A Policy Based Network Configuration Framework

Zhifeng Xiao

School of Computer Science

McGill University, Montreal

June, 2002

A Thesis Submitted to the Faculty of Graduate Studies and Research

in Partial Fulfillment of the Requirements of

the Degree of Master of Science

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisisitons et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-85837-5
Our file Notre référence
ISBN: 0-612-85837-5

ABSTRACT

Networks and their services have been growing rapidly in recent years, so has the complexity of configuring the network operations. In order to manage the rapidly expanding networks, new management protocols are thus needed to change the network management from configuration of individual devices to automation of the whole network. SNMPCONF, a policy-based configuration with SNMP protocol, is proposed by IETF to meet this requirement.

SNMPCONF is suggested to configure networks with a policy MIB (Management Information Base). However, deploying this protocol in existing network devices needs more work, because the policy MIB defined by SNMPCONF is not implemented now in the present network and no configuration MIB defined. Even if the configuration MIB and policy MIB implemented by vendors in future, some existing devices may not be upgraded to support these MIB for their hardware limitation (e.g., memory limit). Another problem is found in the communication between a SNMP supported network and the SNMP unsupported networks (e.g. telecommunication network where TL1 is used in North America). In this study, CLI and TL1 are used to help solving these problems. Based on Modular SNMP from University of Quebec at Montreal, a JAVA framework to deploy SNMPCONF with this approach is implemented and tested on a network built with Cisco routers (Cisco IOS 12.0, routers ranged from Catalyst 2600 to 2900). The preliminary work shows the SNMPCONF can be implemented by wrapping CLI commands as accessory functions in policy language without configuration MIB. As shown in this work, another advantage of this approach is that policy execution is atomic and persistent.

RESUME

De nos jours, les réseaux et leurs services ainsi que la complexité de ses opérations sont en train d'agrandir rapidement. Afin de gérer les réseaux agrandissants, des nouveaux protocoles de gestion seront nécessaries pour changer le gestion du réseau à une automation du réseau entier au lieu d'une configuration des dispositifs particuliers. SNMPCONF, un type de configuration à base de règles et contient le protocole SNMP, est présenté par le groupe de travail IETF pour satisfaire à ces exigences.

SNMPCONF est supposé de configurer des réseaux avec la base de données MIB (Base d'informations de gestion). Cependant, déployer ce protocole dans les unités des réseaux nécessitent plus de travail, car la base de données MIB définie par SNMPCONF n'est pas mise en oeuvre dans le présent réseau et aucune configuration de MIB n'est définie. Même si la configuration et la base de données MIB sont appliquées par des marchands à l'avenir, quelques dispositifs ne peuvent etre implémentés pour appuyer ces MIB à cause de leur limitation matérielle (e.g, limitation de mémoire). On trouve d'autre problème dans la communication entre un réseau SNMP soutenu et les réseaux SNMP non soutenus (e.g, le réseau de télécommunications où TL1 est utilisé en Amérique du Nord). Dans cette étude, CLI et TL1 sont utilisés pour résoudre ces problèmes. Basé sur le Modulaire SNMP de l'Université du Québec à Montréal, avec cette démarche, la structure de JAVA est implémentée et éprouvée (expérimenter) sur un réseau développé avec des routeurs Cisco (Cisco IOS 12.0, des routeurs qui variaient de Catalyst 2600 à 2900) pour déployer SNMPCONF. Le travail préliminaire nous explique que SNMPCONF peut être implémenté au moyen d'un retour automatique aux commandes CLI comme des richesses fonctionnelles dans la langue des règles sans la configuration

MIB. Comme on a démontré dans ce travail, un autre avantage dans cette démarche est que l'exécution de cette base de données est atomique et persistante.

Acknowledgements

This work could not be done without supervision and encouragement of Professor Omar Cherkaoui at University of Quebec at Montreal and Professor Gerald Ratzer and Professor Tim Merrett at McGill University. The author would like to express his deep gratitude to his supervisors. People in Teleinfo Lab of University of Quebec at Montreal offered much help to the author during this work. I would like to thank Ayoub Cherkaoui for discussing Modular SNMP and a special example of a SNMP agent using Modular SNMP. Thanks also to Jianlong Zuo, Tian Lin, Francois Bedard, Alan Sarrazin for their helps and discussions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network management is used to monitor, control, and plan the resources and components of network (Mellquist, 1997). A functional breakdown of requirements is best defined by the international organization for Standardization (ISO). According to ISO, there are five different kinds of management, including fault management, accounting management, configuration and naming management, performance management, and security management (Stallings, 1996):

Fault management, which involves the detection, isolation, and correction of abnormal operations.

Accounting management, which deals with issues such as billing, delivery, and logging of accounting events and modeling of accountable resources.

Configuration management, which is concerned with configuration of network and computational resources, software resources and components of distributed applications.

Performance management, which evaluates the behavior of managed objects and effectiveness of communication activities

Security management, which manages issues such as authentication, access, control, and partial trust relationships.

Among these management activities, configuration is one of the most important parts of the network management. Due to the increasing demands for telecommunication and the emerging of e-business, manually configuring network will soon become an impossible task for network administrators. Therefore, new network management applications are required to deliver fast-responding and rich-automated functionalities

and easily accommodate new services. In this chapter, basic network management concept will be introduced first and then the new policy based network management paradigm will be discussed.

1.1 Element of Network Management System

A network management system is a collection of tools for network monitoring and control. It has two integrated parts: an operator interface with a powerful but user-friendly set of commands for performing most or all network management tasks and a minimal amount of separate equipments. Most of the hardware and software required for network management is incorporated into the existing user equipment (Stallings, 1996).

Network management can be implemented by utilizing various architectures based on the network types and sizes. Two fundamental architectures, centralized management and distributed management, can be used to achieve the same purpose (Stallings, 1996). The centralized management is appropriate in the management of small to medium size networks that are not geographically distributed. In this kind system, all the devices are under the command of a central network manager. The network manager must reside at the same point as these devices it managed. In order to manage and monitor the networks actively, the manager must poll or query network device found within each network. This produces management traffic on the network connections.

In contrast to the centralized model, a distributed model distributes information and control so that each network or geographically distributed site is responsible for itself. In a distributed model, management is pushed down to its lowest level and mid-level managers are responsible for their own domain of management. Important information pertinent to the entire network is forwarded to a manager of managers. This model scales well for networks that are large in size.

In the centralized management, Manager/Agent paradigm is often employed. Figure 1.1 presents the detail architecture (manager/agent) of a network management system. In this kind of system, at least one node is designed as the network control host, or manager. Other parts of the network management system are generally referred as agents. Each network node contains a collection of software devoted to the network management task, called a Network Management Entity (NME). The network management entity is responsible to collect and store information about communications and network-related activities and respond to commands from the network control center. The communication software handles the application-level network communication between the manager and the agents.

In addition to NME, the manager includes a collection of software called the network management application. The network management application responds to user commands by displaying information and/or by issuing commands to network management entities through the network. This communication is carried out using an application-level network management protocol that employs the communications architecture in the same fashion as any other distributed application.

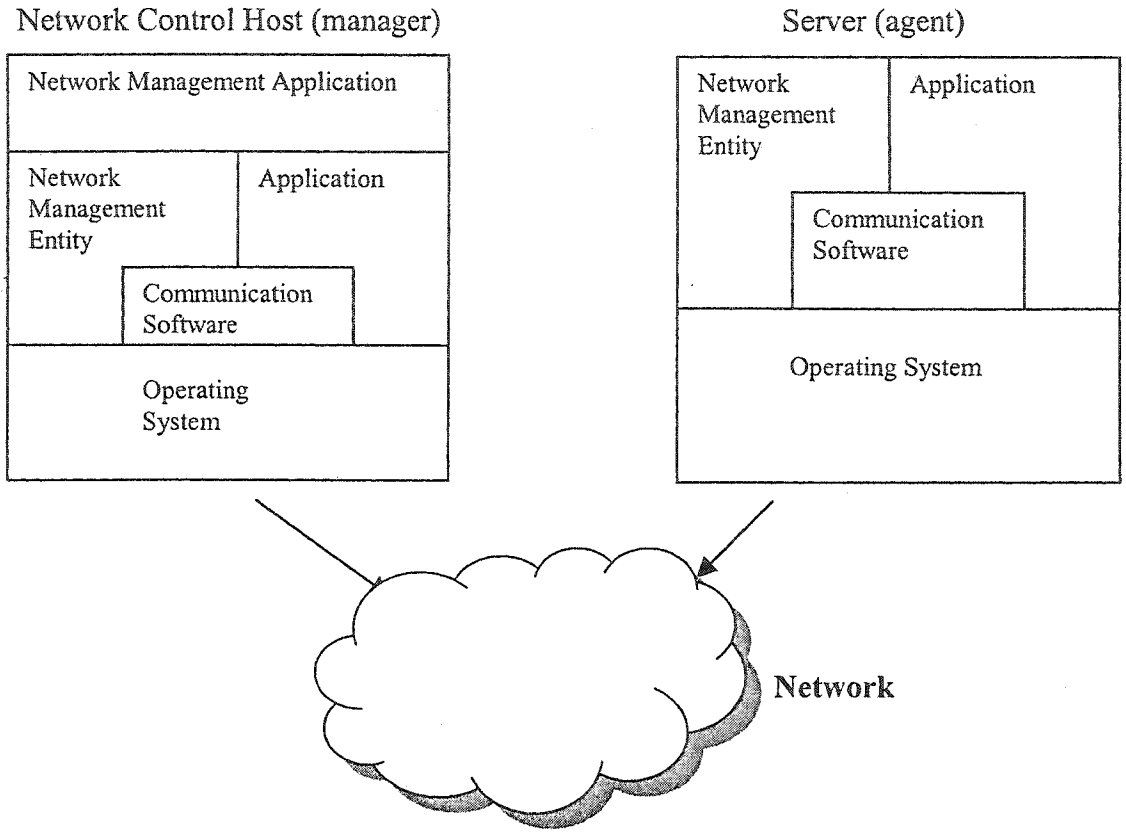


Figure 1.1 Elements of network management system

(modified from Stalling, 1996)(See text for details).

In Figure 1.1, it is assumed that all managers and agents share a common network management software. However, this assumption may not be practical or possible. For example, a network may consist of some older systems, which do not support the current network management standards but are too expensive to be replaced immediately. To handle such cases, it is common to have one of the agents in the server as a proxy for one or more other nodes. When an agent performs in a proxy role, it acts on behalf of one or more other nodes. A network manager who wants to communicate with these nodes

should contact a proxy agent directly. Then, the proxy agent translates the management protocol and uses an appropriate protocol to communicate with the target system. Responses from the target system back to the proxy are similarly translated and passed on to the manager.

The Manager/Agent paradigm is extensively deployed in the present network management and proved to be very practical. For example, the widely supported SNMP protocol, which will be discussed later, employs the manager/agent paradigm. However, as pointed out by Stevens and Weiss (1999), most of present network management systems only enable the administrator to configure individual network device remotely and cannot actually manage networks at the network level due to the limitation of the protocols or the lack of an infrastructure which wins extensive vendor supports.

1.2 Policy-based Network Management

Policy-based network management paradigm is aimed at managing the networks as whole rather than the individual devices and is proposed to fulfill the new requirements due to the increasing network services (Erfani et al., 1999). Policy-based management is not an entirely new concept. With packet-filtering capability, firewalls and switches in effect implement the policy. Configuring specific default routers and establishing static routes for special circumstance are examples of simple policy implementation (Stevens and Weiss, 1999a).

The purpose of a policy system is to manage and control a network as a whole, so that network operations conform to the business goals of the organization that operates

the network. "Ultimately, achieving such control requires altering the behavior of the individual entities that comprise the network. The policy framework represents an alternative approach to controlling the operational characteristics of an IP network. Unlike traditional network management approaches, the systems developed within the policy framework implement policy by centralizing the storage of prescribed rules instead of implementing policy by centralizing control functions into a single software application. A policy system devised under this framework shifts the focus from configuring individual devices to setting policy for the network in aggregate, and controlling device behavior through network policy " (Stevens and Weiss, 1999a).

Policy-based network management is the practice of applying management operations globally on all managed network devices that share certain attributes. "Policies always express in a notion of:

if (an object has certain characteristics) then (apply operation to that object)

Or in the following normal form:

if (policyFilter) then (policyAction)

"A policy filter is a piece of program code which results in a boolean to determine whether or not an network device (object) is a member of a special set upon which an action is to be performed. A policy action is an operation performed on an object or a set of objects " (Waldbusser et al., 2001). A policy filter and a policy action combined form a policy rule.

A policy system may be broken down into the following functional elements (Stevens et al., 1999b, see Figure 1.2):

1) A policy management tool, to enable an entity (e.g.: person, application) to define and update policy rules and optionally, monitor their deployment, for example, a graphical or command line/script interface.

2) A policy repository, for persistent storage and retrieve of policy rules.

3) A policy consumer, which is a convenient grouping of functions, is responsible for acquiring policy rules, deploying policy rules, and optionally translating policy rules into a form useable by policy targets.

4) A policy target, which is a functional element whose behavior is dictated by policy rules and carries out the actions indicated by the policy rules.

Figure 1.2 is very similar to Figure 1.1 in terms of the architecture. If we take policy specification as policy manger and the policy target as policy agent, then the policy based network management system fall into the manager/agent paradigm. The only difference might be the management information which is transported between the agents and the manager. In case of policy system, most of the management information would be the policies.

Recently, a lot of work has been done on the research of policy based network management recently. The work of Dr. Sloman and his group in Imperial College provided a framework for managing large-scale distributed systems. They used domains as a means of partitioning responsibility, which provides a flexible means of introducing boundaries of management responsibility and a framework for specifying management policy. They have developed notations and tools support for specifying policies, and analyzing them for conflicts within a role-based framework. They have also developed techniques for specifying interaction and coordination between different roles and are

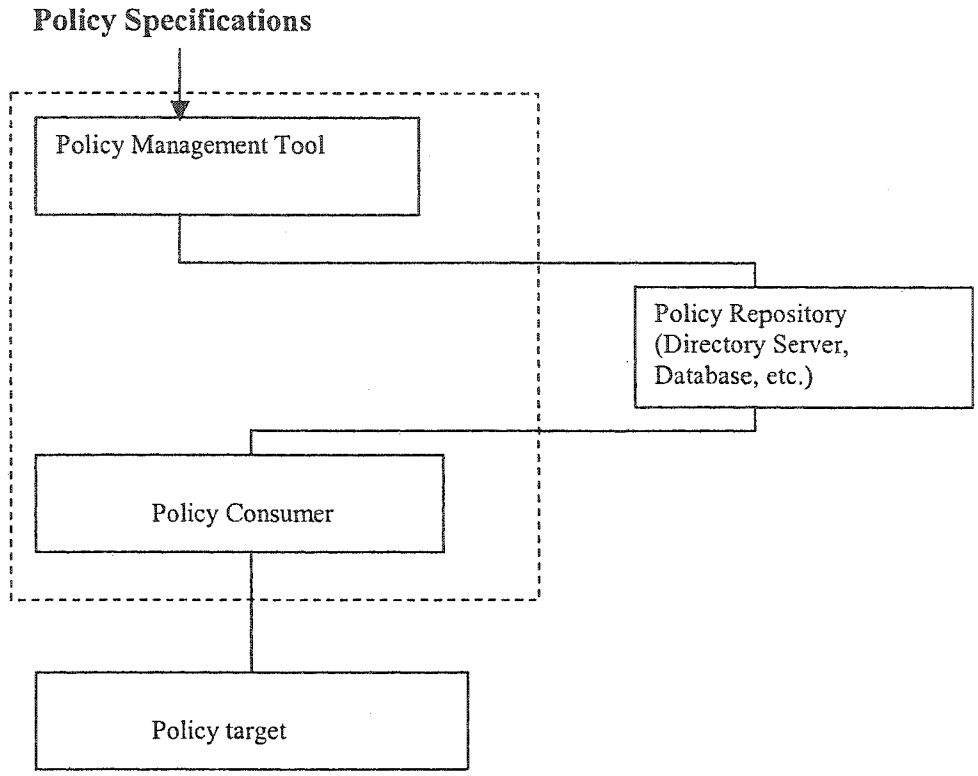


Figure 1.2 The functional components of a policy framework

A policy framework consists of a policy management tool, a policy repository, and policy consumers (Modified from Stevens et al., 1999b, see text for details).

working on tools and techniques for refining high level goals into implementable policies (Marriott and Sloman, 1996; Lupu and Sloman, 1997; Sloman and Lupu 1999; Moffet and Lupu, 1999)

Stevens and Weiss (1999a) discussed many aspects of policy-based management for IP networks. They suggested that an infrastructure should be built in order to really implement policy-based management. However, to build an infrastructure will be a very difficult task given so many different network vendors in the world. The best and fastest way to implement policy-based network management is to combine this policy idea with

the present network management protocol so that we can take advantage of policy-based management now and develop new protocols or infrastructure later. Three such examples are: policy-based management with SNMP (Mahon et al., 1999), policy-based load management (Hossain et al., 1999), and a policy-framework with integrated and differentiated network (Rajan et al., 1999).

These works have built a profound basis for studying policy based network management. Most of these agreed that a common infrastructure is needed to build a policy based management protocol and SNMP was suggested to be used as such infrastructure. Due to its popular and broad vendor supports, SNMP was chosen by IETF as the basic infrastructure to build a new policy based network configuration system. However, some problems have to be solved before this protocol can be implemented. One problem with this approach is that the existing network devices do not support the new policy MIB and there is no configuration MIB available now. Another problem is how to communicate between the policy supported network and policy unsupported network such as telecommunication network where TL1 is still dominated (Lumos, 2001).

The object of the present study is to explore a new approach which can be used to support the deploying of policy based configuration in the existing network and make policy execution atomic. The thesis consists of eight chapters. Chapter 1, which is the present chapter, reviews the progress in network management. Chapter 2 introduces SNMP protocols. Chapter 3 presents policy - based network management with SNMP (SNMPCONF). Chapter 4 shows the approach proposed by this study and its implementation. Chapter 5 presents details of the present implementation. Chapter 6

shows how to use CLI, SNMP, and SNMPCONF to configure networks by an example network. Chapter 7 discuses the advantages of the present implementation. Finally, Chapter 8 summarizes the results from the present study.

Chapter 2

Simple Network Management Protocol (SNMP)

SNMP is a management protocol originally designed for the management of TCP/IP capable data communication network devices. It uses the manager/agent paradigm. The guiding principle for its design was simplicity, to ensure agent implementations could easily be made available.

The SNMP protocol is defined by a series of RFCs. The overall architecture of SNMP framework is described in RFC 2571 (Harrington, Presuhn, and Wijnen, 1999). As discussed by Stevens (1990, pp359-361) and Waldbusser et al. (2001), the SNMP protocol consists of three foundational specifications. These three foundations are

1) A Management Information Base (MIB) to describe the network environment. The information defined in the MIB is accessible and changeable to the manager. A lot of MIBs that can be used for different purposes have been defined since SNMP protocol defined (see Appendix B for details)

2) A set of common structure and an identification scheme used to reference the variables in the MIB, which is called Structure of Management Information (SMI). The first version is specified in RFC 1155 (Rose and McCloghrie, 1990). The second version, called SMIv2, is described in STD 58, RFC 2578, 2579, and 2580 (McCloghrie et al., 1999a, 1999b, 1999c).

3) Message protocols for transferring management information between the manager and the devices. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157(Case et al., 1990). A second version of the SNMP message protocol is called SNMPv2c and described in RFC 1901 and RFC 1906 (Case et al., 1996a, b). The third version is called SNMPv3 and described in RFC 1906 (Case et al., 1996b), RFC 2572 (Case et al., 1999), and RFC 2574 (Blumenthal and Wijnen, 1999).

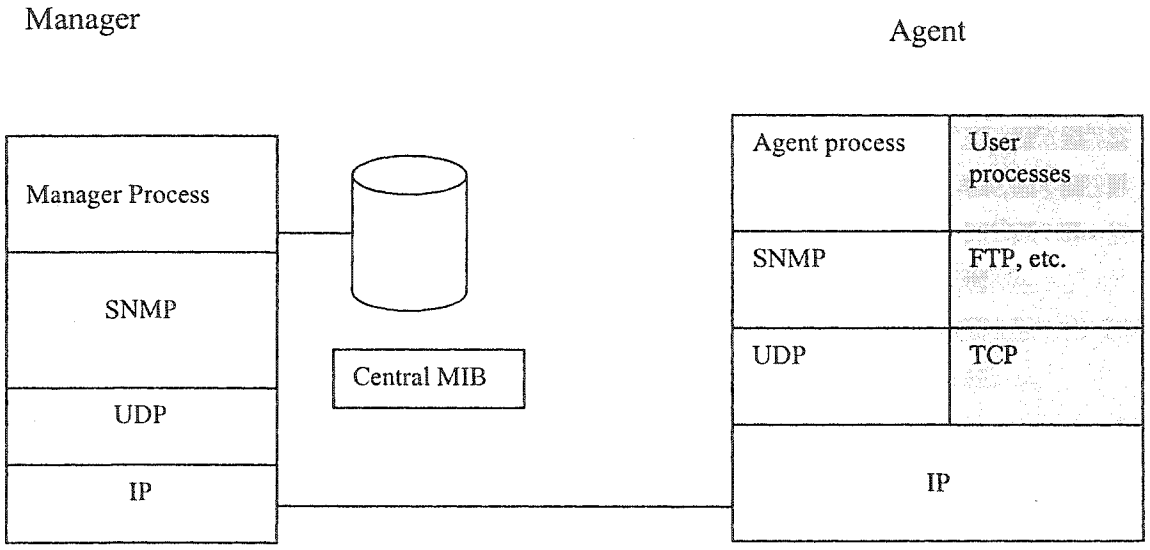


Figure 2.1 A simple SNMP model

Figure 2.1 shows that the SNMP protocol employs the Manager/agent paradigm (Modified from Stallings, 1996,p79, Figure 4.1).

The simple SNMP model is sketched in Figure 2.1. As shown in Figure 2.1, the communication between the manager and the agent is controlled by SNMP. SNMP is

implemented on the top of TCP/IP. Each manager process controls access to a central MIB at the management station and provides an interface to the network manager. Each agent process interprets the SNMP messages and controls the agent's MIB. The shaded portion in the figure depicts the operational environment-that which is to be managed. The unshaded portions provide support to the network management functions (Stallings, 1996, p78).

2.1 The SNMP protocol

SNMP generally uses UDP to transport information. However if necessary, TCP might be used to prevent information loss. Two ports, 161 and 162, are reserved for SNMP. The manager sends its request to the agent through port 161 and the agent sends the traps to the manager through port 162. Therefore a SNMP system can be run as both manger and agent by using two different ports.

SNMPv1 defines five types of messages: the get-request operator, the get-next-request operator, the set-request-operator, the get-response operator, and the trap operator. The first three messages are sent from the manager to the agent, and the last two are from agent to the manager. (The first three will be referred as the GET, GET-NEXT, and SET operator later). SNMPv2 added two new operators: get-bulk-request, which is more efficient than get-next-request, and information-request, which offers a communication between the manager and the manger.

The SNMP message format has been changed from SNMP v1 to SNMP v3. The SNMP v3 message contains fields for global data (such as SNMP version, the message

identifier, the maximum message size, the security model and the level of security), fields for the security model information, and fields for naming scope (context identifier and name) and finally the PDU (Figure 2.2). The SNMP version identifies the version of the SNMP protocol in use. SNMP engines use the message identifier (MsgID) to coordinate the processing of the message by different portion of the framework. Max Size is the maxim message size supported by the sender of the message. Multiple security models may exist concurrently in the SNMP entity, the Security Model specified which security model used. The context ID and Context ID defines the engine which realizes the managed objects referenced. Finally, the PDU (Packet Data Unit) stores the message.

<table>
<tr><td colspan="5">Header Data</td><td rowspan="2">Security Parameters (Model Specific)</td><td colspan="3">Scoped PDU data</td></tr>
<tr><td>SNMP version</td><td>MsgID</td><td>Max Size</td><td>Flags</td><td>Security Model</td><td>Context EngineID</td><td>Context Name</td><td>PDU data</td></tr>
</table>

Figure 2.2 SNMPv3 message format (modified from Cherkaoui et al., 1999)

SNMP version	defines which module will decode the packet.
MsgID	message identifier, used to match a request with a response.
Max Size	Maximum size of the response packet, some equipment may have limited buffers and don't respond well to large responses.
Flags	indicate if a response is expected and a privacy module is used.
Security Model	used to dispatch the packet to the right security module.
Security Parameter	security protocol agreement.
Context EngineID	identifies a context.
Context Name	identifies a context.
PDU data	actual payload, requested variables or repaired values.

2.2 Management Information Base (MIB)

Each SNMP agent maintains a Management Information Base or MIB. When receiving a query from the manager, the agent must check the MIB, obtain or set MIB, and then send the requested information to the manager. The MIB is defined and constructed in according with the Structure of Management Information (SMI) specified in RFC 1155 (Rose and McCloghrie, 1990). The SMI identifies the data types that can be used in the MIB and specifies how resources within the MIB are represented and named.

Each managed object in MIB has a unique identifier, called an MIB Object Identifier (OID). To reference an MIB object, the object identifier must be used. For every MIB object there exists an MIB definition that defines the managed object. The definition covers a variety of attributes, including the object identifiers, the syntax type, access permission, description, and instance information (Stevens, 1990, p365).

An object identifier is a data type specifying an authoritatively named object and is represented by a sequence of integers separated by decimal points. These integers are linked by a tree structure, which is similar to the DNS or a UNIX file system. At the top of this tree, the object identifiers start from an unnamed root (Figure 2.3). Each node in the tree is also given a textual name. For example, the name corresponding to the object identifier 1.3.6.1.2.1 is iso.org.dod.internet.mgmt.mib. However, these textual names are used for human readability only. In the PDU exchanged between the manager and agents, the numeric object identifiers, instead of textual names, are used.

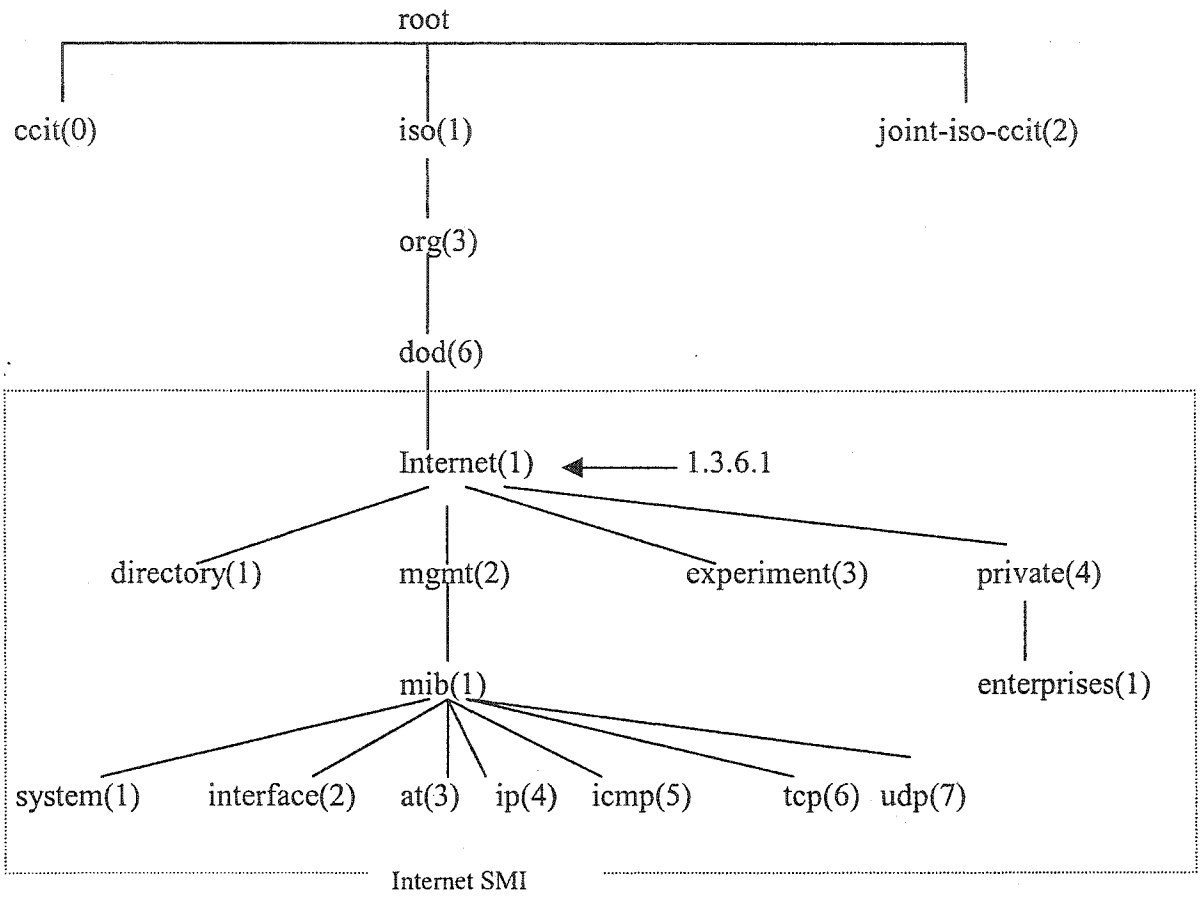


Figure 2.3 Object Identifiers in the Management Information Base

(Modified from Stevens, 1990)

As shown in Figure 2.3, the internet node has the object identifier value of 1.3.6.1. This value serves as the prefix for the nodes at the next lower level of the tree. The SMI document defines four nodes under the internet node:

directory: reserved for future use with OSI directory(X>500).

mgmt: used for object identified in IAB-approved documents.

experimental: used to identify objects used in Internet experiments.

private: used to identify objects defined unilaterally.

The mgmt subtree contains the definitions of MIBs that have been approved by the IAB. At present, two versions of the MIB have been developed, MIB-1 and MIB-2. The second MIB is an extension of the first. Both are provided with the same object identifier in the subtree since only one of the MIBs would be present in any configuration. In the mean time, a set of special MIBs are defined for special purposes (see Appendix A for more information). In the private sub tree, the vendors can define their own MIBs under the node Enterprise.

2.3 Evolution of SNMP

Although the network management as an idea has been around as long as networks, the SNMP did not come to life until 1980. The process actually began in 1987 at a gathering of networking leaders (Stallings, 1996; Simuneau, 1999). Since then, the protocol has been improved in many aspects and evolved from SNMP v1 to SNMP v3.

SNMP v1 had two advantages over the other management protocols:

1) SNMP, its associated Structure of Management Information (SMI), and its Management Information Base (MIB) are quite simple and therefore can be easily and quickly implemented.
2) SNMP is based on the Simple Gateway Monitoring Protocol (SGMP), for which a great deal of operational experience had been gained.

But the SNMP v1 protocol has the following deficiencies:

1) Inefficient to transfer a large amount of management information
2) Lack of error status codes

3) Lack of security and privacy control. SNMPv1 has no capability to authenticate the source of a management message or to prevent eavesdropping.

SNMPv2 addresses many of the deficiencies of SNMPv1. The key enhancements to SNMP that are provided in SNMPv2 fall into the following categories:

1) Extended the Structure of Management information (SMI) of SNMPv1
2) Added Manager to manager capability
3) Added two new protocol operations: get-bulk-request and information-request.

Although the SNMP v2 work took a large step in the right direction, it failed to meet the original design goals of the project. The most noticeable gap was in security and administration.

SNMPv3 added the following administrative features:

1) Expanded the authentication service to include privacy.
2) Specified a View-Based Access Control Model (VACM) which adds granular access control to all managed devices, providing that control down to the MIB objects level. This lets each SNMP agent enforce access policies.
3) The alternative User-based Security Model (USM) offers user-specific authentication of individual SNMP packets. It supports message digest to verify packet integrity. It provides timestamp verification to protect against replay attacks. By using DES encryption, it protects management data from eavesdropping.

4) Added unique name (snmpEgnineID) for each SNMP device. This helps with packet authentication and mapping the relationships between agents.

5) Let managers update SNMP device configurations automatically using SNMP.

2.4 Implementation

There are many different implementations of the SNMP protocol, but few with the object-oriented approach. One of them is SNMP++ by Mellquist (1997) from Hewlett-Packard Company. SNMP++ was implemented using C++ and has been tested on Windows, Linux, HPUX, and Solaris environment. The source code for SNMP++ can be downloaded from http://rosegarden.external.hp.com/snmp++. However, one drawback is that this API needs to be compiler according to the operating system because of C++.

Another one is Modular SNMP from Universite du Quebec a Montreal (Cherkaoui et al., 1998). Modular SNMP was implemented in Java and supports SNMP v1 and SNMP v3. The advantage of this implementation is that the users can select different modules and put them together to form a SNMP tool. The users can even create their own modules and build a new SNMP framework. The architecture of Modular SNMP can be represented by Figure 2.4 and Figure 2.5. From Figure 2.4, on the top of this framework is the class SnmpSessionManger. When applied this framework, users must first declare and create an instance of SnmpSessionManger as:

SnmpSessionManager ssmanager = new SnmpSessionManager();

Then he/she can configure the manager through the SnmpSessionManger API by the following steps:

//Adding the v1 processing module

Ssmanager.Enablev1(true);

Adding the SNMP v3 Processing Module with the appropriate encryption and authentication modules:

Ssmanager.Enablev3(true,USM,MD5,DES);

//Adding the VACM for the access control Ssmanager.EnableVACM(true);

Then start to run the program by:

//starts the session manager
ssmanager.start();

Once we have a reference of a SnmpSessionManager (let's call it ssmanager), we can create as many of SnmpSession as we need.

//creating Sessions

SnmpSession session1 = new SnmpSession(ssmanager);

SnmpSession session2 = new SnmpSession(ssmanager,(SnmpClient)this);

Then, we can open a session:

ssmanager.OpenSession(session1);

As soon as a session is built, we can use SNMP commands such as Get, SET to manipulate MIBs.

It is evident that Modular SNMP can be easily extended to support any new protocol based on SNMP, because of its module design and its excellent implementation.

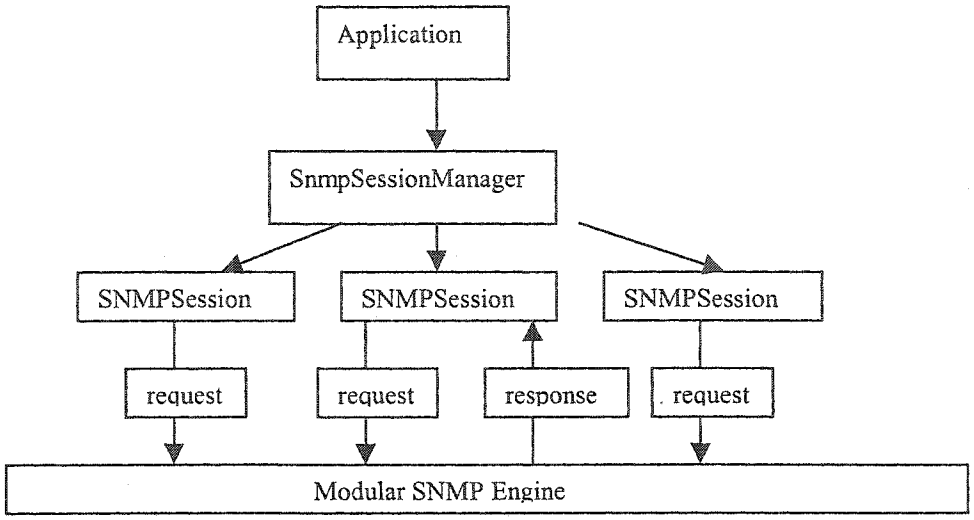


Figure 2.4 The application model of Modular SNMP framework

In this framework, applications will need a session manager to manage SNMP sessions which process SNMP requests or responses thorough a SNMP engine.

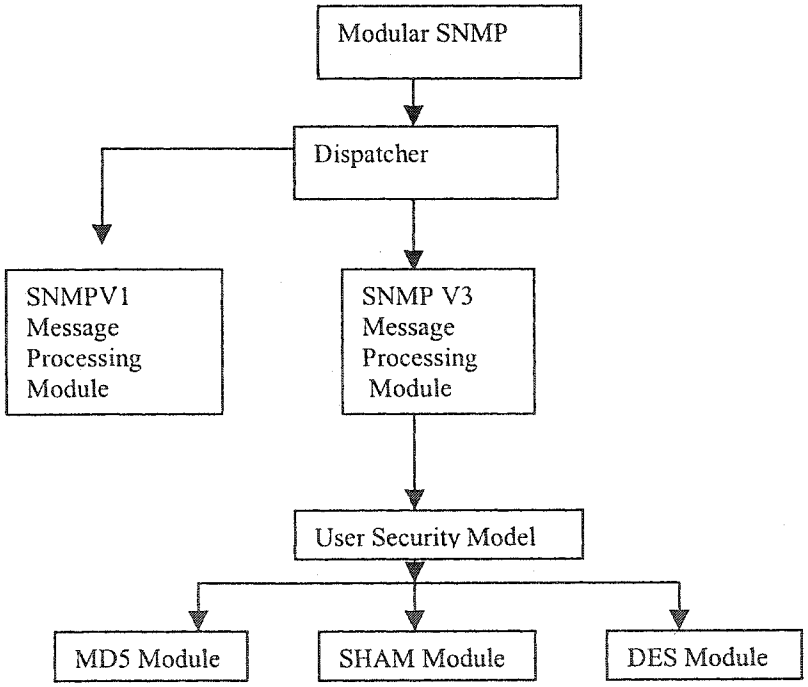


Figure 2.5 The structure of Modular SNMP framework

Modular SNMP processes the SNMP messages using a Dispatcher class that sends the message to SNMP v1 or SNMP v3 Module depending on the module of the messages.

Chapter 3

Policy Based Configuration with SNMP (SNMPCONF)

SNMP protocol was chosen as the basis to implement policy based network configuration due to its popularity and simplicity. SNMPCONF work group of IETF was formed to propose new MIBs which will perform policy based network configuration using SNMP protocol. Up to now, three internet drafts (Best current practice, MacFaden et al., 2001; policy MIB, Waldbusser et al., 2001; and DifferServ MIB, Hazewinkel et al., 2001) were proposed and improved with the work of the group progressing. We will discuss these drafts in details in the following sections.

3.1. Policy Language

In policy MIB draft, Waldbusser et al. (2001) define ten policy tables to register policy related information and a policy language. The policy language is formally defined as a subset of ISO C, but only allows simple data types (array, string, and integer), loop and conditional structure, and a set of predefined accessor functions. This language does not support function definition.

The definition expressed in the EBNF form as shown in Table 3.1:

Table 3.1 Policy language definition (from Waldbusser et al., 2001)

```
-- Lexical Grammar

    letter:         '_' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
                  | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm'
                  | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't'
                  | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'
                  | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
                  | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M'
                  | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T'
                  | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

    digit:          '0' | '1' | '2' | '3' | '4'
                  | '5' | '6' | '7' | '8' | '9'

    non_zero:     '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

    oct_digit:    '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'

    hex_digit:      digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
                          | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'

    escape_seq:     '\''    |   '\"'    |   '\?'    |   '\\'
                  | '\a'    |   '\b'    |   '\f'    |   '\n'
                  | '\r'    |   '\t'    |   '\v'
                  | '\' oct_digit+     | '\x' hex_digit+

    non_quote:    Any character in the UTF-8 character set
                  except single quote ('), double quote ("),
                  backslash ('\') or newline.

    c_char:             non_quote | '"' | escape_seq

    string_literal:     '"' s_char* '"'

    s_char:             non_quote | ''' | escape_seq

    char_constant:      ''' c_char '''

    decimal_constant:   non_zero digit*

    octal_constant:     '0' oct_digit*

    hex_constant:       ( '0x' | '0X' ) hex_digit+

    integer_constant:   decimal_constant | octal_constant | hex_constant

    identifier:         letter ( letter | digit )*

-- Phrase Structure Grammar

    -- Expressions
```

```
primary_expr:       identifier | integer_constant | char_constant
                  | string_literal  |  '(' expression ')'

postfix_expr:       primary_expr
                  | postfix_expr '(' argument_expression_list? ')'
                  | postfix_expr '++'
                  | postfix_expr '--'
                  | postfix_expr '[' expression ']'

argument_expression_list:
                    assignment_expr
                  | argument_expression_list ',' assignment_expr

unary_expr:         postfix_expr  |  unary_op unary_expr

unary_op:           '+' | '-' | '~' | '!' | '++' | '--'

binary_expr:  unary_expr | binary_expr binary_op unary_expr

binary_op:        '||' | '&&' | '|'  | '^'  | '&'  | '!='
                | '==' | '>=' | '<=' | '>'  | '<'  | '>>'
                | '<<' |  '-' | '+'  | '%'  | '/'  |  '*'

assignment_expr:      binary_expr
                    | unary_expr assignment_op assignment_expr

assignment_op:      '=' | '*='  | '/=' | '%=' | '+=' | '-='
                | '<<=' | '>>=' | '&=' | '^=' | '|='

expression:    assignment_expr | expression ',' assignment_expr

-- Declarations

declaration:        'var' declarator_list ';'

declarator_list:    init_declarator
                  | declarator_list ',' init_declarator

-- Statements

statement:    declaration
            | compound_statement
            | expression_statement
            | selection_statement
            | iteration_statement
            | jump_statement

compound_statement:     '{' statement* '}'

expression_statement:  expression? ';'

selection_statement:
        'if' '(' expression ')' statement
      | 'if' '(' expression ')' statement 'else' statement

iteration_statement:
```

```
        'while' '(' expression ')' statement
      | 'for' '(' expression? ';' expression? ';' expression? ')'
          statement

jump_statement:    'continue' ';'
                 | 'break' ';'
                 | 'return' expression? ';'

-- Root production

PolicyScript:     statement*
```

Basically, the policy language is a subset of the C language. Some features that have been removed from the C/C++ language are: function definitions, pointer variables, structures, enums, typedefs, floating point and pre-processor functions (except for comments). Another characteristic of this language is that all variables in a program are in the same scope. In other words, the variables are global by default. In order to enhance the functionality of this policy language, some accessor functions like ToString have been added.

3.2 Policy MIB

In the same draft, 10 policy related tables are defined to help policy-based management. As presented in the followings, these MIB are useful to manage elements, roles, and rules (policies) and their relations. Since the present implementation is focused on the policy language interpreter, our discussion will mainly involve the first two tables, i.e., the policy table and the policy code table. These 10 policy tables are:

1) Policy Table

2) Policy Code Table

3) Element Type Registration Table

4) Role Table

5) Capabilities Table

6) Schedule Table

7) Policy Tracking Table

8) Element-To-Policy Table

9) Policy Debug Table

10) Notification Registration Table

Among these tables, the policy table is used to register and hold policy parameters such as policy groups, schedules and pointers to the policy code table where policy rules (filter and action) are stored. Element registration table, as its name implied, is used to record element types. When a new element is discovered, its type is registered using the OID. The role table is a read-created table that organizes role strings and is sorted by element. This table is used to create and modify role strings and their associations as well as to allow a managed station to learn about the existence of roles and their associations. The capabilities table contains a description of the system capabilities. Policy script and the management station can apply polices to the system based on its capabilities. The schedule table controls the policy schedule. The left four tables are used to debug and trace policy.

To illustrate how to configure networks using the above tables, let's take a look at an example. This example is inspired by the building management example given in the best practice draft (MacFaden, Saperia, and Tackabury, 2001). Suppose that we want to use policy-based network management to manage all buildings in an university campus,

we would like to control the room temperature, light, humidity, and air quality (Carbon Dioxide, Carbon Monoxide, Oxygen and toxic gas content, etc.). The policies for temperature and light control might be define as two groups:

Group1 (temperature control)

1. If (a building temperature >20 °C) Then (turn on air conditioner)
2. If (a building temperature <15°C) then (turn on heating)

Group 2 (light control)

1. If (it is weekend night) then (turn off all lights in all class rooms)

.....

Table 3.2 The policy table for campus building management

pmPolicyIndex ,	1	2	3
pmPolicyGroup	Temperature control	Temperature control	Light control
pmPolicyPrecedence	0	1	1
pmPolicySchedule	1	1	2
PmPolicyElementTypeFilter	1.3.6.1.4.1.1.1.1	1.3.6.1.4.1.1.1.1	1.3.6.1.4.1.1.1.2
pmPolicyConditionScriptIndex ,	1	3	5
pmPolicyActionScriptIndex ,	2	4	6
pmPolicyParameters			
pmPolicyConditionMaxLatency ,	10000	10000	10000
PmPolicyActionMaxLatency ,	10000	10000	10000
pmPolicyMaxIterations ,	1000	1000	1000
pmPolicyDescription	If (a building temperature >20 °C) Then (turn on air conditioner)	If (a building temperature <15°C) then (turn on heating)	If (it is weekend night) then (turn off all lights in all class rooms)
pmPolicyMatches	0	0	0
pmPolicyAbnormalTerminations	0	0	0
pmPolicyExecutionErrors	0	0	0
pmPolicyDebugging	0	0	0
pmPolicyAdminStatus	1	1	1
pmPolicyStorageType	Non-volatile	Non-volatile	Non-volatile
PmPolicyRowStatus			

Table 3.3 The policy code table for the campus building management

pmPolicyCodeScriptIndex	pmPolicyCodeSegment	pmPolicyCodeText	pmPolicyCodeStatus
1	1	t1=Get() If t1>20 then Return true	
2	1	SnmpSet(1.3.4.1..4.1.1.1.1 , 2)	
3	1	t1=Get() If t1<20 then Return true	
4	1	SNMPSet(1.3.4.1..4.1.1.1. 1, 1)	
5	1	If (day=Friday, Saturday, or Sunday) and time>19:00 pm	
6	1	SNMPGET(oid, all class room light) SNMSet (oid light, 0)	

Table 3.4 A hypothetical air conditioner (1.3.6.1.4.1.1.1.1) MIB

AirConditionerStatus(1.3.6.1.4.1.1.1.1)
0
1
2

0-air conditioner off;

1-air conditioner is on and heating;

2-air conditioner is on and cooling

Table 3.5 A hypothetical light controller (1.3.6.1.4.1.1.1.2) MIB

LightControllerStatus(1.3.6.1.4.1.1.1.2)
0
1
1

0-light off, 1-light on

These policies should be registered in the policy tables (Table 3.2 and Table 3.3). Table 3.2 and 3.3 present the above policy groups and their policy scripts respectively. Table 3.4 and 3.5 show a faked MIB for air conditioners and light switches. In Table 3.2, the pmPolicyIndex is a unique index for each policy entry in the table. We have three policies defined and they are registered with pmPolicyIndex from 1 to 3 under two policy groups (Table 3.2). The pmPolicyPrecedence is used to arbitrate the policy conflict. If one element meets several policy conditions (policy filters) and thus several policy actions should be executed against the same element. However, these policies might have conflicts with each other. In this case, only the policy with the highest policy precedence should be enforced. If a policy is enforced, its action can be deployed all the time or at a certain time only. If the policy action is activated only at a certain time, then the execution time is managed by the schedule table. In the policy table, the policy schedule is not directly recorded, but only as an index (pmPolicySchedule) which points to an entry in the schedule table. pmPolicyElementTypeFilter specifies the element types on which the policy will be executed. In our example, for no standard MIB defined, we use a faked OID to indicate the temperature controllers (1.3.6.1.4.1.1.1.1) and light controllers (1.3.6.1.4.1.1.1.2). These two OIDs are under enterprise node in the OID tree (see Figure 2.3). pmPolicyConditionScriptIndex and pmPolicyActionScriptIndex are pointers which indicated where the scripts are stored in the script table (Table 3.3). Our first policy:

If (a building temperature >20 °C) Then (turn on air conditioner)

is recorded as two entries in the policy script table (pmPolicyCodeScriptIndex=1 and 2, Table 3.3) as policy conditions and policy action respectively.

pmPolicyConditionMaxLatency and pmPolicyActionMaxLatency define the interval between the time policy will be re- executed and the time when the policy last enforced in milliseconds. pmPolicyMaxIteration is used to limit the maximum times a loop may be executed and prevent infinite loop. In the present example, we use 1000 as the upper limit. pmPolicyDescription explains the policy using the terms that the human manger can understand. Normally, this is where we put the policy in plain English or other natural language. pmPolicyMatches indicates how many elements match the policy condition in the most recent policy execution. Since our policy has not been executed, this is set to 0. pmPolicyAbnormalTermination and pmPolicyExecutionError record the status of the policy execution such as how many times the policy abnormal terminated and errors occur during the policy execution. pmPolicyAdminStatus indicates the administrative status of the policy and will be changed as the pmPolicyRowStatus changes. It is an integer with value of 0 (Disenabled), 1 (Enabled), or 2 (EnableAuotremove). pmPolicyStorageType determines whether the policy is stored in permanent storage or in the volatile memory. pmPolicyRowStatus indicates the policy status.

The policy scripts are recorded in the policy code table. The pmPolicyScriptCodeIndex is a unique index for each piece of policy code and presented as pointers in the policy table. Following these pointers from the policy table, the related policy code should be easily retrieved from the policy code table.

pmPolicyCodeSegment and pmPolicyCodeText store the policy segments and its scripts respectively.

3.3 Policy Execution

Policy execution is performed in three steps: Element Discovery, Element Filtering, and Policy Enforcement.

1) Element Discovery

As defined in the draft, an element is a uniquely addressable entity on a managed device. The manager should know each element it managed and has to "discovery" each new element. The element type registration table is used for the manager to "remember" what element types are being managed by the system and to register new types if necessary. An element type is registered by providing the OID of an SNMP object (i.e., without the instance). Each SNMP instance that exists under that object is a distinct element.

If a new type of air conditioners is installed in an office in our example, the manger will discover this new element by routine check or by the notification sent by the agent. The manager will then register this new element into the element type registration table and check if any policy would be applicable to this new element type immediately. If the element is a member of the set that the policy acts upon, the associated policy action will be executed instantly.

2) Element Filtering

Element Filtering is used to check which elements match the policy filter. To evaluate a policy, the policy filter is first called for each element and run to complete. This process is called "Element Filtering". The element address is the only state that is passed to the filter code for each invocation. If any syntax or processing error occurs, the filter will terminate immediately for this element. If the filter returns TRUE, the corresponding policy action will be executed for this element.

If an element matches a filter and had not matched that filter in the last time it was checked (or it is a newly-discovered element), the associated policy action will be executed immediately. If the element had matched the filter at the last check, it will remain in the set of elements whose policy action will be run within the pmPolicyFilterActionMaxLatency.

In the Element Filtering process, all the policy condition scripts in our building management example will be run against all air conditioners and light controllers. For air conditioners, those with temperature higher than 20 °C will return TRUE when the policy with pmPolicyIndex 1 is applied, but will give a FALSE if the policy with pmPolicyIndex 2 is checked.

3) Policy Enforcement

For each element that has returned TRUE from the policy filter, the corresponding policy action is called. The element address is the only state that is passed to the policy

action for each invocation. If any syntax or processing error occurs, the action will terminate immediately for this element.

In the given example, during Policy Enforcing process, all the air conditioners which meet the requirements (temperature higher than 20°C or lower than 20°C) will be turned on to heat or cool the room.

3.4 The Reference Models

In order to release the network manager from details of the devices, the policy management should present only the necessary information. However, to facilitate fault resolution and performance management, the more information is offered the easier the management would be. This requires the policy management to support different levels of abstraction. From high level of abstraction to the low level, more details will be revealed. The draft (Best current practice, MacFaden et al., 2001) presents four levels of abstraction.

3.4.1 Domain-Specific

Domain-specific is the highest abstract level in present policy framework of SNMPCONF. A domain is a general area of technology such as service quality or security. When domain referred, they will most often be discussed with technology or application-specific examples. Examples of technical domains include, IPSec and Differentiated Services. When expressed in terms specific to a particular domain, a policy is said to be at the Domain Specific level of detail.

3.4.2 Mechanism-Specific

Mechanisms are technologies used within a particular domain. For example, in the differentiated services domain, RED (Random Early Detection) or WRED (Weighted Random Early Detection) might be used as one of the mechanisms that devices employ to realize a traffic conditioner, called a Dropper, in differentiated services (DS) and the applications on which they rely. Policy descriptions that include the details associated with a particular mechanism, are said to be Mechanism-specific.

3.4.3 Implementation-Specific

Implementation-specific gives details of parameters that a particular vendor might use in an implementation to augment a standard set of mechanism-specific parameters. Very often vendors add special capabilities to basic mechanisms as a way of meeting special customer requirements or differentiating themselves from their competitors. These special capabilities are related to the implementation approach that a vendor has used, thus the term "implementation-specific". For example, if a router vendor implemented a particular routing protocol, they would have the mechanism-specific parameters that control the behavior of that software. The vendor might have chosen to run several instances of that routing protocol, perhaps on different processors, for performance reasons. The parameters that are used to control the distribution of work on the different processors for that protocol would be implementation-specific.

3.4.4 Instance-Specific

Network operators are most familiar and comfortable with information of this instance-specific type. Instance-specific information refers to parameter values that have been associated with a specific instance in a managed element.

For example, the Border Gateway Protocol is an exterior routing protocol that has a number of parameters to describe information other routers that is sharing information with. One such parameter defined in the BGP MIB module (BGPMIB) is the desired connection retry interval for a peer, bgpPeerConnectRetryInterval. An example value would be 120 (seconds). When expressed with this level of specificity, one would say that this is mechanism-specific data. However, if it is presented as bgpPeerConnectRetryInterval 192.0.2.1 = 120 we would be looking at the retry interval of the peer router found at IP address 192.0.2.1. The value for this instance is 120 seconds, instance-specific data.

These four abstraction levels might be useful to help us design policy MIB and design policy management software, but cannot be used as a standard, because there is no clear boundary between these levels.

The building management described in the above again can be served as an example to explain the four abstract levels. In the Domain-Specific level, we could have Temperature, Light, Humidity, and Air Quality as the domains. In each domain, we might have different mechanism-specific policy. If we want control temperature, we at least have two different mechanisms to change temperature: heating to increase the temperature and air-conditioning to lower temperature. In terms of heating, different vendor implement different methods: electric heating, oil-heating, gas-heating, etc....

Even made by the same manufacture, the heating equipment might have some parameters, for example, number of buttons, which are different from the other equipments, because of the different models. These parameters (buttons, etc.) might be used to manage instance-specific policies. Therefore, in the different levels, there are different forms of the policies, even though the goals of these policies are the same. This can be shown in Figure 3.1. Suppose that we want to define polices to control temperature, in the Domain-specific level, the policy may use the following form:

If (cool) Then (increase the temperature)

The same policy in the mechanism specific level can be represented as:

If (temperature is low) Then (heating until the temperature to normal)

If it is in the implementation specific level, it will take the following form:

If (temperature <20 °C) Then (turn on electric heating equipments until the temperature to 20 °C)

Or more specifically, in instance-specific level:

If (temperature meter <20) Then (turn on the power of the electric heating equipments, connect the temperature meter, heat and measure the temperature until the temperature >=20)

In this chapter, the policy framework of SNMPCONF is presented. The core of this draft is the policy script language and policy MIB. The application of policy MIB is relied on the vendors' support and many other configuration MIBs have to be defined. In the present network, there is no configuration MIB available. We need an approach to

support the SNMPCONF without defining new MIB if we want to deploy the SNMPCONF to the existing network.

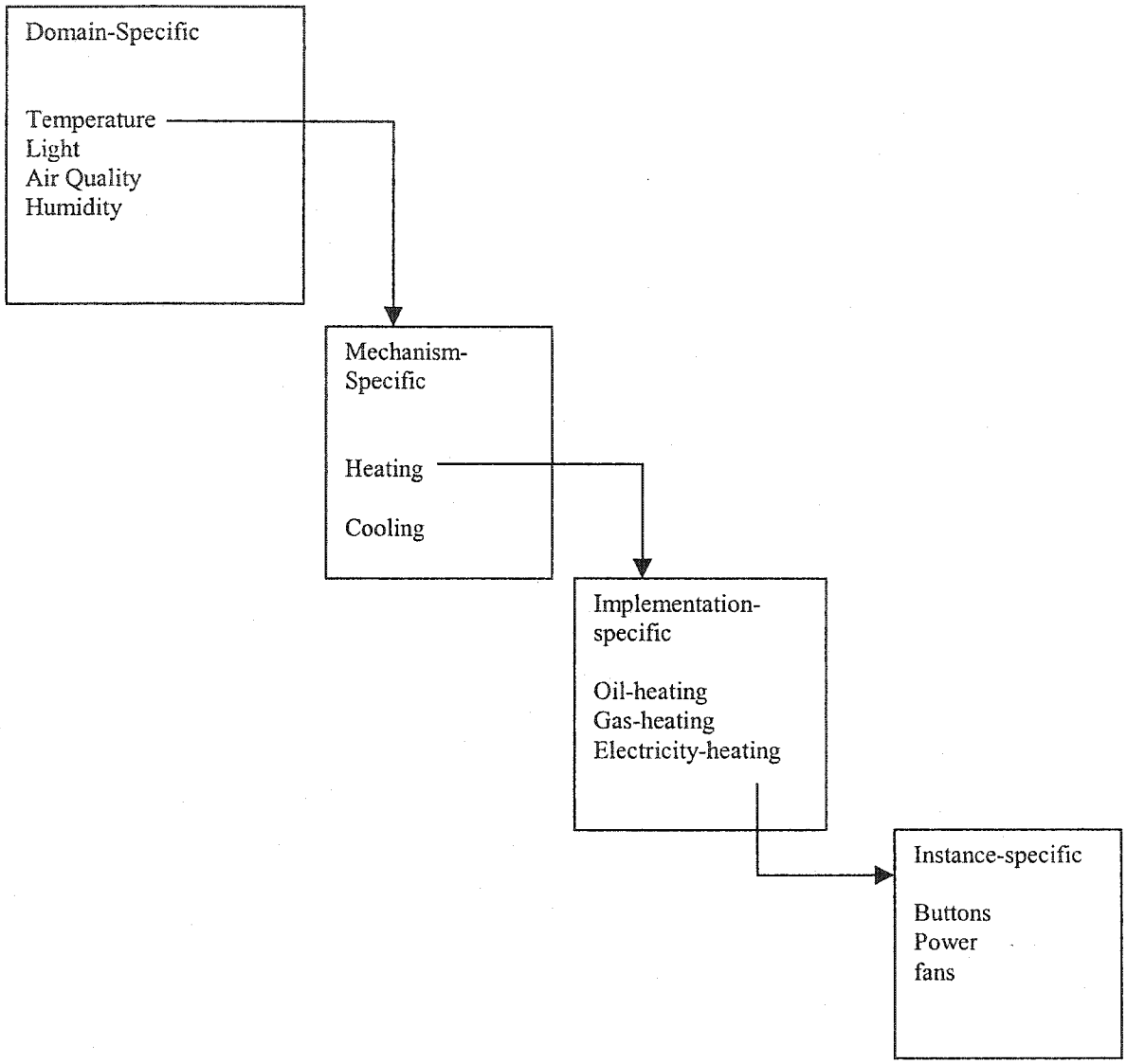


Figure 3.1 The four abstract levels of policies for Building management

Figure 3.1 shows that in different abstraction levels, the policy with the same goal may have different forms using different MIBs (see text for details).

Chapter 4

A Policy-based Network Configuration Framework

As described in the previous chapter, SNMPCONF drafts define the policy execution processes, policy language, two MIBs (Policy MIB and differserv MIB), and application models. In this chapter, we will discuss our approach to support SNMPCONF on the existing network and present the implementation of this approach.

4.1 Problems with Deploying SNMPCONF in Existed Network

From the discussions in the previous chapters, SNMPCONF basically is an extension of the SNMP protocol. It enhances SNMP protocol by providing new policy MIBs and a script language to define the policy. The policy MIB defined in the present drafts can be used to manage the policies only. The policy enforcement is accomplished through operations on MIB. Unfortunately, the MIB in present SNMP (v1 to v3) is designed to monitor the network operation and not suitable to do configurations. A preliminary search found about 78 RFCs defining MIB and about 58 different sets of MIB have been defined so far (see Appendix A). From Appendix A, most of the defined MIB can be used to monitor the network status only. A set of configuration MIB, which has not been defined yet, has to be defined before the SNMPCONF can be utilized to configure the network. However, even if the desired configuration MIB is defined, we still have to make the existed network to support these MIB, because the existing network might not be able to support the new MIB due to resource limits (for example, memory limitation). On the other hand, the existing network devices have many different vendor-

specific functions, which are handy to configure the device but cannot be used or is very difficult to use through MIB. Thirdly, although the existing network devices are not compatible with policy MIB or new configuration MIB, in the near future, these MIBs would be supported by new devices. Therefore, if we can find an alternative solution to utilize these vendor-specific functions to support the SNMPCONF, we may be able to deploy SNMPCONF in the present network without purchasing new hardware and this implementation can also be used without modification even new device deployed.

4.2 CLI and TL1

As pointed out by McFaden et al. (2001), CLI is one of most extensively used methods to configure data network today. CLI stands for Command Line Interface, which allows users to perform network operations for any device in the data network through Telnet or TFTP. However, CLI is generally vendor specific, i.e., similar devices from different vendors may have very different CLI.

A typical process to configure network device using CLI commands generally-involves the following three steps:

1) Login with a proper password. Depending on the users' privilege determined by users' password, the user may have no right to configure the device at all.
2) Execute a series commands to enter the right mode and configuration commands to configure or reconfigure the device. In Cisco router, if one wants to configure an interface, he/she should first login as privileged user, enter Privileged Exec Mode using "Enable"; then enter Global Configuration

Mode and finally the Interface Configuration Mode; only in the Interface Configuration Mode, the user can enter commands to configure.

3) Use "Show" command to check if the parameters are changed to the desired values

This process can be shown in Figure 4.1. From the sketch (Figure 4.1), a series of CLI commands to configure the device can be wrapped into an accessor function in the policy language just like the other accessor functions specified by SNMPCONF. The idea is that if we can store password and the other parameters to be changed as the arguments of the function and the information obtained from "SHOW" as a return from the function, then a series of CLI commands can be represented as a specified function in policy language.

After comparing CLI from Cisco with that from Nortel network, we found that the above observation is true for devices from both vendors. Considering most network devices in the market nowadays made by these two vendors, if CLI commands are represented as accessor functions, then they can be used to define policies directly. A policy interpreter is needed to translate the functions into specified CLI commands according to the network devices for this purpose. This process can be shown as Figure 4.2.

If CLI can be used to support SNMPCONF, we may need also to support TL1 for TL1 is extensively applied in the telecommunication network. TL1 (Transaction Language 1) is a set of ASCII-based instructions, or "messages", that an operation support system uses to manage a network element and its resources in telecommunications network. TL1, developed by Bellcore, is the dominant management

protocol for controlling telecommunications networks in North America nowadays and remains the only widely-implemented, vendor-independent telecommunications management protocol (Lumos, 2001), just like SNMP in the enterprise network. Given the fact that TL1 and CLI dominate the network devices, if we can implement this idea to wrap CLI and TL1 commands, we can realize policy configuration without defining a configuration MIB.

4.3 A Policy Based Network Configuration Framework

As discussed in the above, in order to deploy SNMPCONF in the existed network where no configuration MIB is currently supported, we may need to wrap CLI commands and TL1, therefore a policy interpreter should be defined. For SNMPCONF, SNMP should be used as the basic protocol and SNMPCONF is an application of SNMP. Hence, in a framework to implement SNMPCONF in the present network, an ideal structure might be described by Figure 4.3. As presented in Figure 4.3, the policy MIB and CLI commands are both supported by the interpreter. If a network device does not support SNMPCONF, the policy configuration will be transformed into CLI commands that the device embedded. Through CLI commands, the policy configuration will be enforced in this device in the same fashion as in the other devices that support the policy MIB.

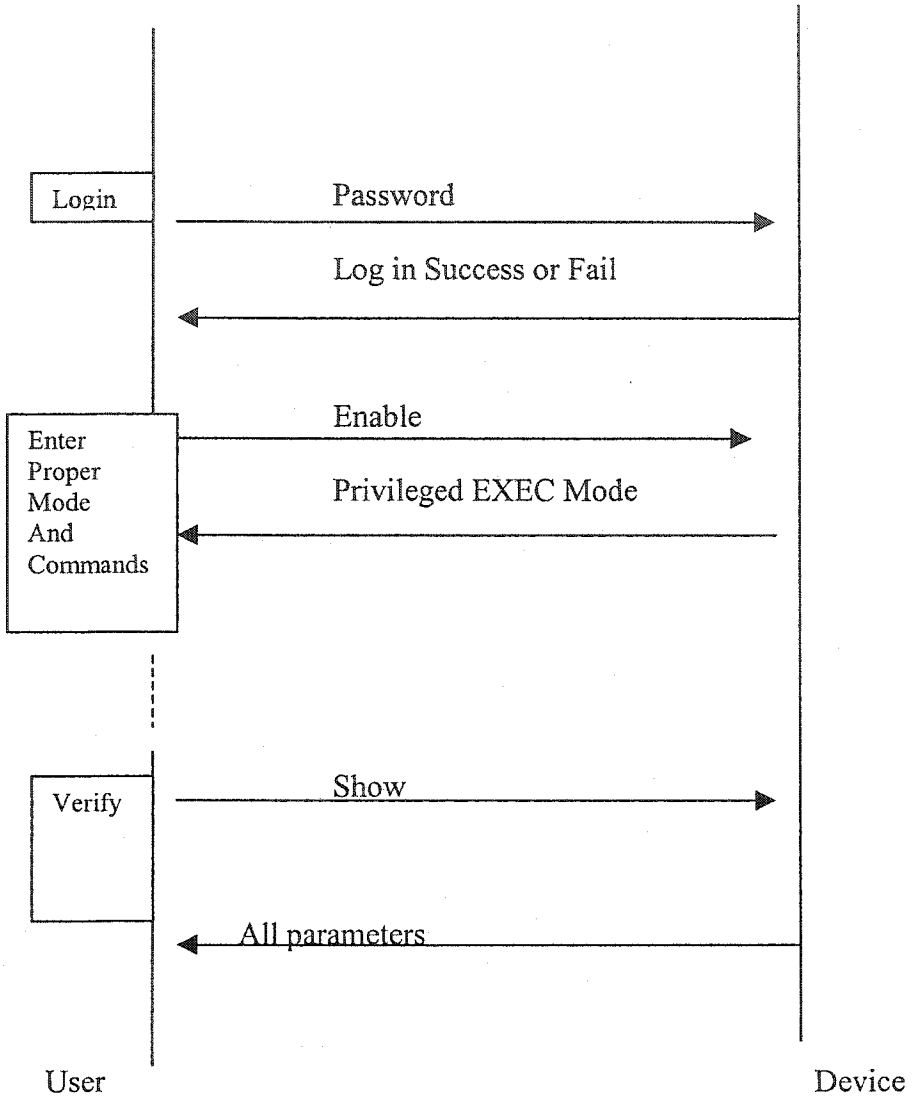


Figure 4.1 The process to configure devices using CLI commands

If CLI is used to configure a device, the user should take three steps: login, change parameters, and verify the changes through a Telnet session. These steps can be wrapped as a function in which the parameters are taken as the function arguments and verified result as the return from the function.

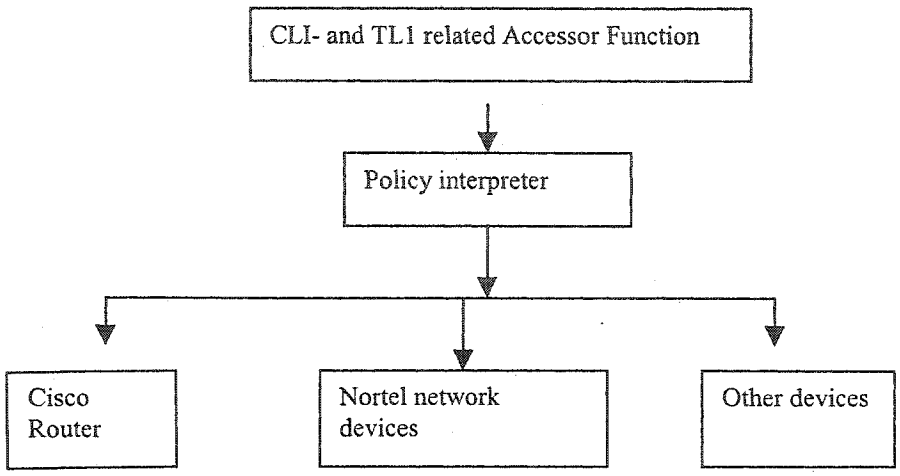


Figure 4.2 Wrapping CLI and TL1 commands into accessor functions

CLI commands from different vendors with different syntax but similar functionality might be wrapped into one accessor function. During policy execution, this function will be interpreted by the policy interpreter, in accordance with the device which the policy interpreter talks to.

In this design (Figure 4.3), the network administrator will design and input policies into the Policy Manager. The Policy Manager will then send the policies to the Policy Gateway through SNMP. In the Policy Gateway, the policy will be interpreted to CLI commands and these CLI commands will be sent to the managed devices to execute if the devices support no policy MIB or configuration MIB. However, if the policy MIB and configuration MIB are implemented in the device, the policy actions will be taken to these devices directly.

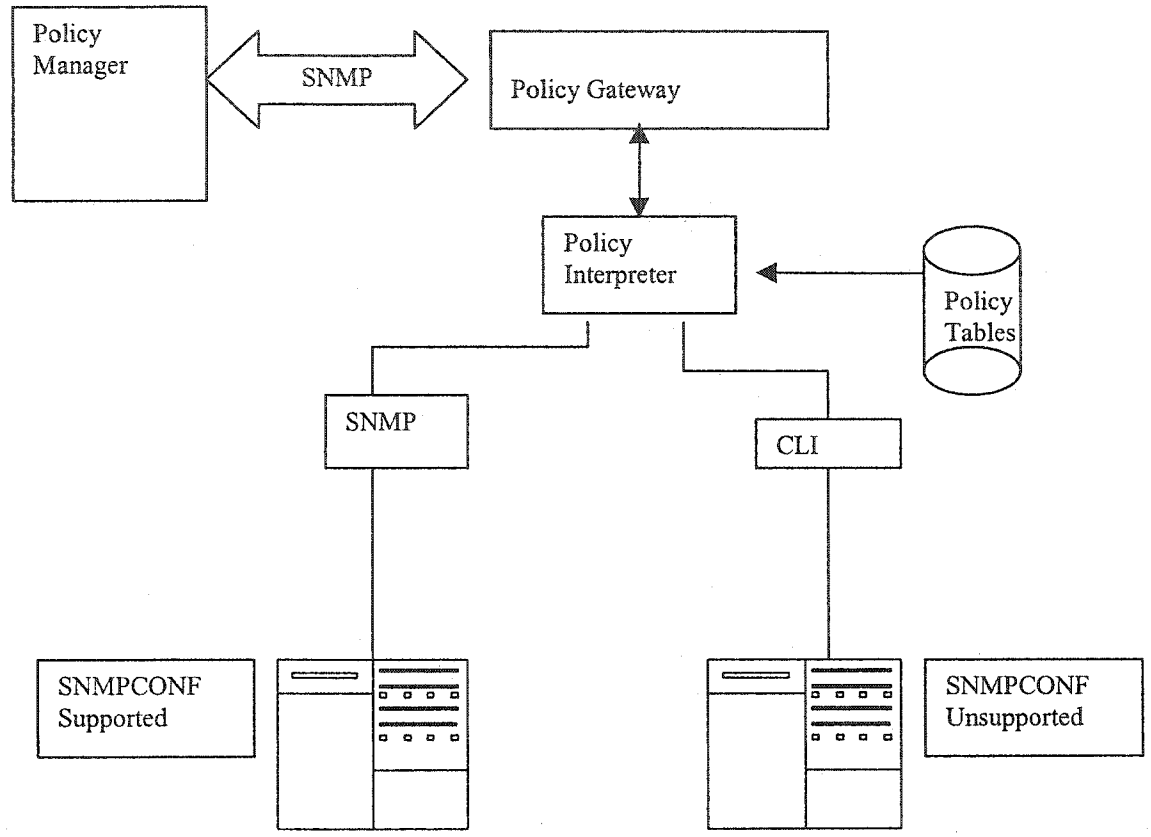


Figure 4.3 The architecture of a policy-based network configuration framework

SNMP is used as a basic link between the policy manager and policy gateway which will take SNMPCONF messages and send them to SNMPCONF-supported devices or translate them into CLI commands to SNMPCONF-unsupported devices.

In the above design, the Policy Interpreter is the key part of the framework. More details should be specified for the interpreter. An interpreter is a special kind of compiler (Aho et al., 1974,1986; Mak, 1996, Appel, 1997). As summarized by Aho et al., 1986) a compiler consists of the following parts (Figure 4.4):

1. Lexer, which breaks the source code into tokens;
2. Parser, which analyzes the phrase structure of the program;
3. Intermediate generator, which uses the structure produced by parser to create a stream of simple instructions;
4. Code optimizer is an optional phase designed to improve the intermediate code so that the ultimate object program runs fast and or/take less space;
5. Code generator, produces the object code by deciding on the memory locations for data, select code to access each data and the registers in which each computation to be done.

Source programs are taken as input to the compiler and the compiler produces a target program. In contrast, an interpreter just simply executes the source code or intermediate code and will not produce any target program. Normally, the interpreter will not do any optimizing.

The Policy Interpreter should have the same structure. However, as specified by the SNMPCONF, the policy interpreter is not necessary to process any error and the policy execution should stop immediately if any error occurs. Therefore our Policy Interpreter should be very simple. It will consist of a lexer, a parser and a code executor as shown in Figure 4.5.

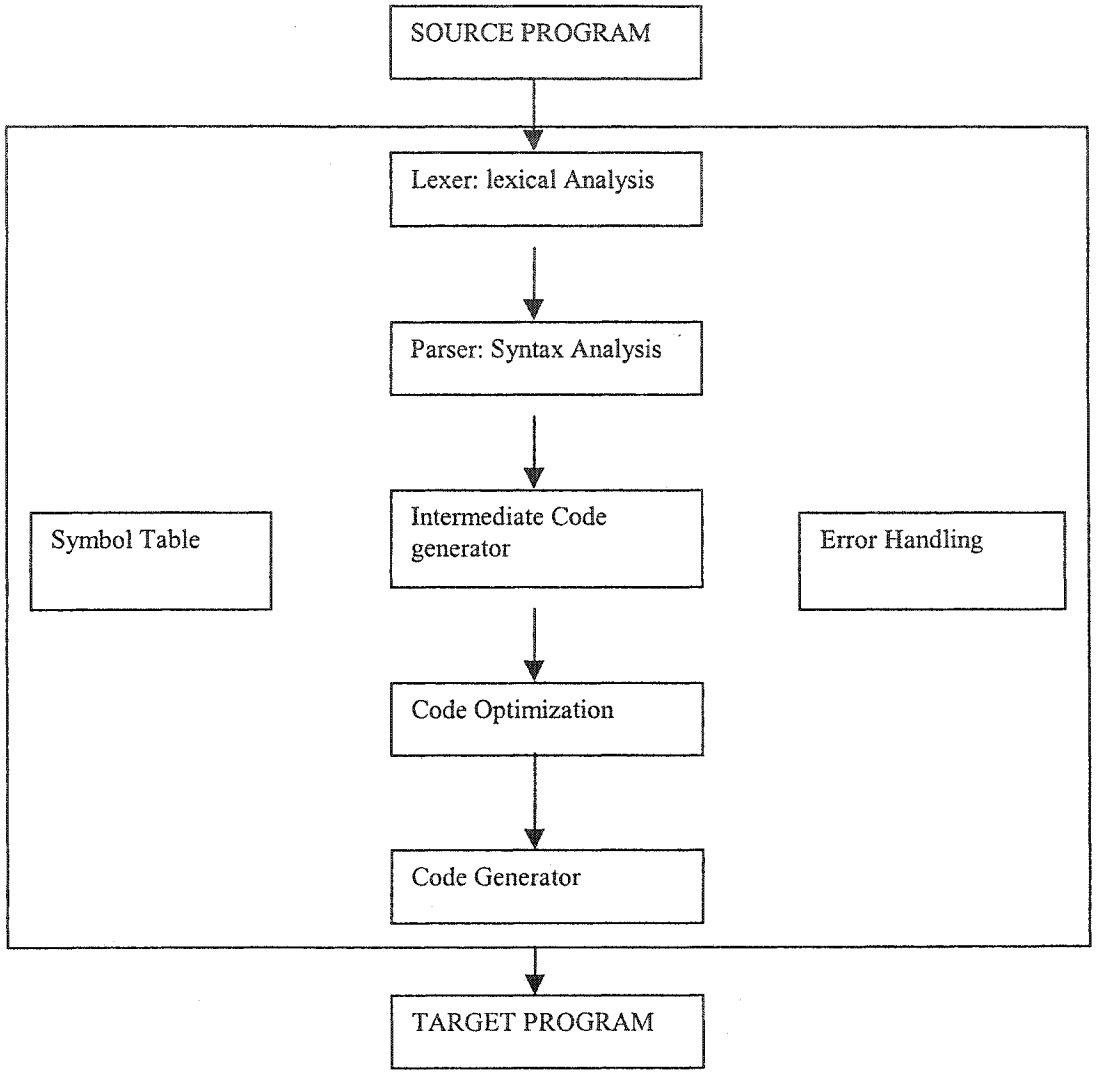


Figure 4.4 The basic structure of a compiler

As shown in Figure 4.4, a compiler should have five components Lexer, Parser, Intermediate code generator, Code optimization, and Code generator. It may have a symbol table and error handling components.

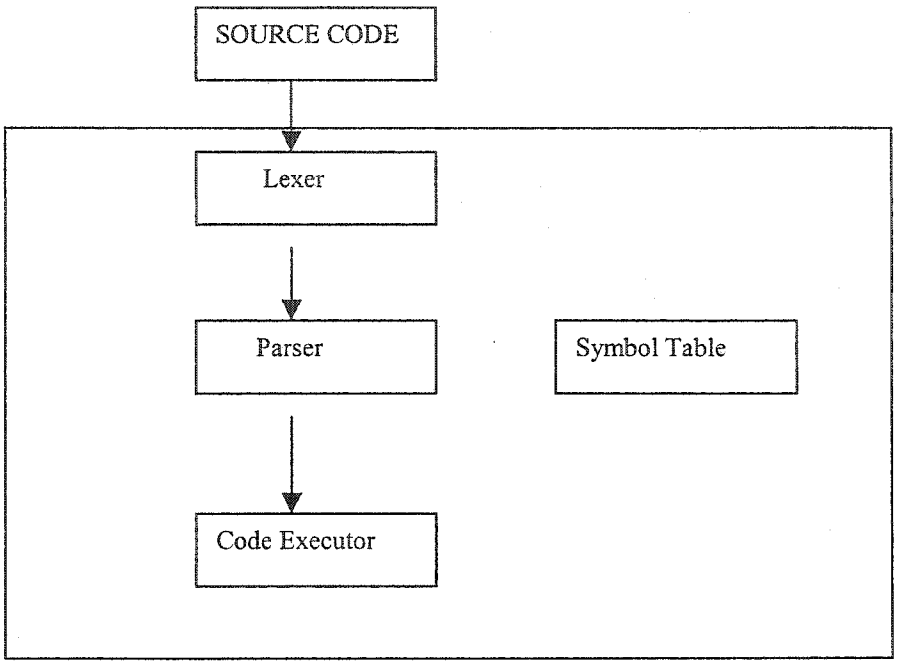


Figure 4.5 The structure of the policy interpreter

The policy interpreter is much simpler than a compiler (comparing with Figure 4.4), but it still needs a lexer, a parser, a code executor, and a symbol table.

As far as SNMP concerned, we need a SNMP implementation which is modular and supports SNMPv1 to v3. If the SNMP implementation is modular, then it should be very easy to extend with new MIB and easy to add SNMPCONF as a new modular. If the implementation supports SNMPv1 to v3, it can be deployed in any SNMP-supported network. We need a SNMP implementation which supports all versions of SNMP so that our framework can communicate with all devices.

Chapter 5

Implementation of the Policy-based Network Configuration Framework

In this chapter, the solution presented in Chapter 4 will be implemented as a framework. The implementation will use JAVA and Modular SNMP (Cherkaoui et al., 1998). In the following, the details of each part will be presented

5.1 Policy Interpreter

Policy interpreter is the core of the policy framework which will take policy code from policy table and execute it. As in the present study, because of popularity of JAVA and implementation of Modular SNMP, the interpreter is programmed in Java. There are many java parser tools available, to name a few, JavaCup, a YACC type implementation in JAVA by Scott Hudson (2001) of Georgia Institute of Technology; ANTL (Another Tool for Language Recognition), which functions like YACC, but is significantly more powerful, flexible, and easy to understand and is designed by Parr (2001); JavaCC, a java parser generator from Sun Microsystem (2001).

JavaCC is written in Java and it has been run on a variety of Java platforms. Moreover, JavaCC comes with a lot of grammars including Java 1.0.2, Java 1.1, and Java 2 as well as a couple of HTML grammars. It has the following characteristics (Sun Microsystem, 2001):

1) "JavaCC generates top-down (recursive descent) parsers as opposed to bottom-up parsers generated by YACC like tools. This allows the use of more general

grammars. Besides more general grammars, top-down parsers have other advantages such as being easier to debug, the ability to parse to any non-terminal in the grammar, and the ability to pass values (attributes) both up and down the parse tree during parsing. This is very useful for us to design and implement a policy interpreter. Further more, JavaCC comes with a tool, JJTree, which is a very powerful tree building preprocessor.

2) "By default, JavaCC generates an LL(1) parser. However, there may be portions of the grammar that are not LL(1). JavaCC offers the capabilities of syntactic and semantic lookahead to resolve shift-shift ambiguities locally at these points. i.e., The parser is LL(k) only at such points, but remains LL(1) everywhere else for better performance. Shift-reduce and reduce-reduce conflicts are not an issue for top-down parsers. JavaCC allows extended BNF specifications - such as (A)*, (A)+, etc. - within the lexical and the grammar specifications.

3) "The JavaCC release includes a wide range of examples including Java and HTML grammars. The studying of examples along with their documentation is a great way to get acquainted with JavaCC. JavaCC error reporting is among the best in parser generators. JavaCC generated parsers are able to clearly point out the location of parse errors with complete diagnostic information."

Base on the above characteristics and its popularity, JavaCC is chosen to build the policy interpreter.

The first step toward building a interpreter is to define the grammar. The policy language grammar defined by Policy MIB draft (Waldbusser, Saperia, and Hongal, 2001) was used as a basic grammar. In order to support CLI, some CLI commands are incorporated as accessor functions into the grammar. Details of the CLI command implementation will be discussed later. In this implementation, the grammar was transformed to a JJTree specification file (SPL.jjt; where SPL stands for Simple Policy Language and the file is shown in the Appendix B) according to JavaCC.

Normally, the grammar specification for the parser generator is defined using Regular Expression. Even Regular Expression is employed, there are still problems of ambiguity. For example, how to deal the famous "dangling else" problem. The problem is dealing with a statement like the following:

If A1 If A2 S2 else S1

and if the grammar for "if" statement looks like:

Void IfStm()

{

"if" C() S() ["else" S()]

}

Then, we can bound the "else" statement to either of the "If " statement. Should we parse it like:

If A1(If A2 S2 else S1)

Or

If A1(If A2 S2) else S1

In this case, JavaCC offered a "LOOKAHEAD" method and we should define the "If" statement in the following ways:

Void IfStm()

{ "if" C() S() [LOOKAHEAD (1) "else S()] }

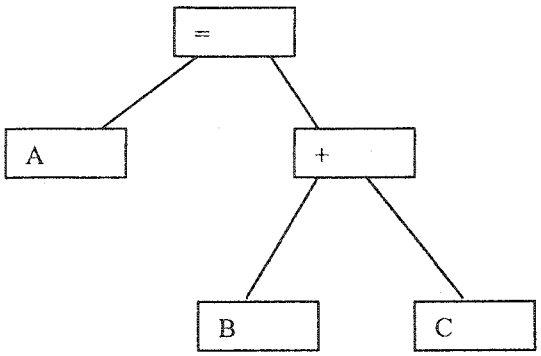
On the same ground, "LOOKAHEAD" is used in several places (See Appendix B).

After JJTree specification is done, a parser framework can be produced with the help of JavaCC and JJTree which is a tool from JavaCC and used to build a parser tree from SPL.jjt. The parser generated from above has to be transformed into a interpreter. This is done by adding one method INTERPRET with a vector as an argument:

public void interpret(Vector param)

to each node of the parsing tree. The interpreter will interpret each function with parameters stored in the vector.

In order to make the interpreter function properly, we also need a symbol table to hold variables and constants, and a stack to execute functions or expressions. The structure of interpreter is shown in Figure 5.1. The stack is used to help the interpretation of the functions. The symbol table is used to hold variable values. When the value of a variable is changed the new value will be recorded into the symbol table. For example, if A=b+c is executed, the parsing tree can be presented as the following:



First, this all nodes of simple tree are pushed into the stack. Then, B and C will be popped from the stack. If B and C are constants, their value will be taken from the symbol table. Otherwise, their value will be calculated first. After their values are obtained, (B+C) is calculated and the value is pushed back to the stack. Finally, A is popped out and the value of (B+C) is assigned to A and the symbol table is updated.

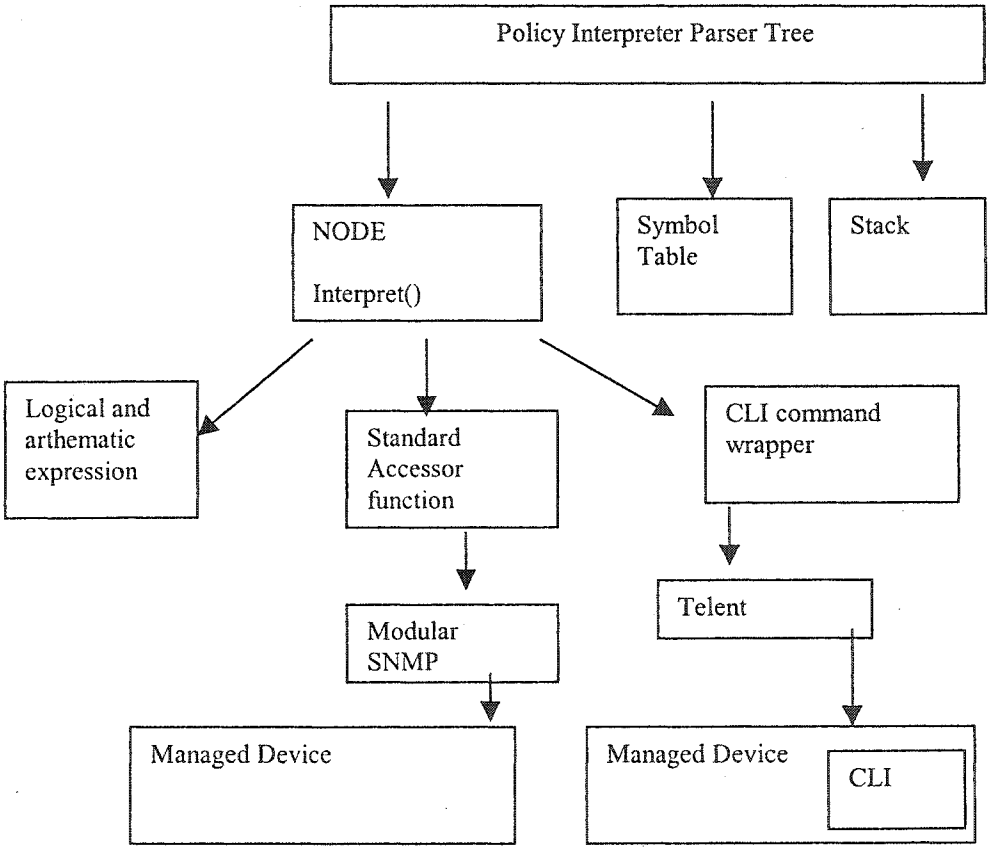


Figure 5.1 The hierarchy of the policy interpreter

5.2 Implementation of CLI Commands

Command Line Interface (CLI) is vendor-dependent. The different vendors' CLI has different syntax and hierarchy, but most of them support Telnet access. Therefore, CLI commands may be wrapped as accessor functions in policy language and executed through Telnet session.

Here CISCO IOS (Internet Operating System) (Cisco System, 1998) was taken as a example to illustrate how to wrap CLI commands in policy language. In order to execute a CLI command, a proper command mode should be entered first. There are two primary command modes and many sub-modes. The hierarchy of Cisco command modes can be presented as a tree. One has to travel through from the root to a node in order to get the desired command mode. This tree is presented as:

User Exec Mode

Privileged Exec Mode

Global configuration Mode

Interface Configuration Mode

Controller Configuration Mode

Hub Configuration Mode

Map_list Configuration Mode

Map_Class Configuration Mode

Line Configuration Mode

Router_Configuration Mode

IPX-Router Configuration Mode

Router_Map Configuration Mode

Key Chain Configuration Mode

Response Time reporter Configuration Mode

Access_list Configuration Mode

ROM Monitor Mode

A typical configuration process using CLI from telnet session can be described as the following steps:

1) Traversal to the desired command mode.

   For example, if we want to define an access-list, we may do so in the global configuration mode. However, if we want to change parameter of a interface, we need to enter privilege mode first.

2) Execute the command with necessary parameters

3) Press CTRL-Z to exit from the present mode

4) Use SHOW command to check if the new configuration takes effect.

5) If the new configuration is desired, we save the configuration to non-flash memory.

Therefore, in our implementation, the CLI commands will be wrapped as accessor functions. Each function is presented as a class which implements the abstract class NODE in the interpreter and the details of the four steps are coded inside the function Interprt(). The parameters will be passed as the functions arguments.

Access Lists in Cisco IOS is used to control traffic over routers and networks. With Access List, administrators can implement access control policies or traffic routing

policies. Taking configuring Access List as an example to illustrate how to wrap CLI commands into an accessor function, we have to do the followings:

a. Connect to the router by telnet
b. Change to global configuration mode
c. Enter command :

   access-list [access-list-number] [permit|deny] [ip][ipmask]

d. Show access-list to confirm if the access-list configured.

These steps can be wrapped as one function as

Bool Accesslist (deviceIP, AccessParameters)

{

Telenet connect the device with deviceIP

Enter the commands with accessParameters supplied

Using show command to confirm if the configuration set

If yes return true

Else return false

}

Where the device IP specifying the IP of the device (router) and AccessParameters supplying all the parameters that needs to configure an access list such as aceeslist number, permit or deny, IP address, ip mask.

Because the Access List is numbered, and the number of the Access-list tells the format it should have. For a standard Access-list, it should use any number between 1-99. Extended Access-list should be denoted any a number in the range from 100-199. In order to make this function more intelligent, we may let the function found the access list

number used and what is the next number available. So the function can be changed into the following:

Bool Accesslist (deviceIP, AccessParameters)

{ Telenet connect the device with deviceIP

Using SHOW to get information about access lists

Check the result from SHOW

and get the highest access list number using regular expressions

Enter the commands with accessParameters supplied

Using show command to confirm if the configuration set

If yes return true

Else return false

}

In the present implementation, the user has to give the access list number.

5.3 Modification of Modular SNMP

Modular SNMP from Universite du Quebec a Montreal (Cherkaoui et al., 1998) is chosen as the basic SNMP structure to implement our policy framework as discussed in Chapter 4. Although Modular SNMP is an excellent implementation of SNMP, because the Policy Based Management with SNMP is an extension of the SNMP protocol, some modifications are thus necessary in the present study to implement policy based management:

First of all, modify the Dispatcher Class so that a policy request can be handled properly.

Secondly, implement a new application model PolicyApp which will process Policy related PDU including dispatch and receiving.

Thirdly, add the minimal Policy MIB (as discussed in Chapter 3) to the Modular SNMP to handle the policy request.

The structure of the modified Modular SNMP is shown in Figure 5.2.

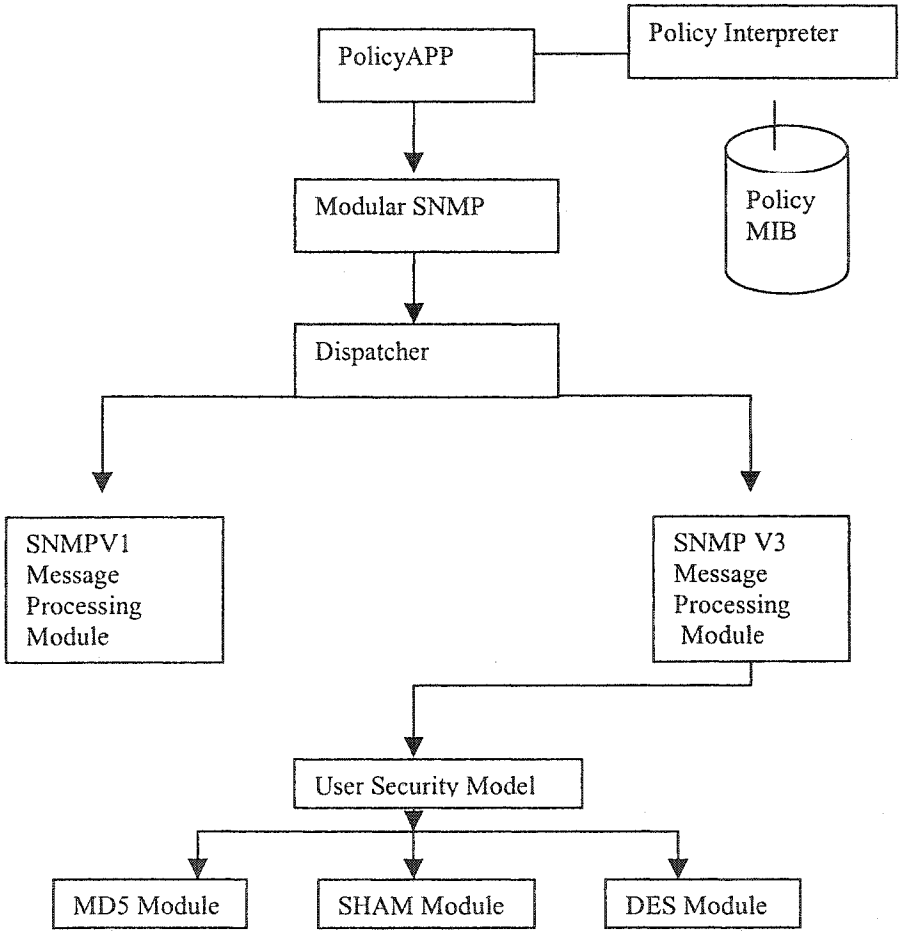


Figure 5.2 New structure of Modular SNMP framework

A new application and MIB are added to the original Modular SNMP and the corresponding changes are also made to the Dispatcher.

Chapter 6

Managing Network by the Policy Framework

In this Chapter, we will show how to use the policy-based SNMP to manage a network and compare it with the traditional CLI and SNMP approach.

6.1 A Simple Campus Network

Suppose a very simple network consists of four Cisco routers and numerous computers. Four computers are attached to router as SNMP agent and the routers are used to control the data flow between the computers and the internet. The architecture of this network is shown in Figure 6.1. The four routers control four sub-networks with different access policies. In the Library subnet, everyone is allowed to access 24 hours/per day, 7 days a week using HTTP. In the Financial Service subnet, only intranet access using HTTP is allowed within time period from 9:00am to 5:00pm Monday to Friday; the Student Service allow every student access from 8:00 am to 6:pm Monday to Friday; the Teaching LAB allows students access from 8:am to 6:00 pm Monday to Friday (see Figure 6.1 for details). All routers use Ethernet interface 0 for TCIP/IP network.

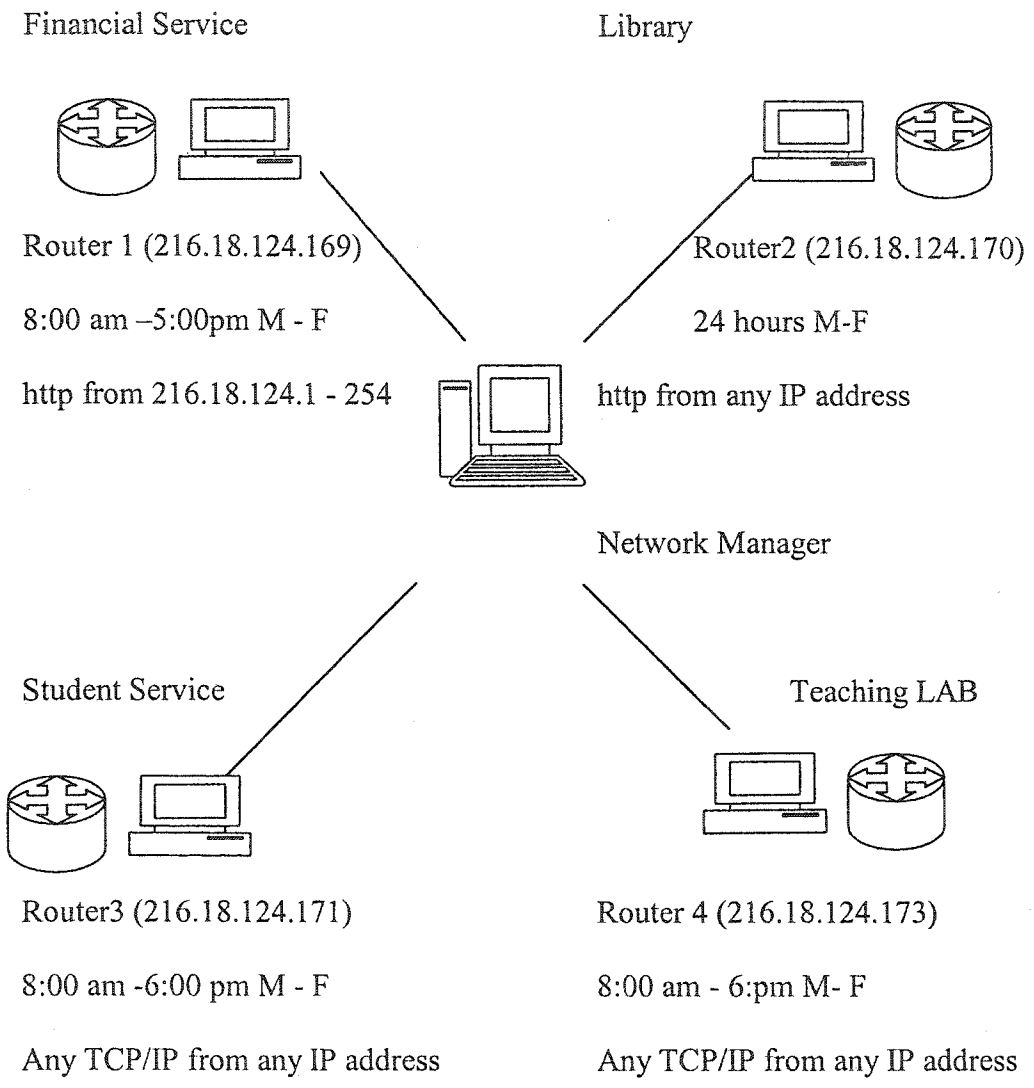


Figure 6.1 Management requirements for a campus network

In Figure 6.1, the intranet is divided into four subnets managed by four Cisco routers respectively, and each router is controlled by a computer which in turn is controlled by a central computer manager. These computers support SNMPCONF. M-F represents weekdays (Monday to Friday) (see text for details).

6.2 CLI Configuration

In this campus network, if a traditional management approach, e.g. CLI, is used to configure these routers, the administrator will have to configure the routers one by one. We will show how to use CLI to configure the network in this section. And then we will describe how to use SNMP to do the configuration next. In order to simplify our task, we assume all routers are made by Cisco and Access List of Cisco IOS can be used. Since we need to filter the data packets according to TCP/IP protocol, e.g., HTTP for Router 1 and 2, the extended access list should be used.

The Cisco IOS commands to configure Router 2 to allow http for every day, 24 hours access can be presented as the followings:

Access-list 105 permit tcp 0.0.0.0 255.255.255.255 eq 80

interface ethernet 0 ip access-group 105 in

The first command presents a numbered extended access list with a number 105. This list permits TCP/IP traffic from any IP address (IP address =0.0.0.0 with wildcard 255.255.255.255, it matches any IP address) and the port used is 80 (HTTP). Here we choose 105 as the access list number with no special meaning. It could be any integer between 100 and 200 as long as it is not presently used in the Router 2 (In the following router configuration, the access list number is chosen on the same ground without any special meaning). The last command applies this access list 105 to Ethernet interface 0.

The CLI commands to configure Router 1 can be presented using extend access list as:

time-range Time2 periodic weekdays 8:00 to 17:00

Access-list 101 permit tcp 216.18.124.0 0.0.0.255 216.18.124.169 0. 0. 0. 0 eq 80 time-range Time2

Access-list 101 deny ip 0.0.0.0 255.255.255.255 216.18.124.1 0.0.0.0

interface ethernet 0 ip access-group 101 in

The first entry defines a time frame from Monday to Friday (weekdays) and 8:00 am to 5:00 pm (8:00 to 17:00) with a name *Time2.* This name can be used in the later commands as a reference. The second command allows every http data packet (from port 80) from the intranet with IP 216.18.124.0/24 to be transported the destination 216.18.124.1 during the time defined by the time_range *Time2.* The first address/mask pair *(216.18.124.0 0.0.0.255)* means *216.18.124.any* because the 255(11111111) is a wildcard in the last byte; the second address/mask pair *(216.18.124.169 0. 0. 0. 0)* indicates the destination address 216.18.124.169. We place "eq 80" at the end of IP address/mask pairs, meaning "allow packets with the destination port 80". In the end of this command, time range *Time2* is used to activate the command in the period specified by *Time2.* The third command denies every packet from any other IP address. Every packet other than HTTP from the intranet will be denied implicitly. The last command forces this access list to be applied to the Ethernet interface 0.

For router 3, the standard access list is enough:

time-range Time3 periodic weekdays 8:00 to 18:00

Access-list 5 permit 0.0.0.0 255.255.255.255 time-range Time3

interface ethernet 0 ip access-group 5 in

This is similar to the CLI commands for Router 1 and 2, but we do not distinguish the source and destination address and specify the protocols.

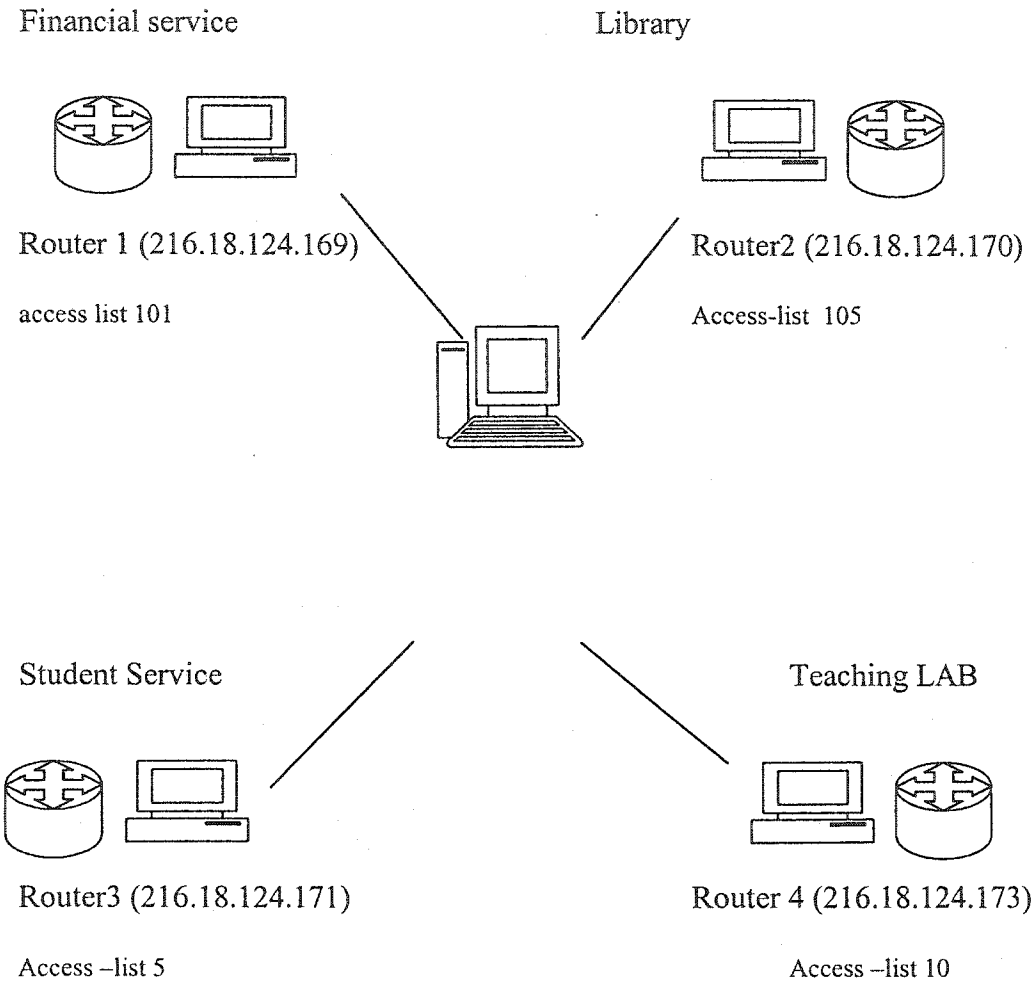


Figure 6.2 A campus network configured by CLI commands

Under this kind of management, if a new Library subnet is added to the network, the manager should configure Library 2 manually. In the best case, he/she only need copy the configure file from Router 2. See text for more detail discussions and Figure 6.1 for the requirements.

For Router 4, again, the standard access list is used:

time-range Time3 periodic weekdays 8:00 to 18:00

Access –list 10 permit 0.0.0.0 255.255.255.255 time-range Time3

interface ethernet 0 ip access-group 10 in

In order to configure these four routers, the network administrator has to login to each of the four computers attached to the routers, use telnet to access the routers, and type the commands listed in the above to complete the task. Even though the commands are similar or identical (commands for Router 4 and Router 3), the administrator has to repeat tying these commands for each router.

6.3 SNMP Approach

If SNMP is used to do the configuration, we first need to design and implement new MIBs because the standard MIB and the MIB defined by other RFCs (see Appendix A for a list of MIBs defined) cannot be used to configure the Ethernet interface with a time frame. We need at least three tables: a standard access list table or an extend access list table, an interface table, and a time range table. These tables will store parameters for standard access list or extend access list, interface, and time_range commands. For example, the standard access list table may be defined as:

```
AccessListEntry ::= SEQUENCE {
AccessListIndex          Unsigned32,
AccessListIP             UTF8String,
AccessListWildCard       UTF8String,
```

AccessListPermission INTEGER

AccessListTimeRange Unsigned32

}

The time range table should be very similar to that of Policy Schedule Table defined in the policy MIB draft (Waldbusser et al., 2001). After these tables are defined, the SNMP protocol can be used to send SNMP commands from the SNMP manager to the SNMP Agent. It is obvious that this approach is similar to that of SNMPCONF discussed in the above.

In order to illustrate the idea, we make these three MIBs tables (Table 6.1, Table 6.2 and Table 6.3) for router 3. Table 6.1 presents a private MIB for Access List and the contents for Access List 5 on router 3 in the above section. As their names imply, the user may figure out that Table 6.1 is actually a direct representation of CLI Standard Access List command. The AcessListIndex represents access list number; AccessListIP shows the source IP; the AccessListWildcard holds the wild card used in the access list; the acessListPermission indicates "permit" or "deny"; and the AcessListTimeRange points to time_range entry in the Time_range Table and the actual time_range is defined in the Time_range Table (Table 6.2). Table 6.2 is used to store information about the time_range command on router 3 and Table 6.3 is a MIB translation of the CLI interface command (the third command in the above section for Router 3). The following SNMP commands should be used to set router 3, instead of CLI commands:

SNMPSET(AccessListindex.0, 5)

SNMPSET(AccessListIP.0,0.0.0.0)

SNMPSET(AccessListWildCard.0,255.255.255.255)

SNMPSET(AccessListPermission.0, permit)

SNMPSET(AccessListTimRange.Time3)

SNMPSET(TimeRangeName.Time3)

SNMPSET(TimeRangePeriod.0,Periodic)

SNMPSET(TimeRangeRange.0,Weekday)

SNMPSET(TimeRangeStratrtTime.0,8:00)

SNMPSET(TimeRangeEndTime.0,18:00)

SNMPSET(NIFIndex.0,1)

SNMPSET(NIFName.0, Ethernet 0)

SNMPSET(NIFAccessList.0, 5)

These commands should be execute either completely or no one at all, because we cannot have a partial access list or partial time-range enforced on an interface, but this is a very difficult task.

Table 6.1 The private access list MIB for Router 3

AccessList-Index 1.3.6.1.4.1.9.2.1	AccessList-IP 1.3.6.1.4.1.9.2.2	AccessList-WildCard 1.3.6.1.4.1.9.2.3	AccessList-Permission 1.3.6.1.4.1.9.2.4	AccessList-TimeRange 1.3.6.1.4.1.9.2.5
5	0.0.0.0	255.255.255.255	Permit	Time3

Table 6.2 The private time_range MIB for Router 3

TimeRange-Index 1.3.6.1.4.1.8.2.1	TimeRage-Name 1.3.6.1.4.1.8.2.2	TimeRagne-Period 1.3.6.1.4.1.8.2.3	TimeRagne 1.3.6.1.4.1.8.2.4	TimeRange-Starttime 1.3.6.1.4.1.8.2.5	TimeRange-EndTime 1.3.6.1.4.1.8.2.6
1	Time3	Periodic	Weekdays	8:00	18:00
2	Tim10	Periodic	Weekend	12:00	14:00

Table 6.3 The private network Interface MIB for Router3

NIFIndex 1.3.6.1.4.1.7.2.1	NIFName 1.3.6.1.4.1.7.2.2	NIFAccessList 1.3.6.1.4.1.7.2.3
1	Ethernet 0	5
2	Ethernet 1	20

If a new router is added to the network, the network manager may copy some configuration file from router to router and save some work. However, if a new router is added, he /she has to repeat the task again. Given that the network grows very fast, the administrator's work will increase with the network size growth and more network administrator will be needed if the network is too big for the current administrators to

handle. In other words, the device by device configuration increases the work load for the administrator when the network grows.

6.4 SNMPCONF Approach

If policy based SNMP (SNMPCONF) is used in this network, the configuration task will be reduced to define different policies and the system will automatically perform the configuration task. When a new router is added, we may register it as the existent element type and enforce the existing policies or we register it as a new type and add new policies to manage it.

The first step to configure this campus network with policies is to define policies for it in accordance with the requirements. The policies for this campus network are described by Table 6.4, 6.5, and 6.6. In Table 6.4, four routers are registered as four different element types as represented by the prefix of their OID such as *1.3.6.1.4.1.1.2.1*. These OIDs are not defined by the standard MIB, but under the private enterprise node in the MIB tree (see Chapter 2) and we suppose that these OIDs are not used by the network device vendors (this may not be true, but it should not be of any harm for the illustration purpose). Three policies are registered with index from 1 to 3 under policy group "Access Control". The policies contents are shown under "pmPolicyDescription" and the policy scripts as indexes which points to Table 6.5 under pmPolicyConditionScriptIndex and pmPolicyActionScriptIndex for policy action and filter respectively. The policy script codes are stored in Table 6.5 and linked to the Table 6.4 through the indexes (pmPolicyCodeScriptIndex in Table 6.5 equals pmPolicyConditionScriptIndex or pmPolicyActionScriptIndex in Table 6.5. The policy schedules for these policies are

stored in the Policy Schedule Table (Table 6.6) and linked to the policies in Table 6.4 by the index (pmSchedIndex in Table 6.6 equals pmPolicySchedule in Table 6.4). In Table 6.6 a time period for a specified policy to be executed is set by changing the value of the bits at the desired position. For example, seven bits are used to represent the weekdays from Sunday to Saturday. Because Policy #1 is scheduled to run from Monday to Friday, then the bits to represent this schedule can be set as 0111110. The first bit represents Sunday and the last bit represents Saturday. Both of tem are set as 0 because the policy is not active on Sunday and Saturday. The second to sixth bit are set as 1 for they represent weekday from Monday to Friday and the policy is active on Monday to Friday.

Table 6.4 The policy table for the campus building management

pmPolicyIndex ,	1	2	3
pmPolicyGroup	Access control	Access control	Access control
pmPolicyPrecedence	0	1	1
pmPolicySchedule	1	2	3
PmPolicyElementTypeFilter	1.3.6.1.4.1.1.2.1	1.3.6.1.4.1.1.3.1	1.3.6.1.4.1.1.1.2
pmPolicyConditionScriptIndex	1	3	5
pmPolicyActionScriptIndex ,	2	4	6
pmPolicyDescription	Http access is permit for all intranet from 8:00am to 5:00 pm Monday to Friday	Http access is permit for everyone 24 hours everyday seven days a week	access is permit for every one from 8:00 am to 6:00 pm Monday to Friday.

Table 6.5 The policy code table for the campus building management

PmPolicy-CodeScriptIndex	PmPolicy-CodeSegment	pmPolicyCodeText (1.3.6.1.3.107.2.2)	pmPolicy-CodeStatus
1	1	V1=SNMPGET(1.3.6.1.4.1.1.2.1) If v1!= null then Return true else return false	
2	1	Extended_access_list (permi tcp 216.18.124.1 0.0.0.2555. 216.18.124.1 0. 0.0. 0 eq 80)	
3	1	V1=SNMPGET(1.3.6.1.4.1.1.3.1) If v1!= null then Return true else return false	
4	1	Extended_access_list (permit permit tcp 0.0.0.0 255.255.255.255 eq 80	
5	1	V1=SNMPGET(1.3.6.1.4.1.1.1.2) If v1!= null then Return true else return false	
6	1	Permit 0.0.0.0 255.255.255.255	

Table 6.6 The policy schedule table for the campus building management

PmSchedIndex	1	2	3	4
PmSchedGroupIndex	1	1	1	1
pmSchedDescr	8:00am to 5:00 pm Monday to Friday	24 hours everyday	8:00am to 6:00pm Monday to Friday	8:00am to 6:00 pm Monday to Friday
pmSchedTimePeriod				
PmSchedMonth				
PmSchedDay				
PmSchedWeekDay (1.3.6.1.3.107.8.5)	0111110	1111111	0111110	1111111
PmSchedTimeOfDay (1.3.6.1.3.107.8.8)	000000001111 111111000000	111111111111 111111111111	000000011111 11111100000	000000011111 111111111111
PmSchedLocalOrUtc				
PmSchedStorageType				
PmSchedRowStatus				

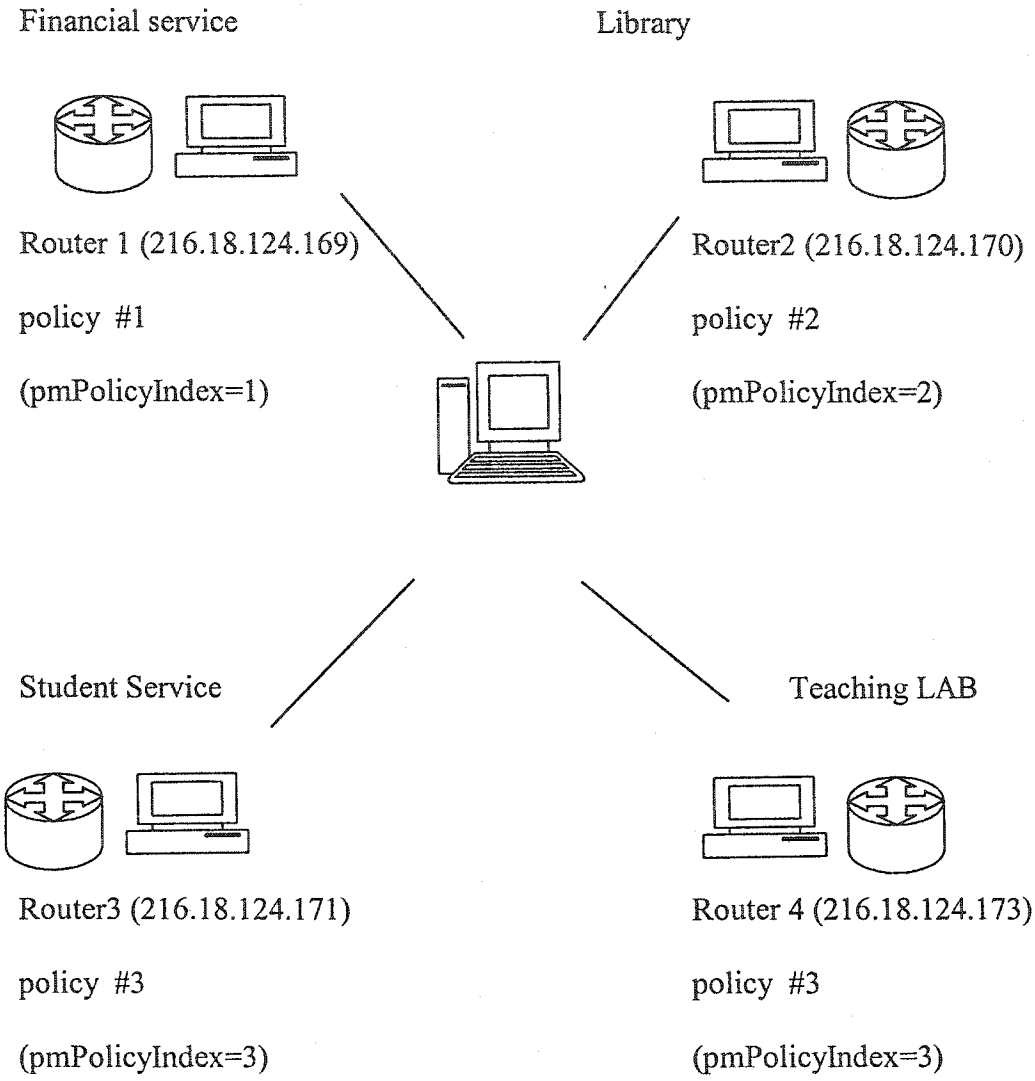


Figure 6.3 A campus network managed by policy-based SNMP (SNMPCONF)

Under policy-based SNMP management, the configuration is done through policies enforcement. If a new Library (Library 2) is added to the network, the manager just needs to register this element and automatically deploys the policies related. This

simplified the network management. See text for more detail discussions, Table 6.4 and Table 6.5 for policies, and compare this approach with that of Figure 5.4.

6.5 Present Framework

The present implementation can be set up to do the policy-based configuration as shown in Figure 6.6. In Figure 6.6, Student Service router is taken as an example. Because the policy MIB is not fully implemented at present, especially the policy schedule table, we need to divide the simple policy #3 into two sub-policies for the schedule convenient. Policy #3 can be decomposed as:

1) Permissions will grant to everyone from 8:00am to 6:00pm Monday to Friday
2) Accesses will be denied from 6:pm to 8:am next day on Monday to Friday and the whole day on Saturday and Sunday.

Because Access list command is implemented as accessor function in the present work, we need to use accesor functions to represent the above sub policies:

1) Monday to Friday 8:00 am, execute

    noaccess list 5

    stdaccesslist 5 permit any

The first line code will remove the existing access list 5 and second line adds a new access list 5 which allow every one access. After executing these two line codes, the system now allows every one to access. Therefore, every one will get access until we change access list 5. Because this policy is valid until 6:00pm, we have to enforce the following code.

2) Monday to Friday 6:00pm, execute

noaccess list 5

stdaccesslist 5 deny any

After executing these two lines of codes, the system will deny anyone to access. The combination of the above two sub policies (or four lines of codes) actually enforces policy #3.

The user may set up the policies through the SNMP manger repeating the following SNMP command five times:

{

SnmpSet(ip address, 1.3.6.1.3.107.8.5, weekday)

SnmpSet(ip address, 1.3.6.1.3.107.8.8, 8)

SnmpSet(ip address 1.3.6.1.3.107.2.2, policy code1)

SnmpSet(ip address, 1.3.6.1.3.107.8.5, weekday)

SnmpSet(ip address, 1.3.6.1.3.107.8.8, 20)

SnmpSet(ip address 1.3.6.1.3.107.2.2, policy code2)

}

policy code 1= { no accesslist 5;stdaccesslist 5 permit 0.0.0.0 255.255.255.255}

policy code 2= { no accesslist 5;stdaccesslist 5 deny 0.0.0.0 255.255.255.255}

where the weekday=Monday to Friday. IP address in the present example is 216.18.124.171 (see Figure 6.4). The OID 1.3.6.1.3.107.8.5 and 1.3.6.1.3.107.8.8 are defined in the policy MIB for weekdays and time of the day; the OID 1.3.6.1.3.107.2.2 is used for the policy code. Because in the present implementation, a Java thread is used to

schedule the policy, we have to set it up for every weekday, this why we need to send this policy code five times. After SNMP commands executed, the SNMP agent will execute the policy accordingly on the router it managing as scheduled.

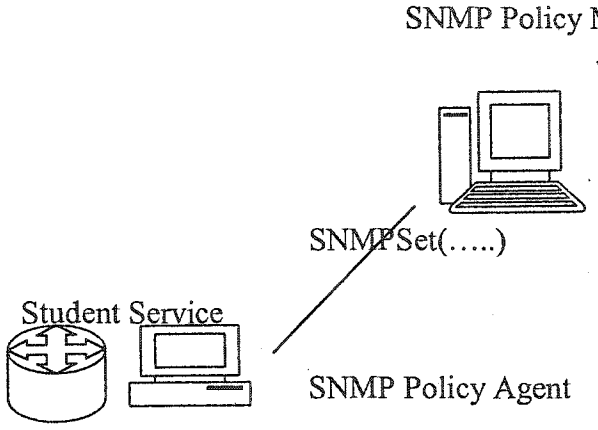


Figure 6.4 Manage student service sub-network using policy based SNMP

The policies are sent from the SNMP Policy Manger to the SNMP Policy Agent and The SNMP Policy Agent will execute the Policy according to the schedule.

In the above, a simple campus network with four routers was used as an example to show how to configure network using three different approaches, CLI, SNMP, and SNMPCONF. It is evident that CLI commands are easy to use when single device is concerned. SNMP is not good to configure network devices, because new MIBs have to be designed and too many parameters needed to do a simple configuration. Another disadvantage of the SNMP approach is that atomicity of the configuration can not be

guaranteed, because so many MIB object have be set in order to perform a very simple configuration. SNMPCONF is so far the best protocol to configure the network as whole and easy to be adapted as new device added.

Chapter 7

Discussions

In Chapter 4 and 5, a solution is proposed to implement SNMPCONF in the existed network where no configuration MIB is supported. Chapter 6 compares the CLI, SNMP, and SNMPCONF approach to configure a very simple network. In this chapter, the advantage and disadvantage of the present implementation will be discussed.

7.1 Advantages and drawbacks of the present solution

It is evident that from the description in the above, the present implementation overcomes the obstacle of lack MIBs using CLI commands. In the transient stage from SNMPCONF-unsupported to SNMPCONF implemented, the present approach works in the existed network and will be useful for future applications. Because even SNMPCONF is fully implemented in the new network device, the support from the old device is still need in order to deploying SNMPCONF. The present study presents a solution to this kind of problems.

After examining the definition, deployment, and management of policies related to QoS (Quality of Service) in an IP network, Rajan et al. (1999) found that it would be difficult to exchange information between two neighboring administrative domains. They suggested a service level agreement should be specified before hand. The same problem may exist in the policy configuration. If there are two different administrative

domains in a network, one deploys SNMPCONF, but the other one does not even support SNMP (e.g., telecommunication network). How can them exchange information? The possible answer is using CLI or TL1 as presented in this work.

Another advantage of this approach is that it makes the policy execution atomic and persistent easily. In other words, the policy execution in the present framework can be rollback as a transaction.

As defined, a transaction is a collection of actions conformed with ACID properties (Orfali et al., 1998). ACID stands for Atomicity, Consistency, Isolation, and Durability. Atomicity means that a transaction is an indivisible unit of work: All of its parts succeed or they all fail. Consistency means that after a transaction executes, the system should be in a correct state or the transaction should be aborted. Isolation accounts that a transaction's behavior is not affected by the other transaction that execute concurrently. Durability represents that a transaction's effect are permanent after it commits. Its changes should survive system failure.

If we use CLI command to do policy configuration for Cisco routers, the policy execution could be transactional. In Cisco routers, all running configuration is hold in the RAM, which is erased if the device loses the power. The configuration change through CLI commands is placed in the RAM, too. If we want to save these changes we made, we have to copy them to the Non-Volatile RAM or NVRAM. Otherwise, the changes will be lost if the devices reboot or lose power. This characteristic make our configuration be of transactional. If we figure out that for some reasons the policy script is partially executed through the "SHOW" command, we may reboot the device or re-execute the script from the beginning. On the other hand, if we are sure that the configuration change

is correct and completed, we may copy the present configuration to the NVRAM and make the change take effect permanently. This shows that the accessor function wrapped from CLI commands has atomicity and durability. However, in the present implementation, the property consistency and isolation are not guaranteed.

The third advantage is that the present implementation can use many vendor-specific characteristics. In CISCO router, standard access list can be used to filter IP traffic by IP address, but extend access list can be more specific, it can be used to filter traffic by their protocols

The limitation of the present implementation is that the policy conflict is not checked. Policies are stored in the policy MIB as scripts and the present framework has no way to "understand" the meaning of the policies or the consequences of executing the policies. If policy A contradicts to the policy B, the policy interpreter will execute both and is not aware of their conflicts. Therefore, the network manager should avoid the policy conflicts before entering the policies into the system.

The second shortage of the present implementation is that the implementation does not cover all CLI commands and TL1 is not implemented because of its unavailability.

7.2 Comparison with the MIB Approach

The framework presented in the thesis uses CLI commands instead of MIB, which does not conform to the SNMPCONF. As discussed before, SNMPCONF uses SNMP

plus new MIB. Because the policy MIB is a standard to all vendors, policy defined by the policy MIB can be executed across all networks. In contrast, if the present approach is used, special attentions should be focused on the different devices from different vendors, even device from the same vendor with different version of CLI. This is the biggest problem as compared with the standard SNMPCONF. However, the present framework is intended to present a solution to the problem that no configuration MIB is available rather than replacing SNMPCONF. This problem with configuration MIB occurs only during the transition stage from present to the whole support of SNMPCONF or the devices which can not be upgrade to support SNMPCONF.

Another problem with the present implementation is how to build policy in different abstraction levels according to the different requirements. According to the drafts, four levels of policy abstraction are described from the lowest level of abstraction to the highest, Instance-specific, Mechanism-specific, Implementation-Specific, and Domain specific. In order to build policy abstraction in the different level, one approach is to use MIB as shown by the house light example (MacFaden, Saperia, and Tackabury, 2001). In the lower level, more device-specific MIB that presents more details about the device need to be defined. In the higher level, the common MIB or general purpose MIB might be used. In the house light example (Figure 7.1), the objective is to control the light and temperature of a large building. In the implementation-specific level, a 'standard' house lighting-MIB module is defined in which every parameter needed to control the heating and air conditioning are presented as an MIB object. This module contains both configuration and counter objects that allow operators to see how much cooling or heating a particular configuration has consumed. At a mechanism specific

level, a policy table that represents the configuration information is also included. This table in combination with the policy MIB Module will allow operators to configure many rooms all at once, change the configuration parameters based on time of day, and make a number of other sophisticated decisions based on policy. Therefore, using MIBs can handle problems with policy abstraction, but the present solution cannot handle the policy abstraction. In the high or low level, the accesor function wrapped from CLI are the same.

7.3 The Other Models

Some models other than SNMPCONF have been suggested to perform policy-based network management (Mahon et al., 1999; Rajan et al., 1999) in recent years. It would be interesting to compare these models with the present solution, we will compare these related to SNMP with SNMPCONF.

Mahon et al. (1999) have discussed the some possible policy based approaches with different protocols such as LDAP, SNMP, and HTTP. Because SNMPCONF is the object of the present study, we only discuss the architecture of LADP with SNMP (Figure 7.2).

In the architecture of LDAP, SNMP could be used as a transport method to provide the notification using a 'Set' operation on a MIB, with information about what policy is to be used for each Policy Target for which the Policy Consumer is responsible. The Policy Consumer in return could send the status information back to the Policy

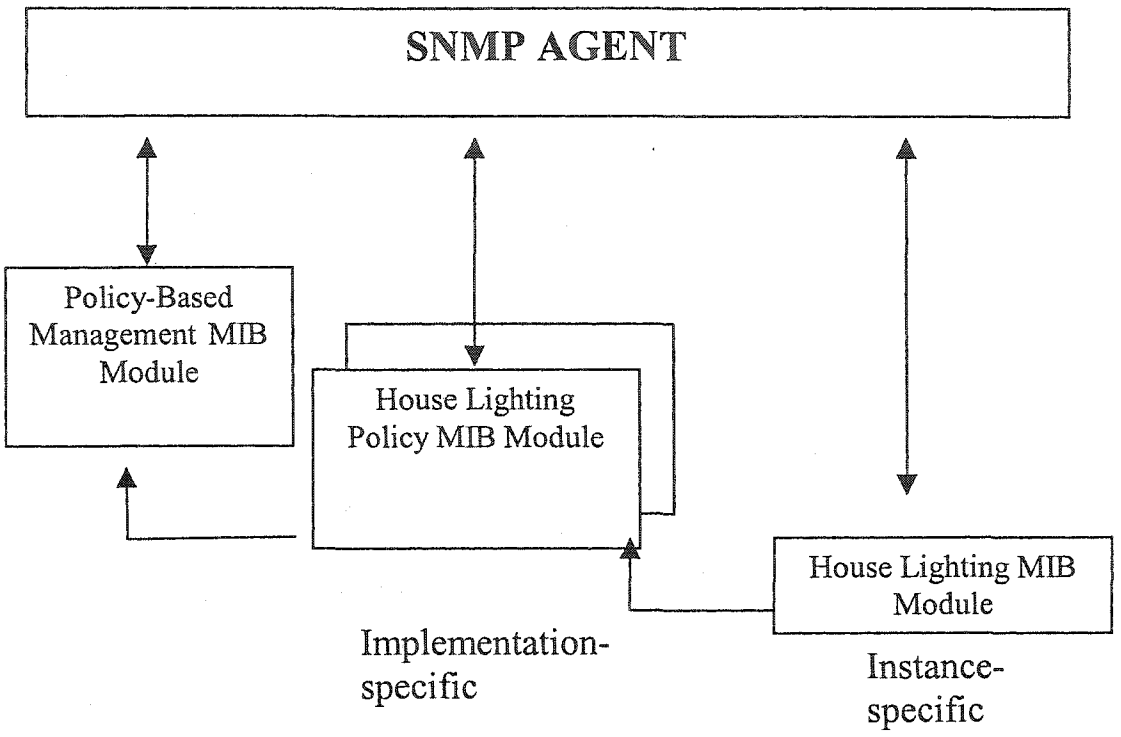


Figure 7.1 The house light control example

This example shows that with the abstraction level decreased from mechanism-specific to implementation-specific, instance-specific, them MIB modules increased from policy MIB to House lighting Policy MIB and House Lighting MIB Module (from MacFaden, Saperia, and Tackabury, 2001)

Management Application at any time using SNMP traps. In this case, SNMP is better to be built on TCP than UDP, because of reliability offered by TCP.

If SNMP could be used on all of the data paths in Policy-Based Management (Figure 7.2) not only notifying the Policy Consumers of new Policy Data, but delivering Policy Data as well, this will eliminate the need for the Policy Consumer to query the Policy Repository.

Figure 7.2 shows how an all SNMP solution would look while still using an LDAP enabled directory as an export mechanism to other Policy Management Applications. Undetermined is any mechanism (or requirement) for notification to other Management Applications of new policy.

This solution is similar to SNMPCONF, but use LDAP as a policy repository and SNMP as a transport protocol. LDAP is a protocol and stands for the Lightweight Directory Access Protocol. Lightweight, as opposed to its ancestor, the X.500 DAP protocol (Wilcox, 1999). LDAP is basically a specialized database. Some of the characteristics are:

1. It consists of entries organized in a hierarchy.
2. It favors reading over writing.
3. Every entry has a primary key called the Distinguished Name (DN).
4. Its notion of schema is much more flexible than that of a RDBMS.

A typical use of an LDAP directory is to store information about entities like people, offices, machines and that sort. But one could equally well store most other relatively static information in there, e.g. information about books or movies or cars, or anything can describe by a set of attributes.

The obvious problem with this architecture is that LDAP is good at storing static information, but not good to updating dynamic information. As we know, policy, on the

other hand, is dynamic and should be updated very often. Another problem is using SNMP. One advantage of SNMP is its MIB. If MIB is not used, then SNMP is not necessarily to be employed in this architecture. Besides, SNMP is not efficient to transport information over network. It seems that this LDAP with SNMP solution is using the shortage of each protocol.

In the above, the advantage of the framework present here is that it offers a way for SNMPCONF to communicate with SNMPCONF unsupported network or devices and many vendor-specific characteristics can be efficiently utilized.

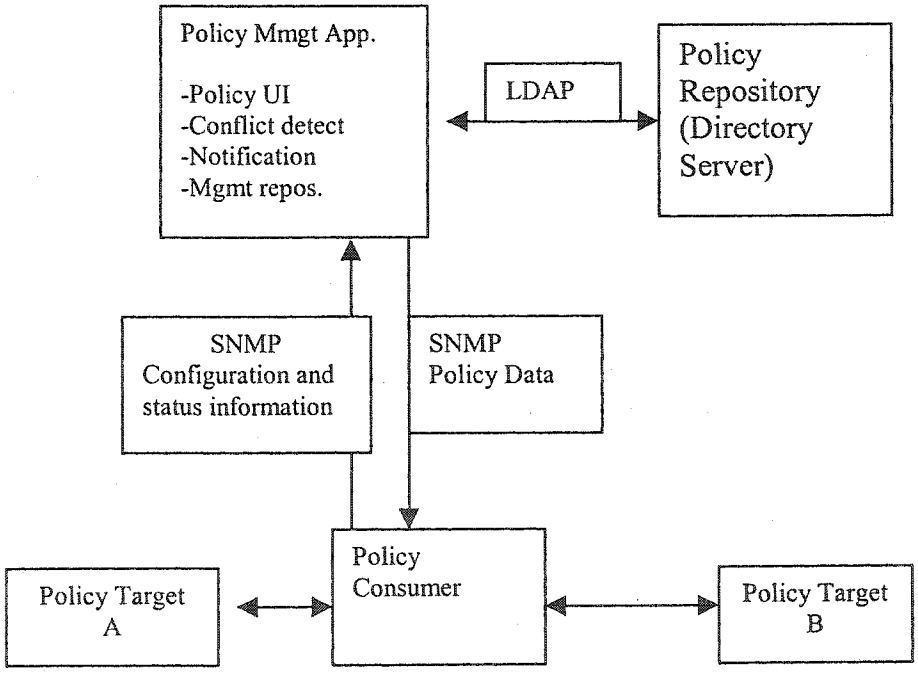


Figure 7.2. Network management with SNMP and LDAP and Policy

(modified from Mahon, Bernet and Herzog, 1999)

Chapter 8

Conclusions

In this work, we proposed a Java framework to solve the problem how to communicate between a SNMPCONF network and a SNMPCONF-unsupported network or devices and implemented this framework with the help of Modular SNMP. Some conclusions can be drawn:

1) CLI and TL1 can be used to help deploying SNMPCONF in an existing network where no configuration MIB available.
2) A Java framework, which consists of a policy interpreter that supports the policy language and functions wrapped from CLI commands, and Modular SNMP which is modified to support agent, is built to deploying SNMPCONF.
3) The modification of Modula SNMP is done through adding new PolicyMP application and policy MIB to process policy messages.
4) Present study shows that policy configuration with CLI can take advantage of special characteristics of the device, because the CLI commands best represent the characteristics of vendors. The disadvantage is that special attention should be placed on the different format and syntax of similar CLI commands by different vendors and different versions of the CLI from the same vendor.
5) This study presents a possible solution to exchange management information between a telecommunication network where TL1 is used and an enterprise network in which SNMP and CLI are extensively deployed.

References

Aho A.V. and ULLman, J.D., 1974 Principle of Compiler Design. Addison-Wesley Reading, M.A.. 644pp.

Aho A.V., Sethi, R., and ULLman, J.D., 1986 Compilers: Principles, Techniques, and Tools. Addison-Wesley Reading, M.A.. 866pp.

Appel, A.W., 1997, Modern Compiler Implementation in Java: Basic Technique. Cambridge University Press. 398 pp.

Blumenthal, U., and B. Wijnen, 1999, User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), RFC 2574.

Case, J., Fedor, M., Schoffstall, M. and Davin, J., 1990, A Simple Network Management Protocol (SNMP). RFC 1157.

Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, 1996a, Introduction to Community-based SNMPv2. RFC 1901.

Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, 1996b, Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1906.

Case, J., Harrington D., Presuhn R., and Wijnen B., 1999, Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). RFC 2572.

Cherkaoui, Omar, Hillaire, Y. S., Mili, H., Serouchi, A., 1998, Towards a Modular and interoperable SNMPv3. IEEE, 391-394.

Cisco System, 1998, Cisco IOS configuration Fundamentals. Cisco Press. 1448pp.

Erfani, Shervin, Lawrence, V. B., Malek, M., and Sugla B., 1999, Network Management: Emerging Trends and Challenges. Bell Labs Technical Journal v.4, No.4, 3-22.

Feit, Sidnie M. 1994, SNMP: A Guide to Network Management. McGraw Hill. 674pp.

Hazewinkel, H., and Partain, D., 2001, The DiffServ Policy MIB. Draft-ietf-snmpconf-diffpolicy-04.txt. http:://www.ietf.org

Hossain, Ashfaq, Shu, H.F., Gasman, C. R., and Royer, R. A., 1999, Policy-based Network Load Management. Bell Labs Technical Journal V.4, No.4, 75-94.

Hudson, Scott, 2001, CUP User's Manual.
http://www.cs.princeton.edu/~appel/modern/java /CUP/manual.html

Lumos, 2001, TL1 Beginne s Guide. http://www.tl1.com/library/TL1/Overview/.

Lupu, E. and Sloman, M., 1997, Conflict Analysis for Management Policies. Fifth IFIP/IEEE International Symposium on Integrated Network Management IM'97, San-Diego, May 1997, Chapman & Hall Publishers, pp 430-443.

Lupu, E. and Sloman, M., 1999, Conflicts in Policy-based Distributed Systems Management. IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management, Vol 25, No. 6, Nov. 1999, pp.852-869

MacFaden, M., Saperia J., and Tackabury, W., 2001, Configuring Networks and Devices With SNMP. Draft-ietf-snmpconf-bcp-05.txt. http://www.ietf.org.

Mahon, H., Bernet, Y., Herzog, S., 1999, Requirements For A Policy Management System. Draft-ietf-plolicy-req-01.txt, http://www.itef.org/.

Mak R., 1996. Writing Compilers and Interpreters. (2nd edition) John Wiley & Sons Inc. 8383pp.

Marriott, D. and Sloman, M., 1996, Management Policy Service for Distributed Systems. Proc. IEEE Third International Workshop on Service in Distributed and Networked Environments (SDNE 96), Macau, 2-9.

McCloghrie, K. and Rose, M., 1991, Management Information Base for Network Management of TCP/IP-based Internets: MIB-II. RFC 1213.

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, 1999a, Structure of Management Information Version 2 (SMIv2). STD 58, RFC 2578.

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, 1999b, Textual Conventions for SMIv2. STD 58, RFC 2579.

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, 1999c, Conformance Statements for SMIv2. STD 58, RFC 2580.

Mellquist, Peter Erik, 1997, SNMP: An Object-Oriented Approach To Developing Network Management Applications. Prentice Hall. 239p.

Moffett, J.D. and Lupu, E.C., 1999, The Uses of Hierarchies in Access Control. Fourth ACM Role Based Access Control (RBAC) Workshop, Fairfax, Virginia, USA, Oct. 28-29, 1999.

Orfali R., Harkey D., and Edwards J., 1994, Essential Client/Server Survival Guide. John Wiley & Sons Inc. 527pp.

Parr T.J. , 2001, What's An ANTLR? http://www.antlr.org /pccts133.html.

Rajan, R., Verma, D., Kamat, S., Felstaine, E., Herzog, E., 1999, A Policy Framework for Integrated and Differentiated Services In The Internet. IEEE Network Sept./Oct., 36-41.

Rose, M. and McCloghrie, K., 1990, Structure and Identification of Management Information for TCP/IP-based Internets. RFC 1155

Simuneau, Paul, 1999, SNMP Network Management. McGraw-Hill. 477pp.

Sloman, M. and Lupu, E., 1999,Policy Specification for Programmable Networks. Extended version of paper in Proceedings of First International Working Conference on Active Networks (IWAN'99). Berlin, June 1999, ed. S. Covaci, published by Springer Verlag Lecture Notes in Computer Science

Stallings, William, 1996 SNMP, SNMPv2, and RMON: Practical Network Management. 2nd Edition, Addison-Wesley Pub Co. 478pp.

Stevens M. L. and Weiss Walter J., 1999a, Policy-Based Management for IP Network. Bell Labs Technical Journal v4, No.4, 75-94.

Stevens M. L., Weiss Walter J., Mahon, H., Moore, B., Strassner, J., Waters, G., Westerinen, A., and Wheeler, J., 1999b, Policy Framework. Draft-ietf-policy-framework-00.txt, http://www.itef.org.

Stevens, W. Richard, 1990, UNIX Network Programming. Prentice Hall, Englewood Cliffs, N.J, 772pp.

Sun Microsystem, 2001, JavaCC - The Java Parser Generator. http://www.webgain.com/products/metamata/java_doc.html.

Waldbusser S., Saperia, J., and Hongal T., 2001, Policy Based Management MIB. Draft-ietf-snmpconf-pm-06.txt, http://www.itef.org.

Wilcox, Mark, 1999, Implementing LDAP. Wrox Press Ltd., 493pp.

Appendix A RFCs which defined MIB

Number	Title	Author or Ed.	Date	More Info (Obs&Upd)
RFC3144	*Remote Monitoring MIB Extensions for Interface Parameters Monitoring*	D. Romascanu	August 2001	
RFC3014	*Notification Log MIB*	R. Kavasseri	November 2000	
RFC2982	*Distributed Management Expression MIB*	R. Kavasseri (Editor of this version)	October 2000	
RFC2981	*Event MIB*	R. Kavasseri (Editor of this version)	October 2000	
RFC2934	*Protocol Independent Multicast MIB for IPv4*	K. McCloghrie, D. Farinacci, D. Thaler, B. Fenner	October 2000	
RFC2933	*Internet Group Management Protocol MIB*	K. McCloghrie, D. Farinacci, D. Thaler	October 2000	
RFC2932	*IPv4 Multicast Routing MIB*	K. McCloghrie, D. Farinacci, D. Thaler	October 2000	
RFC2922	*Physical Topology MIB*	A. Bierman, K. Jones	September 2000	
RFC2896	*Remote Network Monitoring MIB Protocol Identifier Macros*	A. Bierman, C. Bucci, R. Iddon	August 2000	
RFC2895	*Remote Network Monitoring MIB Protocol Identifier Reference*	A. Bierman, C. Bucci, R. Iddon	August 2000	Obsoletes RFC2074
RFC2864	*The Inverted Stack Table Extension to the Interfaces Group MIB*	K. McCloghrie, G. Hanson	June 2000	
RFC2863	*The Interfaces Group MIB*	K. McCloghrie, F. Kastenholz	June 2000	Obsoletes RFC2233
RFC2790	*Host Resources MIB*	S. Waldbusser, P. Grillo	March 2000	Obsoletes RFC1514
RFC2789	*Mail Monitoring MIB*	N. Freed, S. Kille	March 2000	Obsoletes RFC2249, RFC1566
RFC2788	*Network Services Monitoring MIB*	N. Freed, S. Kille	March 2000	Obsoletes RFC2248
RFC2737	*Entity MIB (Version 2)*	K. McCloghrie, A. Bierman	December 1999	Obsoletes RFC2037
RFC2720	*Traffic Flow Measurement: Meter MIB*	N. Brownlee	October 1999	Obsoletes RFC2064
RFC2708	*Job Submission Protocol Mapping Recommendations for the Job Monitoring MIB*	R. Bergman	November 1999	
RFC2707	*Job Monitoring MIB - V1.0*	R. Bergman, T. Hastings, S. Isaacson, H. Lewis	November 1999	
RFC2669	*DOCSIS Cable Device MIB Cable Device Management Information Base for DOCSIS compliant Cable Modems and Cable Modem*	M. St. Johns, Ed.	August 1999	

	Termination Systems			
RFC2667	*IP Tunnel MIB*	D. Thaler	August 1999	
RFC2621	*RADIUS Accounting Server MIB*	G. Zorn, B. Aboba	June 1999	
RFC2620	*RADIUS Accounting Client MIB*	B. Aboba, G. Zorn	June 1999	
RFC2619	*RADIUS Authentication Server MIB*	G. Zorn, B. Aboba	June 1999	
RFC2618	*RADIUS Authentication Client MIB*	B. Aboba, G. Zorn	June 1999	
RFC2613	*Remote Network Monitoring MIB Extensions for Switched Networks Version 1.0*	R. Waterman, B. Lahaye, D. Romascanu, S. Waldbusser	June 1999	
RFC2605	*Directory Server Monitoring MIB*	G. Mansfield, S. Kille	June 1999	Obsoletes RFC1567
RFC2593	*Script MIB Extensibility Protocol Version 1.0*	J. Schoenwaelder, J. Quittek	May 1999	
RFC2564	*Application Management MIB*	C. Kalbfleisch, C. Krupczak, R. Presuhn, J. Saperia	May 1999	
RFC2562	*Definitions of Protocol and Managed Objects for TN3270E Response Time Collection Using SMIv2 (TN3270E-RT-MIB)*	K. White, R. Moore	April 1999	
RFC2493	*Textual Conventions for MIB Modules Using Performance History Based on 15 Minute Intervals*	K. Tesink, Ed	January 1999	
RFC2438 BCP0027	*Advancement of MIB specifications on the IETF Standards Track*	M. O'Dell, H. Alvestrand, B. Wijnen, S. Bradner	October 1998	
RFC2320	*Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIv2 (IPOA-MIB)*	M. Greene, J. Luciani, K. White, T. Kuo	April 1998	
RFC2249	*Mail Monitoring MIB*	N. Freed, S. Kille	January 1998	Obsoletes RFC1566, Obsoleted by RFC2789
RFC2248	*Network Services Monitoring MIB*	N. Freed, S. Kille	January 1998	Obsoletes RFC1565, Obsoleted by RFC2788
RFC2233	*The Interfaces Group MIB using SMIv2*	K. McCloghrie, F. Kastenholz	November 1997	Obsoletes RFC1573, Obsoleted by RFC2863
RFC2096	*IP Forwarding Table MIB*	F. Baker	January 1997	Obsoletes RFC1354
RFC2074	*Remote Network Monitoring MIB Protocol Identifiers*	A. Bierman, R. Iddon	January 1997	Obsoleted by RFC2895
RFC2064	*Traffic Flow Measurement: Meter MIB*	N. Brownlee	January 1997	Obsoleted by RFC2720
RFC2039	*Applicability of Standards Track MIBs to*	C. Kalbfleisch	November	

	Management of World Wide Web Servers		1996	
RFC2037	*Entity MIB using SMIv2*	K. McCloghrie, A. Bierman	October 1996	Obsoleted by RFC2737
RFC2020	*IEEE 802.12 Interface MIB*	J. Flick	October 1996	
RFC1792	*TCP/IPX Connection Mib Specification*	T. Sung	April 1995	
RFC1759	*Printer MIB*	R. Smith, F. Wright, T. Hastings, S. Zilles, J. Gyllenskog	March 1995	
RFC1749	*IEEE 802.5 Station Source Routing MIB using SMIv2*	K. McCloghrie, F. Baker, E. Decker	December 1994	Updates RFC1748
RFC1748	*IEEE 802.5 MIB using SMIv2*	K. McCloghrie, E. Decker	December 1994	Obsoletes RFC1743, RFC1231, Updated by RFC1749
RFC1743	*IEEE 802.5 MIB using SMIv2*	K. McCloghrie, E. Decker	December 1994	Obsoletes RFC1231, Obsoleted by RFC1748
RFC1724	*RIP Version 2 MIB Extension*	G. Malkin, F. Baker	November 1994	Obsoletes RFC1389
RFC1697	*Relational Database Management System (RDBMS) Management Information Base (MIB) using SMIv2*	D. Brower, Editor, B. Purvy, RDBMSMIB Working Group Chair, A. Daniel, M. Sinykin, J. Smith	August 1994	
RFC1696	*Modem Management Information Base (MIB) using SMIv2*	J. Barnes, L. Brown, R. Royston, S. Waldbusser	August 1994	
RFC1612	*DNS Resolver MIB Extensions*	R. Austein, J. Saperia	May 1994	
RFC1611	*DNS Server MIB Extensions*	R. Austein, J. Saperia	May 1994	
RFC1593	*SNA APPN Node MIB*	W. McKenzie, J. Cheng	March 1994	
RFC1573	*Evolution of the Interfaces Group of MIB-II*	K. McCloghrie, F. Kastenholz	January 1994	Obsoletes RFC1229, Obsoleted by RFC2233
RFC1567	*X.500 Directory Monitoring MIB*	G. Mansfield, S. Kille	January 1994	Obsoleted by RFC2605
RFC1566	*Mail Monitoring MIB*	S. Kille, N. Freed	January 1994	Obsoleted by RFC2249, RFC2789
RFC1565	*Network Services Monitoring MIB*	S. Kille, N. Freed	January 1994	Obsoleted by RFC2248
RFC1559	*DECnet Phase IV MIB Extensions*	J. Saperia	December 1993	Obsoletes RFC1289
RFC1514	*Host Resources MIB*	P. Grillo, S. Waldbusser	September 1993	Obsoleted by RFC2790

RFC1513	*Token Ring Extensions to the Remote Network Monitoring MIB*	S. Waldbusser	September 1993	Updates RFC1271
RFC1461	*SNMP MIB extension for Multiprotocol Interconnect over X.25*	D. Throop	May 1993	
RFC1447	*Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)*	K. McCloghrie, J. Galvin	April 1993	
RFC1414	*Identification MIB*	M. St. Johns, M. Rose	February 1993	
RFC1389	*RIP Version 2 MIB Extensions*	G. Malkin, F. Baker	January 1993	Obsoleted by RFC1724
RFC1382	*SNMP MIB Extension for the X.25 Packet Layer*	D. Throop	November 1992	
RFC1381	*SNMP MIB Extension for X.25 LAPB*	D. Throop, F. Baker	November 1992	
RFC1369	*Implementation Notes and Experience for the Internet Ethernet MIB*	F. Kastenholz	October 1992	
RFC1354	*IP Forwarding Table MIB*	F. Baker	July 1992	Obsoleted by RFC2096
RFC1289	*DECnet Phase IV MIB Extensions*	J. Saperia	December 1991	Obsoleted by RFC1559
RFC1239	*Reassignment of experimental MIBs to standard MIBs*	J.K. Reynolds	Jun-01-1991	Updates RFC1229, RFC1230, RFC1231, RFC1232, RFC1233
RFC1238	*CLNS MIB for use with Connectionless Network Protocol (ISO 8473) and End System to Intermediate System (ISO 9542)*	G. Satz	Jun-01-1991	Obsoletes RFC1162
RFC1231	*IEEE 802.5 Token Ring MIB*	K. McCloghrie, R. Fox, E. Decker	May-01-1991	Obsoleted by RFC1743, RFC1748, Updated by RFC1239
RFC1230	*IEEE 802.4 Token Bus MIB*	K. McCloghrie, R. Fox	May-01-1991	Updated by RFC1239
RFC1229	*Extensions to the generic-interface MIB*	K. McCloghrie	May-01-1991	Obsoleted by RFC1573, Updated by RFC1239
RFC1227	*SNMP MUX protocol and MIB*	M.T. Rose	May-01-1991	
RFC1213 STD0017	*Management Information Base for Network Management of TCP/IP-based internets:MIB-II*	K. McCloghrie, M.T. Rose	Mar-01-1991	Obsoletes RFC1158, Updated by RFC2011, RFC2012, RFC2013

RFC1212 STD0016	*Concise MIB definitions*	M.T. Rose, K. McCloghrie	Mar-01-1991	
RFC1158	*Management Information Base for network management of TCP/IP-based internets: MIB-II*	M.T. Rose	May-01-1990	Obsoletes RFC1065 Obsoleted by RFC1213

Data source:http:// www.ietf.org

Appendix B JavaCC JJtree speciation for Policy Language

```
//simple polciy language

options {
   MULTI=true;
}

PARSER_BEGIN(SPLParser)

public class SPLParser {
}

PARSER_END(SPLParser)


SKIP : /* WHITE SPACE */
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}

TOKEN : /* Types */
{
   < INT: "int" >
 |< BOOL: "boolean" >
 |<STRING:"string">
 | <FLOAT: "float"  >
}

TOKEN : /* LITERALS */
{
  < INTEGER_LITERAL: (<DIGIT>)+ >
}

TOKEN: /*snmp general function name*/
{
<GETVAR:"getvar">
|<EXISTS:"exists">
|<SETVAR:"setvar">
|<SEARCHCOLUMN:"searchcolumn">
|<SETROWSTATUS:"setrowstatus">
|<COUNTERRATE:"counterrate">
|<COUNTER32DELTA:"counter32Delta">
|<NEWPDU:"newpdu">
|<WRITEVAR:"writevar">
|<READVAR:"readvar">
|<SNMPSEND:"snmpsend">
|<ROLEMATCH:"roleMatch">
|<CAPMATCH:"capMatch">
|<ELEMENTNAME:"elementName">
|<IC:"ic">
```

```
|<IV:"iv">
|<SETSCRATCHPAD:"setScratchpad">
|<GETSCRATCHPAD:"getScratchpad">
|<SIGNALEXCEPTION:"signalException">
|<GETPARAMETERS:"getParameters">
|<REGEXP:"regexp">
|<REGEXP_REPLACE:"regexp_replace">
|<OIDLEN:"oid_len">
|<OIDNCMP:"oidncmp">
|<INSUBTREE:"insubtree">
|<SUBID:"subid">
|<SUBIDWRITE:"subidwrite">
|<OIDSPLICE:"oidsplice">
|<PARSEINDEX:"parseindex">
|<STRINGTODOTTED:"stringToDotted">
//|<STRIING:"String"> not implemented in this version, because we use
string as a type
//|<INTEGER:"integer"> not implemented in this version, because we use
string as a type
|<TYPE:"type">
|<CHR:"chr">
|<SUBSTR:"substr">
|<STRNCMP:"strncmp">
|<STRNCASECMP:"strncasecmp">
|<STRLEN:"strlen">
|<RANDOME:"randome">
|<SPRINTF:"sprintf">
|<SSCANF:"sscanf">
|<STDACCESSLIST:"stdaccesslist">
}



/*
 * Program structuring syntax follows.
 */

void CompilationUnit() :
{
   String name;
}
{
   (
       VarDeclaration() ";"
     |
       Statement()

   )*
   <EOF>
}

void VarDeclaration() :
{ Token t; }
{
  (
    "boolean" { jjtThis.type = "BOOL"; }
   |
```

```
    "int" { jjtThis.type = "INT"; }
   |"string" {jjtThis.type = "String";  }
  )
  t = <IDENTIFIER>
  { jjtThis.name = t.image; }
}

/*
 * Expression syntax follows.
 */

void Expression() #void:
{}
{
 LOOKAHEAD( PrimaryExpression() "=" <STRING_LITERAL> )
  StringAssignment()
 |
 LOOKAHEAD( PrimaryExpression() "=" )
  Assignment()

|
  ConditionalOrExpression()
}
 void StringAssignment() #StringAssignment(2) :
{}
{
  PrimaryExpression()"=" <STRING_LITERAL>

}

void Assignment() #Assignment(2) :
{}
{
  PrimaryExpression() "="  Expression()

}

void ConditionalOrExpression() #void :
{}
{
  ConditionalAndExpression()
  ( "||" ConditionalAndExpression() #OrNode(2) )*
}

void ConditionalAndExpression() #void :
{}
{
  InclusiveOrExpression()
  ( "&&" InclusiveOrExpression() #AndNode(2) )*
}

void InclusiveOrExpression() #void :
{}
{
  ExclusiveOrExpression()
  ( "|" ExclusiveOrExpression() #BitwiseOrNode(2) )*
}
```

```
void ExclusiveOrExpression() #void :
{}
{
  AndExpression()
  ( "^" AndExpression() #BitwiseXorNode(2) )*
}

void AndExpression() #void :
{}
{
  EqualityExpression()
  ( "&" EqualityExpression() #BitwiseAndNode(2) )*
}

void EqualityExpression() #void :
{}
{
  RelationalExpression()
  (
     "==" RelationalExpression() #EQNode(2)
   |
     "!=" RelationalExpression() #NENode(2)
  )*
}

void RelationalExpression() #void :
{}
{
  AdditiveExpression()
  (
    "<" AdditiveExpression() #LTNode(2)
   |
    ">" AdditiveExpression() #GTNode(2)
   |
    "<=" AdditiveExpression() #LENode(2)
   |
    ">=" AdditiveExpression() #GENode(2)
  )*
}

void AdditiveExpression() #void :
{}
{
  MultiplicativeExpression()
  (
    "+" MultiplicativeExpression() #AddNode(2)
   |
    "-" MultiplicativeExpression() #SubtractNode(2)
  )*
}

void MultiplicativeExpression() #void :
{}
{
  UnaryExpression()
  (
```

```
    "*" UnaryExpression() #MulNode(2)
   |
    "/" UnaryExpression() #DivNode(2)
   |
    "%" UnaryExpression() #ModNode(2)
  )*
}

void UnaryExpression() #void :
{}
{
  "~" UnaryExpression() #BitwiseComplNode(1)
|
  "!" UnaryExpression() #NotNode(1)
|
  PrimaryExpression()

}

void PrimaryExpression() #void :
{
   String name;
}
{
 function()
|  Literal()
|
  Id()

|  "(" Expression() ")"

}

void Id() :
{
   Token t;
}
{
   t = <IDENTIFIER>  { jjtThis.name = t.image; }
}

void Literal() #void :
{
   Token t;
}
{
 (
  t=<INTEGER_LITERAL>
    {
       jjtThis.val = Integer.parseInt(t.image);
    }
 )#IntConstNode
|
  BooleanLiteral()
|(t=<STRING_LITERAL>
    {
       jjtThis.val = t.toString();
```

```
    }
  )#StringConstNode


}

void BooleanLiteral() #void :
{}
{
  "true" #TrueNode
|
  "false" #FalseNode
}

/*
 * Statement syntax follows.
 */

void Statement() #void :
{}
{
  ";"
|
  LOOKAHEAD(2)
  LabeledStatement()
|
  Block()
|
  StatementExpression()

|
  IfStatement()
|
  WhileStatement()
|
  IOStatement()
}

void LabeledStatement() #void :
{}
{
  <IDENTIFIER> ":" Statement()
}

void Block() :
{}
{
  "{" ( Statement() )* "}"
}

void StatementExpression() :
/*
 * The last expansion of this production accepts more than the legal
 * SPL expansions for StatementExpression.
 */
{}
{
```

```
  Assignment() ";"
}

void IfStatement() :
/*
 * The disambiguating algorithm of JavaCC automatically binds dangling
 * else's to the innermost if statement.  The LOOKAHEAD specification
 * is to tell JavaCC that we know what we are doing.
 */
{}
{
  "if" "(" Expression() ")" Statement() [ LOOKAHEAD(1) "else"
Statement() ]
}

void WhileStatement() :
{}
{
  "while" "(" Expression() ")" Statement()
}

void IOStatement() #void :
{ String name; }
{
   ReadStatement()
  |
   WriteStatement()
}

void ReadStatement() :
{ Token t; }
{
   "read" t = <IDENTIFIER>
   { jjtThis.name = t.image; }
}

void WriteStatement() :
{ Token t; }
{
   "write" t = <IDENTIFIER>
   { jjtThis.name = t.image; }
}
//added to test functions, zhifeng
void function()#void :
{
 }
{
GETVARFUNCTION()
|EXISTSFUNCTION()
|SETVARFUNCTION()
|SEARCHCOLUMNFUNCTION()
|SETROWSTATUSFUNCTION()
|COUNTERRATEFUNCTION()
|NEWPDUFUNCTION()
|WRITEVARFUNCTION()
|READVARFUNCTION()
|SNMPSENDFUNCTION()
```

```
|ROLEMATCH()
|CAPMATCH()
|ELEMENTNAME()
|IC()
|IV()
|SETSCRATCHPAD()
|GETSCRATCHPAD()
|SIGNALEXCEPTION()
|GETPARAMETERS()
|REGEXP()
|REGEXP_REPLACE()
|OIDLEN()
|OIDNCMP()
|INSUBTREE()
|SUBID()
|SUBIDWRITE()
|OIDSPLICE()
|PARSEINDEX()
|STRINGTODOTTED()
//|<STRIING:"String"> not implemented in this version, because we use
string as a type
//|<INTEGER:"integer"> not implemented in this version, because we use
string as a type
|TYPE()
|CHR()
|SUBSTR()
|STRNCMP()
|STRNCASECMP()
|STRLEN()
|RANDOME()
|SPRINTF()
|SSCANF()
|STDACCESSLIST()
}
void GETVARFUNCTION():
 { Token t;}

{
t=<GETVAR>
//|<EXISTS>|<SETVAR>|<SEARCHCOLUMN>|<SETROWSTATUS>|<COUNTERRATE>|<NEWPD
U>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id()   ")"

}
void EXISTSFUNCTION():
 {Token t;
}

{
t= <EXISTS>
//|<SETVAR>|<SEARCHCOLUMN>|<SETROWSTATUS>|<COUNTERRATE>|<NEWPDU>|<WRITE
VAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"
```

```
}
void SETVARFUNCTION():
 {Token t;
}

{
t= <SETVAR>
//|<SEARCHCOLUMN>|<SETROWSTATUS>|<COUNTERRATE>|<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SEARCHCOLUMNFUNCTION():
 {Token t;
}

{
t= <SEARCHCOLUMN>
//|<SETROWSTATUS>|<COUNTERRATE>|<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SETROWSTATUSFUNCTION():
 {Token t;
}

{
t= <SETROWSTATUS>
//<COUNTERRATE>|<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void COUNTERRATEFUNCTION():
 {Token t;
}

{
t= <COUNTERRATE>
//<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}
void NEWPDUFUNCTION():
 {Token t;
}
```

```
{
t= <NEWPDU>
//<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}
void WRITEVARFUNCTION():
 {Token t;
}

{
t= <WRITEVAR>
//<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void READVARFUNCTION():
 {Token t;
}

{
t= <READVAR>
//<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}
void SNMPSENDFUNCTION():
 {Token t;
}

{
t= <SNMPSEND>
//<NEWPDU>|<WRITEVAR>
//|<READVAR>|<SNMPSEND>>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void ROLEMATCH():
 {
Token t;
}

{
t= <ROLEMATCH>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}
```

```
void CAPMATCH():
 {
Token t;
}

{
t= <CAPMATCH>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void ELEMENTNAME():
 {
Token t;
}

{
t= <ELEMENTNAME>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void IC():
 {
Token t;
}

{
t= <IC>

{ jjtThis.name = t.image;} "(" ")"

}

void IV():
 {
Token t;
}

{
t= <IV>

{ jjtThis.name = t.image;} "(" PrimaryExpression()")"
//here PrimaryExpression() should be an int

}

void SETSCRATCHPAD():
 {
Token t;
}

{
t= <SETSCRATCHPAD>
```

```
{ jjtThis.name = t.image;} "(" PrimaryExpression()","
PrimaryExpression()"," PrimaryExpression()")"

}

void GETSCRATCHPAD():
 {
Token t;
}

{
t= <GETSCRATCHPAD>

{ jjtThis.name = t.image;} "(" PrimaryExpression()","
PrimaryExpression()"," PrimaryExpression()")"

}

void SIGNALEXCEPTION():
 {
Token t;
}

{
t= <SIGNALEXCEPTION>

{ jjtThis.name = t.image;} "(" ")"

}

void GETPARAMETERS():
 {
Token t;
}

{
t= <GETPARAMETERS>

{ jjtThis.name = t.image;} "(" ")"

}

void REGEXP():
 {
Token t;
}

{
t= <REGEXP>

{ jjtThis.name = t.image;}  "(" PrimaryExpression()","
PrimaryExpression()"," PrimaryExpression()
  ( "," PrimaryExpression()  )*")"

}
```

```
void REGEXP_REPLACE():
 {
Token t;
}

{
t= <REGEXP_REPLACE>

{ jjtThis.name = t.image;} "(" PrimaryExpression()","
PrimaryExpression()"," PrimaryExpression() ")"

}

void OIDLEN():
 {
Token t;
}

{
t= <OIDLEN>

{ jjtThis.name = t.image;} "(" PrimaryExpression() ")"



}



void OIDNCMP():
 {
Token t;
}

{
t= <OIDNCMP>

{ jjtThis.name = t.image;} "(" PrimaryExpression()","
PrimaryExpression()"," PrimaryExpression()")"
//(string,string, int)
}

void INSUBTREE():
 {
Token t;
}

{
t= <INSUBTREE>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SUBID():
 {
Token t;
```

```
}

{
t= <SUBID>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SUBIDWRITE():
 {
Token t;
}

{
t= <SUBIDWRITE>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}


void OIDSPLICE():
 {
Token t;
}

{
t= <OIDSPLICE>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void PARSEINDEX():
 {
Token t;
}

{
t= <PARSEINDEX>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}


void STRINGTODOTTED():
 {
Token t;
}

{
t= <STRINGTODOTTED>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"
```

```
}

//|<STRIING:"String"> not implemented in this version, because we use
string as a type
//|<INTEGER:"integer"> not implemented in this version, because we use
string as a type
void TYPE():
 {
Token t;
}

{
t= <TYPE>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void CHR():
 {
Token t;
}

{
t= <CHR>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SUBSTR():
 {
Token t;
}

{
t= <SUBSTR>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void STRNCMP():
 {
Token t;
}

{
t= <STRNCMP>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void STRNCASECMP():
 {
```

```
Token t;
}

{
t= <STRNCASECMP>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void STRLEN():
 {
Token t;
}

{
t= <STRLEN>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void RANDOME():
 {
Token t;
}

{
t= <RANDOME>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SPRINTF():
 {
Token t;
}

{
t= <SPRINTF>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}

void SSCANF():
 {
Token t;
}

{
t= <SSCANF>

{ jjtThis.name = t.image;} "(" Id() ( "," Id() )* ")"

}
```

```
void STDACCESSLIST():
 {
Token t;
}

{
t= <STDACCESSLIST>

{ jjtThis.name = t.image;}  "(" PrimaryExpression() ","
PrimaryExpression() ")"

}


TOKEN : /* IDENTIFIERS */
{
  < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
|
 <STRING_LITERAL: "\"" (~["\"","\\","\n","\r"] | "\\"
(["n","t","b","r","f","\\","\'","\""] | ["0"-"7"] (["0"-"7"])? | ["0"-
"3"] ["0"-"7"] ["0"-"7"]))* "\"">

|
  < #LETTER: [ "a"-"z", "A"-"Z" ] >
|
  < #DIGIT: [ "0"-"9"] >
}
```

Appendix C Glossary

ASN.1	Abstract Syntax Notation 1
BGP	Border Gateway Protocol
BCP	Best Current Practice
CCITT	International Telegraph and Telephone Consultative Committee
Cisco IOS	Cisco Internet Operating System
CLI	Command Line Interface
DN	Distinguished Name
DES	Data Encryption Standard
DNS	Domain Name System
DS	differentiated services
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
IAB	Internet Architecture Board
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	Internet Standard Organization
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
MD5	message-digest algorithm version 5
NME	Network Management Entity
NVRAM	Non_voltaile RAM
OID	Object ID
PDU	Protocol Datagram Unit
QoS	Quality of Service
RAM	Random Access Memory
RDBMS	Relational Database Management System
RED	Random Early Detection
RFC	Requests for Comments
SMI	Structure and Identification of Management Information
SNMP	Simple Network Management Protocol
SNMPCONF	Policy Based Configuration with SNMP
TCP	Transmission control protocol
TL1	Transaction Language 1
UDP	User Datagram Protocol
USM	User-based Security Model
VACM	View-based Access Control model
WRED	Weighted Random Early Detection
X.500	A CCITT specification for directory services
YACC	Yet Another Complier Compiler