and enforced in a possibly geographically dispersed system; and finally, when a policy becomes obsolete, how it is retracted throughout a system without disrupting the system operation. Another aspect of this complexity comes from possibly conflicting policies in a system that need to be resolved. Conflicts arise often in a PBMS because there are often multiple policy authors responsible for managing different operating behaviors (self-healing policy versus self-protection policy) of the system. Conflicts can arise during definition time or at run-time and they need to be resolved to make a decision.

In the following chapters, we will cover some of these implementation issues of a PBMS in more detail.

1.7. Summary

The term "policy" can mean many different things to people depending on the context, background, and application domain. This chapter provided a formal definition of "policy" in the system management context, and discussed different types of practical policies and their usages. In particular, it singled out configuration constraint policy, metric constraint policy, action policy, and alert policy. It then briefly introduced that they can be represented in a policy information model in the form of condition-action rules or event-condition-action rules. We will review the policy information model more in detail in Chapter 3. The chapter also presented high-level scenarios for using policies for self-management of a system. The characteristics of self management have been studied in the context of self-configuration, self-optimization, self-healing, and self-protection. Finally, the chapter described a generic architecture for policy-based management systems as defined by IETF and DMTF. More specific examples of policy-based management systems will be presented in later chapters.

Endnotes

[1] The system details will be relevant during policy implementation.

[2] For instance, if the computer is a machine running a Unix-like operating system, all this information can be easily obtained by executing the "ps" command.

[3] In storage area networks, a zone is a logical grouping of ports that belong to servers, switches, and storages in which only the ports in the same zone can exchange data.

Index