CIM and XML in Network Management

By

Haohong Shen

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of

Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

CIM and XML in Network Management

BY

Haohong Shen

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of Master of Science

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Acknowledgements

I would like to thank all those who helped me write this thesis. In particular I would like to say a special thank you to Dr. David Blight and Dr. Bob McLeod who provided invaluable guidance through the years and gave me suggestions whenever I asked for. I would also like to thank those I worked with in the lab.

I am grateful to my family for always being there for me. Their love, support and encouragement sustain me through difficult times.

Abstract

As the popularity of Internet soars, new applications are being adapted to run on the Internet due to the market demand. Network device vendors install more functions on different types of network devices to solve specific problems but, in so doing, complicate the interoperation of these devices with each other. The proliferation of protocols, along with the reliance on individual device management, exacerbates this problem and makes managing network devices even harder.

The use of Directory Enabled Networks (DENs) simplifies the tasks of network administrators. DEN uses directory as a special purpose database that contains information about the nodes, or devices, attached to a network. The directory is structured around network objects: users, applications, printers, file servers, switches, routers, remote access servers, etc. For each object there is a set of attributes stored in the database. Directory support for device configuration management will enable network devices to retrieve their configuration parameters from a directory server, rather than requiring the network manager to perform box-by-box configuration. The Distributed Management Task Force (DMTF) Common Information Model (CIM) presents a consistent view of the managed environment that is independent of the various protocols and data formats used by those devices and applications. It can be used to describe network objects. The thesis presents detailed description of DEN and CIM. It also introduces XML (eXtensible Markup Language), which is powerful enough to express complex data structures to satisfy the need of complex applications. In system management environments, the XML representation of

management information could be used to transfer data between co-cooperating management platforms.

As the acceptance of the Internet by the business community increases, companies realize that sharing network resources by using the Internet is definitely cost-effective than using dedicated facilities. One of the major problems with messaging on the Internet is insufficient security. Because the Internet is a public network, it offers little protection to the data it carries. Virtual Private Networks (VPNs) offer operational savings while maintaining the security associated with private network infrastructure. Three tunneling protocols: Point-to-Point Tunneling Protocol (PPTP), Layer 2 Tunneling Protocol (L2TP) and IP Security Protocol (IPSec) are discussed in details in Chapter Three. The VPN solution is implemented at the management level and at the packet level.

This thesis demonstrates the use of CIM and XML in a network management environment. A VPN policy file is used in the implementation of the network management system.

CONTENTS

FIGURES

TABLES

Acronyms and Abbreviations

AH	Authentication Header
CIM	Common Information Model
DAP	Directory Access Protocol
DEN	Directory Enabled Network
DMTF	Distributed Management Task Force
DTD	Document Type Definition
ESP	Encapsulating Security Payload
HTML	HyperText Markup Language
IPSec	IP Security Protocol
L2TP	Layer 2 Tunneling Protocol
LDAP	Lightweight Directory Access Protocol
MOF	Managed Object Format
NMS	Network Management System
PPTP	Point-to-Point Tunneling Protocol
QoS	Quality of Service
SA	Security Association
SGML	Standardized Generalized Markup Language
SNMP	Simple Network Management Protocol
SPI	Security Parameter Index
UML	Unified Modeling Language
VPN	Virtual Private Network
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	eXtensible Style Language

1. INTRODUCTION

1.0 Introduction

The Internet was originally designed for the movement of files between research sites, and for the relay of electronic mail on a packet switched network. The network provides best effort service to all traffic, and the application is responsible for effectively taking advantage of that service. People are demanding more robust services for less cost, which may be achieved by using Internet technologies [1]. New applications are being adapted to run on the Internet due to market demand. To support those new applications, network device vendors install more functions on different types of network devices to solve specific problems. The result is complicated interoperation of these devices with each other. The proliferation of protocols, along with the reliance on individual device management, exacerbates this problem and makes managing network devices even harder. Effective device configuration is a significant problem facing network administrators today. As networks get bigger, the problem gets more complex.

The use of Directory Enabled Networks (DENs) simplifies the tasks of network administrators. DENs describe a philosophy for transforming the network from a passive collection of devices that route and forward traffic to an active set of cooperating devices that intelligently provide services to the user [2]. A directory is a special purpose database that contains information about the nodes, or devices, attached to a network. In a DEN, the directory is structured around network objects: users, applications and devices. For each object there is a set of attributes stored in the database. Directory support for device

configuration management enables network devices to retrieve their configuration parameters from a directory server, rather than requiring the network manager to perform box-by-box configuration via Telnet, Web, or GUI-based management alternatives. When a network device is initialized, it may use a simplified access protocol such as, Lightweight Directory Access Protocol (LDAP) [4] to query the directory server and obtain its configuration parameters.

A fundamental problem with directory services is that as the size and complexity of an enterprise's network infrastructure grows, the number of different directory services also increases [2]. Much of the information contained in any given directory is duplicated across multiple directory services. This results in inefficient and labor-intensive processes for adding, deleting, moving, and changing information. Whenever there is a modification to a record in one directory, that change must be manually cascaded through all the other directories on the network. The chance that the network as a whole will have conflicting configurations grows. Configuring policies on a device-by-device basis is difficult and can lead to inconsistencies. Policy-based networking solves this problem by using a highly distributed, logically centralized system to administratively control the various components of a network and the policy system itself.

To manage a network, the managing system and the managed system need to exchange information about the state of the network. A network may include devices from different vendors. To enable different vendors to provide interoperable network services, we need a common information model to describe such information so that all the network devices can understand the information. The Distributed Management Task Force (DMTF)

Common Information Model (CIM) [6] is such an information model. CIM includes a core model and eight common models, each of these is an object-oriented information model, which can be built upon to model and represent managed systems and their components.

To exchange information, network management systems need a communication protocol and an information encoding scheme. The eXtensible Markup Language (XML), which is a subset of the Standard Generalized Markup Language (SGML), can be used as the encoding scheme. XML is similar in concept to HyperText Markup Language (HTML), but whereas HTML is used to convey display information about a document, XML is used to represent structured data in a document. An XML document can optionally have a description of its grammar attached. The grammar for an XML document is described using a mechanism known as a Document Type Definition (DTD). The DTD describes the allowable elements in the XML document and describes the structure of those elements.

Because XML is based upon an open industry standard, the XML representation of management information could be used to transfer data between co-cooperating management platforms, which need not necessarily be running the same operating environment. Interoperability is enabled because of a common understanding of the XML representation of management information. XML is particularly suitable as a data representation mechanism for use in heterogeneous environments. However, to represent CIM data, an XML vocabulary must be defined and agreed upon. The XML management vocabulary would be in effect a mapping of the CIM meta-model to an XML DTD.

Any communication protocol such as, simple network sockets, HTTP, etc., can be used to transport the data depending on the needs and the preferences of network administrators. HTTP emerges as the ideal candidate because of the wide spread of browsers, their simple user interface and wide accessibility.

A simple network management system based on the work done by Dr. D. Blight [17] [18] was developed in this thesis. The system models a policy-based network. The network management system communicates with the network via simple network sockets. The information exchanged between them is encoded in XML. The policy server of the managing system listens to requests from users. A user sends a policy request to the managing system. A policy that realizes a virtual private network (VPN) is used. The reason why a VPN policy was chosen is because VPN uses the Internet as a channel for private data communications. More and more businesses are implementing VPNs to utilize the low cost of the Internet. The Internet includes different networks and devices from different vendors.

This thesis is divided into five chapters. Chapter one is the introduction section, which discusses the need for policy-based networks, the idea of using CIM and DENs in the implementation of policy-based networks and XML as the transport scheme for network communications.

Chapter two provides a brief introduction to Unified Modeling Language (UML). It defines the goal and structure of DENs. It also describes LDAP. The motivation for using policy-based network management is also discussed. A detailed example of an emerging policy architecture is presented. Chapter two also includes the motivation for

CIM and detailed information on its Core Model and the eight Common Models. The chapter concludes with an introduction to XML and its use in network management systems.

Because a VPN policy file will be used in the simple network management system, Chapter three is used to describe VPNs. It begins with an introduction to VPNs and the financial benefits that VPNs provide. It discusses three tunneling protocols: PPTP, L2TP, and IPSec. The first two establish tunnels at Layer 2 and are connection oriented. The last one operates at Layer 3. IPSec is discussed in detail because it has emerged as the protocol due to its encryption capability.

Chapter four presents a CIM implementation. Though the application did not fully implement CIM, it provides the foundation which can be built upon. The class hierarchy and major classes for the implementation are discussed in detail. It also includes procedures, which can be used to convert Managed Object Format (MOF) files to XML and to Java program files. The Java program files can then be extended according to the specific needs of the network administrator to customize network applications. The implementation includes a server, which listens to the users for VPN policy requests. The server checks to see if the requested file is in the system. If it is, it pulls the file from the directory and converts it to XML format. The user receives the policy in XML format, reads it and takes action accordingly. To take a look at specific aspects associated with VPNs, IPSec is implemented in the implementation as well.

The thesis concludes with chapter five, which includes suggested future work.

2. NETWORK MANAGEMENT

2.0 Directory Enabled Networks (DENs)

A directory service is a physically distributed, logically centralized repository of infrequently changing data that is used to manage a computing environment [1]. The directory typically stores information in a hierarchical fashion thus providing a unique namespace for each entry. Each object in the directory has a set of attributes that describe the information carried by the object. A directory service provides a means for locating and identifying users and available resources in a distributed system. In today's Internet based network computing, a directory enabled network uses directory services to store critical information to facilitate access, management, and search operations. A DEN may use LDAP to access, manage, and manipulate directory information.

As the Internet penetrates enterprise networks and Intranets become Extranets, networks as a whole are becoming increasingly complex. There are different types of network elements, each running a potentially different set of protocols and services over possibly different media. Consequently, a network may have too many different directory services for network administrators to manage successfully. The key to solving this problem is to switch to a new paradigm—policy based management. Policy based management allows network administrators to manage device configurations, users, and services via a central directory enabled policy management system.

2.0.1 Principal Goals of DENs

DENs have four main objectives [2]:

1. to model network elements and services, and their interaction with other elements in a managed system,
2. to provide the means for interoperable network-enabled solutions to be built,
3. to enable applications to leverage the power of the network without requiring the user to know or set esoteric network related information, and
4. to define a way to manage the network, as opposed to an element within a network.

2.0.2 Intelligent Networks

Historically, networks have treated the traffic from every user and application the same. As the applications using the Internet grow, enterprises, service providers, and even small businesses are changing to a more intelligent, services-oriented model wherein the network can provide differentiated services to certain users and applications, Thereby adding intelligence to traditional networks.

In a DEN, network administrators store policies and information about the network in a directory. Network devices and applications publish information about them to the directory and obtain information about other resources from the directory. In an intelligent network, a robust information model is used for modeling the network and its interaction with the environment. A directory serves as a unified information repository and also contains specialized system components (for example, policy servers) for managing and coordinating information exchange.

The best way to define the features and functions of an intelligent network is to see how it would be managed and function compared to the way existing networks are managed and function (Table 1) [2].

Table 1. Comparison between current networks and the intelligent networks

Feature or Function	Existing Network Implementation	Intelligent Network Implementation
Configuration Approach	Focus is on configuring individual devices	Focus is on configuring the network and ensuring that all devices in the network work together
Configuration Process	Separate processes to configure people, applications, and devices	Links devices to people, applications, and other resources as part of the same process
Management	A set of loosely related applications that are oriented towards managing individual devices	One or more layered applications that manage the network as well as devices in the network
Policy	If present, used to control individual device configurations	Integral part of the system. Used to control the configuration of the system and its components
Use of Directory	Isolated functionality, if used at all	Integral—It is a unifying information repository
Use of an Information Model	Not Used	Integral—it models network elements, the network, network services, and how the network relates to the rest of the system
Passive or Active System	Passive—network elements are assigned a role to play in providing network services	Active—network elements play a dynamically changing role in providing network services
Modeling System Components	Treat applications, devices, services, and people as separate entities	Treat applications, devices, services, and people as an integral partnership

2.0.3 Lightweight Directory Access Protocol (LDAP)

X.500 is the name given to a series of standards (ISO 9594, Data Communications Network Directory, Recommendations X.500-X.521) developed by the International Standardization Organization (ISO) that specify how information can be stored and accessed in a global directory service [3]. The primary construct holding information in the directory is the "entry". Each entry, which is built from a collection of attributes, contains information about one object. A main distinction of X.500 is that it organizes directory entries in a hierarchical namespace capable of supporting large amount of information. A sample entry in the directory could be a user's name, phone number, title, country name and organization name. In this example, the X.500 hierarchy would be country, organization within the country, and individual name within the organization. Once traversing to the individual, a single collection of attributes about that individual can be stored, as in the example, a title and a phone number. Additional information can be added at the lowest level. As a standard, X.500 enables the development of an electronic directory of people in an organization so that it can be part of a global directory available to anyone with Internet access.

X.500 specifies that communications between the directory client and the directory server use the directory access protocol (DAP) [4]. LDAP is designed to provide access to the X.500 directory without incurring the resource requirements of the Directory Access Protocol (DAP). The LDAP specification defines standard communications methods for inputting and accessing information in directories but does not make any statement about the structure or contents of a directory. LDAP pulls information from the directory. It runs

directly over TCP, and can be used to access a standalone LDAP directory service (Figure 1) or to access a directory service that is back-ended by X.500 (Figure 2).

In Figure 1, a LDAP server is used to store and access the directory itself. A standalone LDAP server eliminates the need for the OSI protocol stack. This makes the LDAP server much more complicated, since it must store and retrieve directory entries, which are functions normally performed by X.500 directory server.

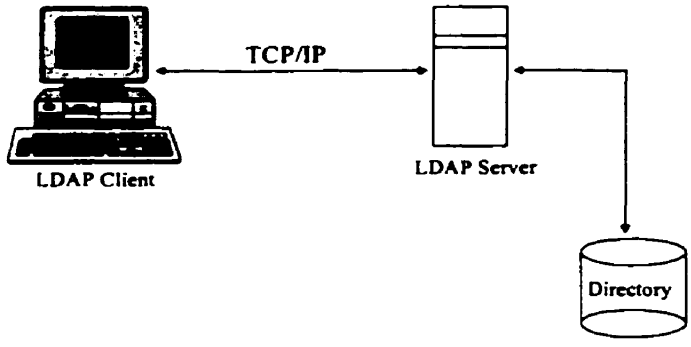

Figure 1. Standalone LDAP server

In Figure 2, an application client program initiates an LDAP message by calling an LDAP API. But an X.500 directory server does not understand LDAP messages. To solve the problem, the LDAP client communicates with an LDAP server that forwards requests to the X.500 directory server. The LDAP server services requests from the LDAP client by becoming a client of the X.500 server. As indicated in the figure, the LDAP server communicates with LDAP client using TCP/IP and communicates with the X.500 server using OSI protocols.

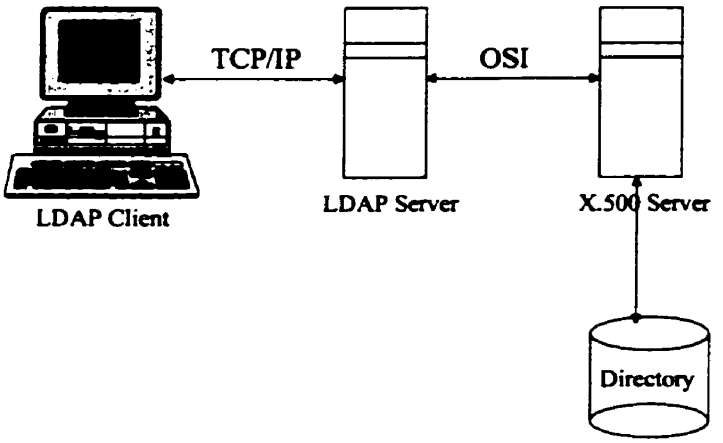

Figure 2. LDAP server acting as a gateway to an X.500 server

Because LDAP uses a common client/server interface for setting up and maintaining network devices across a heterogeneous network environment, it provides vendor interoperability. The security, policy and network configuration files are located in the directory, thus, all the components of a network have easy access to the desired information. This, in turn, reduces the need for redundant information.

2.0.4 Network Management

Since many parameter settings for various devices (such as routers and switches) are the same, the configuration files and parameters can be effectively stored in a directory server. Directory support for device configuration management allows network devices to retrieve their configuration parameters directly from the directory server. The benefits are obvious as the size and scope of an organization's network expands. If a network has router configuration files stored in the directory server, when new routers are introduced to the network, they will not have to be individually configured. Instead, the routers can examine

the directory for appropriate router configuration parameters and then go online on their own. Figure 3 shows this scenario.

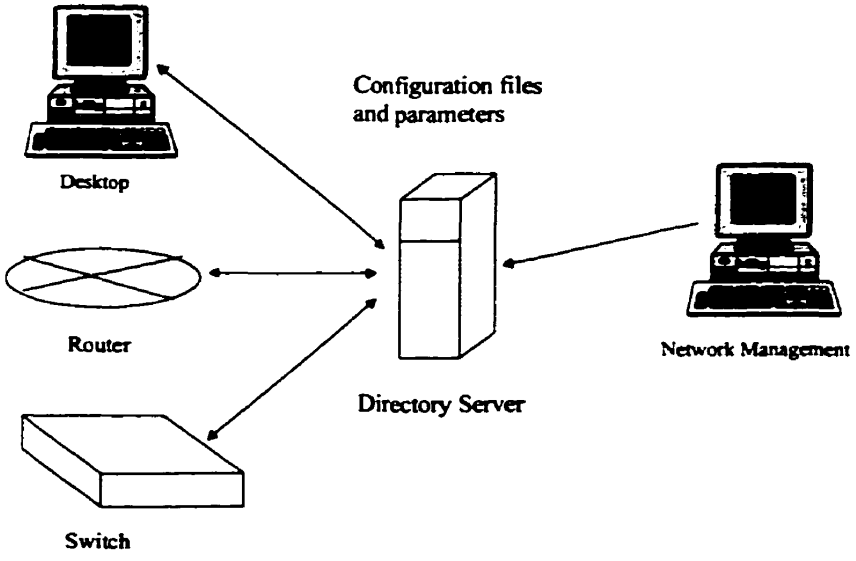

Figure 3. Centralized device configuration management

2.1 Policy-Based Networks

New tools, applications, and services are causing phenomenal growth in the volume of users on the Internet. There are many businesses that conduct some of their businesses on line. Each day, companies strive to correlate business decisions to things that actually happen on their network, which includes:

- access control—selecting which users have access to which network resources,
- application prioritization—prioritizing applications according to their roles in company operations,

- service differentiation—delivering bandwidth and services to each customer according to their needs and the application they are using, and
- traffic management—managing the traffic through networks.

All of these actions require applying corporate business policies to specific network actions. To ensure the end-to-end network performance meets the businesses needs, network administrators need the ability to make and deploy rules, which automate configuration and management of the services, for access and use of network resources. A comprehensive, policy-based system allows the network administrators to define, in a succinct and organized fashion, policies that automatically effect changes on specific equipment in the network environment.

Over time progresses, stated policies are deployed further and further into the network, producing intelligent business-driven systems. This leads to consistency and predictability of network operation, and superior operation results. It gives the ability to define and control operations according to centralized policies with the needs of the business foremost. This is Policy-based networking [5]. Policy-based networking is much more complex than device configuration management. The main difference between a directory configuration system and a policy-based networking system is state. In a simple directory configuration, a device pulls a static configuration from the central repository. In a policy-based system, a policy server interprets the stored static information in the context of other circumstantial data. Thus, network operation is enabled through application of simple and consistent guidelines. One example is a policy file that specifies the time at which a particular application can be run by a group of users on a network. The static

information—user profiles are stored in the directory. The policy server has to know the type of the application and the identity of the user in order to make a decision.

In a standard policy-based network, policies consist of two components [2]:

- A set of conditions under which the policy applies (this might include parameters such as user name, addresses, protocols and applications types.), and
- A set of actions that apply as a consequence of satisfying or not satisfying the conditions including bandwidth guarantees, access control, service load balancing, cache redirection, and intelligent routing.

Policy-based networking employs a policy server that accesses information from multiple sources, collecting all relevant information, making policy decisions, and communicating these decisions to the network devices. A sample configuration for a policy-based network is shown in Figure 4.

The network is composed of a policy manager, policy server, applications server, and directory server. The policy manager is a user interface program, which allows the network administrators to create new policies or to modify existing policies. The policy server sends policy information to policy decision points (PDPs), where decisions about a network are made according to the pre-set rules and current network states. The directory server provides information storage, where information such as user profiles and configuration files for devices is stored. The policy server can use LDAP to retrieve information from the directory server. The network nodes (routers and switches) are where policy decisions are implemented. Sometimes, the policy conditions depend on the values

of certain fields in the packet header. The policy server can not make a policy decision since it does not have specific knowledge. The router or switch must make a policy decision in this case. It uses LDAP to retrieve necessary information from the directory server in order to make a decision.

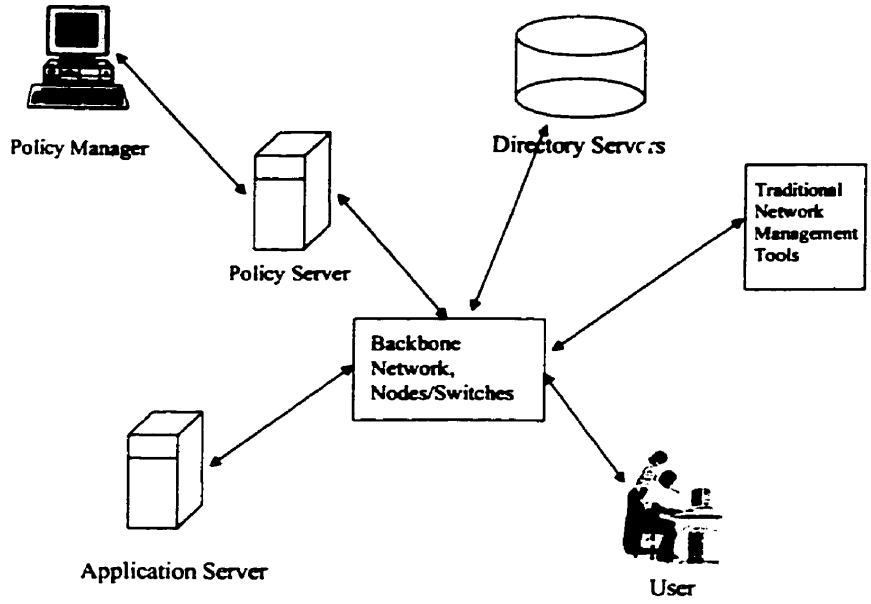

Figure 4. A sample configuration for policy based network

In this sample configuration (Figure 4), a user launches an application by sending a request to the application server. The launch of the application triggers a policy request. A router or a switch submits the request to the policy server. The policy server uses LDAP to access the user's profile and other related information from the directory server. Then the policy server formulates a response to send to the policy client, in this case a router or

switch, whether it should permit or deny the user's request to access the application. The policy-enabled router or switch carries out enforcement of policy decisions.

Configuration complexity, inconsistent policies, lack of application control, and insufficient visibility are four main problems in building intelligent networks. Policy-based networks solve these problems by using automated policy configuration, centralized policy control, network application recognition, and active audit of policies. This provides network administrators integral control over the network, which in turn reduces operating cost and gives the company competitive advantages.

The benefits of policy-based networking include:

- Network administrators can prioritize services according to which applications are critical to business needs during what specific periods and for certain users,
- Automation of network configuration and user administration reduces the cost of ownership and the network administrators' productivity is improved as well,
- Integral control over the entire network enhances overall end-to-end consistency. Network administrators can apply consistent behaviors across the entire network domain they must manage, and
- Network security is strengthened because corporate exposure is minimized via a consistent, ever-present security implementation for resources and information.

2.2 CIM

The capability to manage and maintain a comprehensive information model that enables each of the network components to be related to each other is the key to use of a DEN. The Distributed Management Task Force (DMTF) Common Information model

(CIM) is one such model. It is an object-oriented information model that describes how a system may be managed. CIM's major goal is to present a consistent view of the managed environment independent of the various protocols and data formats supported by those devices and applications [6].

CIM supplies a set of classes that are used to implement an inheritance hierarchy and a set of relationship hierarchies. These classes can be used as building blocks to model arbitrarily complex systems. Since CIM is a standardized information model, components and management applications by different vendors can share information using CIM. This allows integration of different management information from different sources that use different syntaxes and expressions.

CIM is based on an object-oriented model. Object oriented modeling is a formal way of representing something in the real world. Object models are built from classes that relate to each other through associations and generalization. The object model describes a system's data structure, its objects and their relationships. A class is a description of a group of objects with similar properties (object attribute), common behavior (operations and state diagrams), similar relationships to other objects, and common semantics. CIM uses Unified Modeling Language (UML) [7] as its modeling language. In UML, a class is represented by a rectangle containing the name of the class. A class with properties is represented by a rectangle divided into two regions, one containing the name of the class and the other a list of properties. Methods are represented by a third region containing the list of methods. An edge drawn between a subclass and its superclass with an arrow indicating the superclass, represents inheritance, or a subclass/superclass relationship.

Associations are represented by edges with the name of the association usually placed near the center of the line [7]. Associations are classes in CIM. Therefore, it is possible for an association to have properties. Aggregation is a kind of association between a whole, called the assembly, and its parts, called the components. It is drawn with a small diamond added next to the assembly end.

An example of UML is shown in Figure 5. The following can be concluded from the example: boats have owners; owners are persons. Person has two properties—Name and Address. Boat has two properties—Passengers and Displacement. A boat may be either a sailboat or a powerboat. A powerboat contains an engine. The usage of a boat is the number of days a particular user has used that boat.

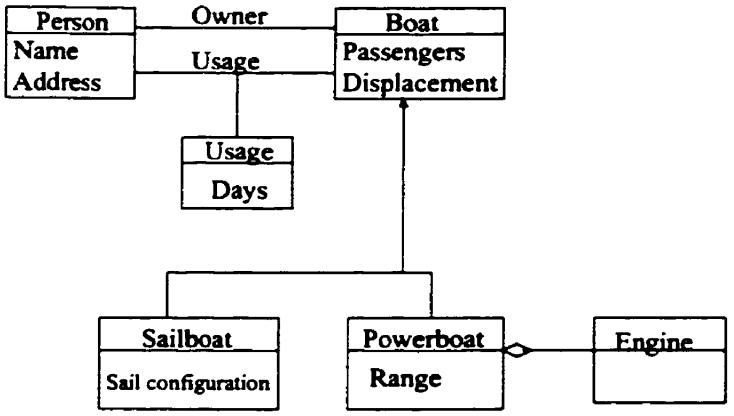

Figure 5. Illustration of UML

Property data types are limited to the intrinsic data types or arrays of those data types. Structured types are constructed by creating new classes [6]. If the property is an array property, the corresponding variant type is simply the array equivalent of the variant for the underlying intrinsic type. Table 2 lists the intrinsic data types and their interpretation.

Table 2. The intrinsic data types and their interpretation

Intrinsic Data Type	**Interpretation**
Uint8	Unsigned 8-bit integer
Sint8	Singed 8-bit integer
Uint16	Unsigned 16-bit integer
Sint16	Signed 16-bit integer
Uint32	Unsigned 32-bit integer
Sint32	Signed 32-bit integer
Uint64	Unsigned 64-bit integer
Sint64	Signed 64-bit integer
String	UCS-2 string
Boolean	Boolean
Real32	IEEE 4-byte floating point
Datetime	A string containing a data-time
<classname> ref	Strongly typed reference
Char16	16-bit UCS-2 character

CIM is structured into three distinct layers: Core Model, Common Model and extension schema. The core model is the topmost layer, which is composed of a small set of classes. It is an information model that captures notions that are applicable to all areas of management. The common model is the second layer. Like the core model, it is composed of a set of classes, as well. It captures notions that are common to particular management areas, but independent of a particular technology of implementation. There

are eight common models: System, Device, Application, Network, Physical, User, Policy, and Database. As the names imply, each of them is concerned with a certain part of networking. The third layer of CIM is called the extension schema. It represents technology specific extensions of the common model. These schema are specific to environments, such as operating systems.

The CIM core model is a good starting point for building a desired network application model since it is not specific to any platform. At the root of the class hierarchy is an abstract class: CIM_ManagedSystemElement. Any component of a system that can be managed should be its subclass. It has two subclasses: CIM_LogicalElement and CIM_PhysicalElement. CIM_PhysicalElement is the root of the physical portion of the class hierarchy. It defines common information a physical device should have, such as manufacturer information and model information. It could be used to distinguish products from different vendors. The CIM_LogicalElement is an empty class whose sole purpose is for classification. The reason for this is that it is very hard to abstract the common elements of logical functions because of diversity. Four subclasses of CIM_LogicalElement, CIM_System, CIM_LogicalDevice, CIM_Service and CIM_ServiceAccessPoint are defined in the core model. To model different logical aspects of a CIM_ManagedSystemElement, one can build concrete classes based on these four classes. The core model also defines various types of associations. These associations are represented by abstract classes, which relate elements of CIM_ManagedSystemElement.

The common model provides a set of base classes for extension into the area of technology-specific schemas. The eight common models and their definitions are shown in

Table 3. Each of these common models includes many classes and relations that are not listed in this thesis.

Table 3. The eight common models

System	Defines the key components of a system and how to assemble them. These include computer system, operating system, file, and processes.
Device	Defines how to realize physical devices in hardware, along with how to model the connections between devices. These include mass storage devices, media, sensors, printers, power supplies, and other components.
Application	Defines how to manage the installation of software in a system. This includes the concepts of software features and elements as well as the ability to check if conditions are met for installing software and actions to be taken as part of executing the software.
Network	Defines specialization to the physical and logical element class hierarchies to model network elements and devices. This also includes modeling network protocols and network systems.
Physical	Defines the physical organization, containment structure and composition of devices and device interconnections.
User	Models users, groups and organizations, and how these objects interact with the other components of a managed system.
Policy	Builds on the original policy model proposed by DEN and generalizes this beyond network policy to a policy that can control any type of managed entity.
Database	Models the operation of relational database (this is still work in progress).

CIM, as an implementation-neutral schema, facilitates the common understanding of management data across different management systems and management information integration from different sources. CIM is a conceptual model that is not bound to a particular implementation. This allows it to be used to exchange management information in a variety of ways.

Each CIM model is described in a language called the Managed Object Format (MOF). This language is based on a standard Interface Definition Language (IDL). The

purpose of MOF is to define the capabilities of a distributed service along with a common set of data types for interacting with those services. Figure 6 shows an excerpt from the CIM Core model file. The class definition is defined using the keyword "class" and is enclosed in braces. Qualifiers are enclosed in brackets and appear immediately before the component they affect. The name of the class is defined first. A colon separates the class name and its superclass name if it has one.

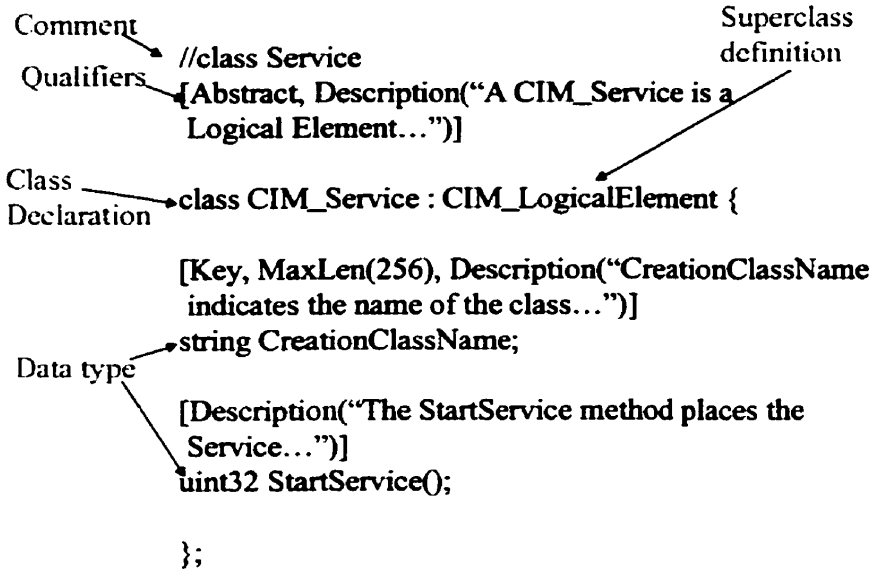

Figure 6. Expressing a CIM class in MOF

CIM is abstract in nature—all the classes are not fully implemented. It only provides a skeleton for developing network applications. The methods in all classes are only described in terms of their parameters, the parameters' types, order, and whether they are inputs. The contents of the methods are not specified. Since it is impossible to

anticipate the specific needs of each product, applications must provide support for some or the entire core model plus some or all the appropriate common models. Since the current mechanism for describing management information is in MOF, applications must be able to import and export properly formed MOF constructs.

2.3 XML

HTML is easy to learn because it uses a fixed tag set to mark up documents. The ability to incorporate images and audio into a document makes it popular among web developers. With free Web browsers such as Microsoft's Internet Explore and Netscape's Navigator being deployed universally, HTML has become one of the primary means to mark up documents delivered via the Web. However, one of its advantages, its fixed set of tags is also one of its most significant disadvantages. The only way to add functionality to HTML via new tags is to take a proposal detailing the desired functionality to the World Wide Web Consortium (W3C). The discussion process can be lengthy, and not all proposed tags are general enough to be included in future HTML specifications.

This problem of limited flexibility can be solved with XML. XML is similar to HTML in the sense that it is a text based markup language. With a few exceptions, most tags in HTML are for formatting purposes. The tags instruct the browser how a piece of text ought to be displayed. Browsers simply ignore any tags they do not recognize. XML does not have a predefined set of elements, rather it allows XML vocabularies to be defined using a Document Type Definition (DTD). With XML, you can define your own set of tags by means of a DTD. All the tags used by HTML can be easily described using XML.

Sometime, an XML document is made up of elements that consists of textual information contained between a start tag, and an end tag. The information between the start tag and the end tag is referred to as the content or data. Each element must have a start tag and an end tag. Although elements may be nested, they must not overlap.

A unique feature of XML documents is that they are self-validating—the rules of the standard used to create the document can themselves be part of the document. There are two types of validity. The first type deals with the physical structure of the document. A document that meets the criteria described earlier is called well formed. In the second level of validation, a Document Type Definition (DTD) accompanies the document. The DTD can be part of the document or an external document. The DTD specifies the grammar of what the XML document contains.

An example of a simple XML document is shown below:

```
<?xml version = "1.0?">
<!DOCTYPE name [<!ELEMENT name (first, last)>]>
<name>
        <first>John</first>
        <last>Doe</last>
</name>
```

In this example, three tags <name>, <first>, <last> were created to describe a person's name. The DTD included in the document specifies that the name entity must include two other entities called "first" and "last". In the above example, a name entity is composed of two parts: a first name and a last name. It also shows the nested relationship between these entities.

The first and the largest benefits of XML are its simplicity and extensibility. It is a character-based format, therefore, XML messages can easily be read, created, and modified by widely available text editors. In XML, tags can be named with understandable strings. An XML document has a rooted tree structure. For many applications, a tree structure is general and powerful enough to express fairly complex data. Thus XML has sufficient power to express complex data structures to satisfy the needs of many applications. Another important benefit of using XML is its capability to handle international character sets. Today, many businesses are international in scope. This is especially true for Internet applications because the Internet easily leaps national borders. XML 1.0 is defined based on the ISO-10646 (Unicode) character set so all possible characters are legal characters.

An XML document by itself is merely a collection of data. A programming language is required to process that data and perform useful operations using the data. Just as a browser brings HTML to life, programming languages can bring XML documents to life. Java is the ideal choice among all the languages because it offers platform independence, built-in Internet support and good support for internationalization.

Figure 7 illustrates a simplified model of a Java distributed application that processes XML. In this model, the processors are implemented using Java. The sender end processor obtains data from a data source, then processes the data and ultimately produces a document object model (DOM) representation. The processor passes the DOM representation of the XML data to the sender. The receiver receives the XML data and passes it to its processor. The receiver end processor processes the XML data and stores data in its data store.

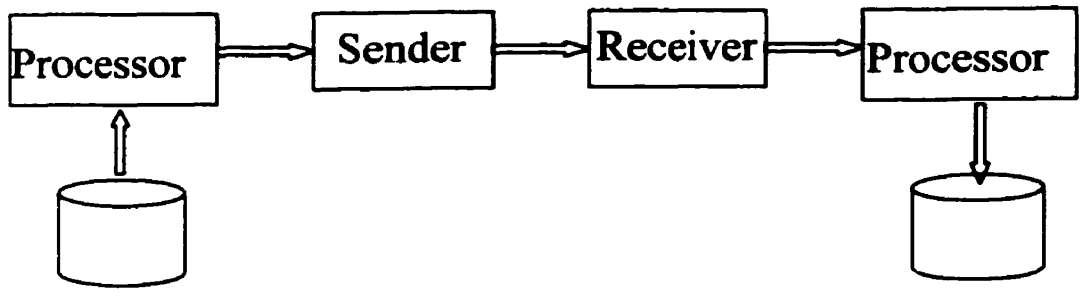

Figure 7. A simplified model of a Java distributed application that processes XML

XML over HTTP allows for a smooth transition from HTML-based sites to an XML-based site. A possible implementation is shown in Figure 8. The sender converts the DOM representation of the XML into text and sends the text to the receiver. The receiver then converts the text back to the DOM representation.

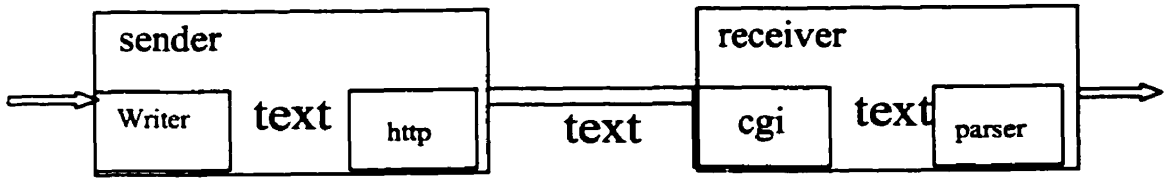

Figure 8. A sender and receiver sending XML text via HTTP

CIM does not define a standard for data exchange. XML has been proposed to define declaration of CIM objects that can be easily transformed into many representations. For illustrative purposes, one possible mapping of a CIM object to XML is shown in Appendix A. The class used in Appendix A is CIM_ManagedSystemElement.

3. VIRTUAL PRIVATE NETWORKS

3.0 Virtual Private Networks (VPNs)

The introduction of the World Wide Web (WWW) and HTML, fueled the steady acceptance of the Internet by the business community. The Internet provides valuable business opportunities to such enterprises. There have been two major technical problems with conducting businesses on the Internet. One is agreeing on a standard message format to use. XML can be used as the standard message format because of its flexibility. The other technical problem is insufficient security. The Internet is a public network, and offers little protection to the data it carries. If data integrity can not be trusted, the messaging can not be trusted either. A private network offers one of the best security environments for data transitions in a global economy. However, private networks are expensive. Installation, setup, and routine administration can easily run into millions of dollars. Virtual private networks afford enterprises the security, performance, availability, and multiprotocol environment of a private network over the inexpensive Internet.

There are primarily three types of use for VPNs: remote access, Extranet, and dial-in access. VPNs reduce network operational cost because users can use the Internet instead of dialing a dedicated number, which could be long distance, to access the corporate network and the corporations do not need to install private access lines to their sites.

Virtual Private Networking technology provides the ability to use the public Internet as an appropriate channel for private data communication. It accomplishes this by creating a "tunnel" through the Internet. To a user, it appears as if the data is being sent over a dedicated private link—despite the fact that this communication occurs over a public network.

3.1 Tunneling Protocols

Tunneling is a technology that enables one network (private network or enterprise network) to send its data via another network's (the Internet) connections. Instead of sending a frame as produced by the originating node, the tunneling protocol encapsulates the data in a new packet with an additional header. The additional header provides routing information. The original IP header and the packet payload or transport data are encrypted. In the case of IPSec, the outer header, put in position by IPSec, is also authenticated. The authenticated and encrypted IP packet is routed over the Internet to the destination. To ensure the security, each tunnel uses its own encryption key. The tunnel server selects the appropriate encryption key for authentication and encryption operations.

Some mature tunneling technologies include the Point-to-Point Tunneling Protocol (PPTP) developed by Microsoft, the Layer 2 Tunneling Protocol (L2TP) developed by Cisco, and IP Security (IPSec) Tunnel Mode. For a tunnel to be established, both the tunnel client and the tunnel server must, of course, be using the same tunneling protocol.

3.1.1 The Point-to-Point Tunneling Protocol (PPTP)

The PPTP is an extension to the Point-to-Point Protocol (PPP). It uses a TCP connection for tunnel maintenance. The data flowing through the tunnel are Generic Routing Encapsulation (GRE) encapsulated PPP frames. In essence, PPTP wraps PPP packets in IP. A remote user using PPTP can work in either a regular LAN environment or an Intranet environment as if they were locally connected. An example of how a PPTP packet is assembled prior to transmission is shown in Figure 9 (The example shows the use of PPTP in the case of remote access.).

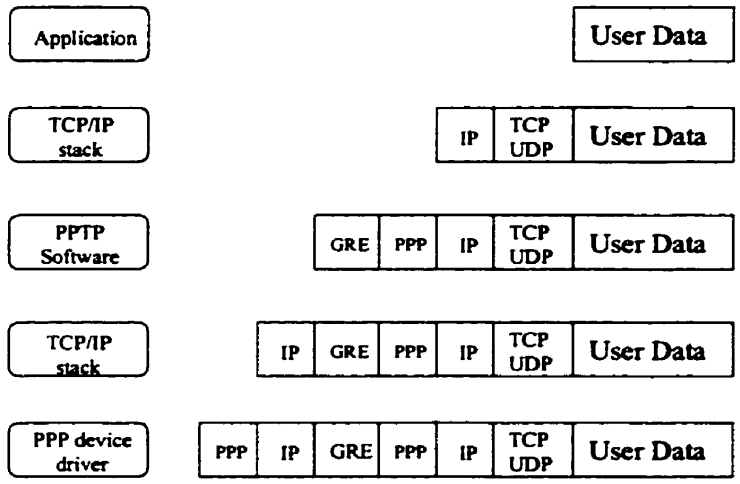

Figure 9. Construction of a PPTP Packet

The application generates a datagram, which is sent to the network. Depending on the type of application, the network transport layer adds either a TCP or UDP header to the datagram. The PPTP software, which resides in the network access server, adds its header to the new datagram. Before sending it out to the network, an IP header and a PPP header are added. The final frame layout shows the encapsulation for the dial-up client.

3.1.2 The Layer 2 Tunneling Protocol (L2TP)

L2TP is a network protocol that encapsulates PPP frames to be sent over IP, X.25, Frame Relay, or Asynchronous Transfer Mode (ATM) networks. It uses UDP and a series of L2TP messages for tunnel maintenance. L2TP uses UDP to send L2TP-encapsulated PPP frames as the tunneled data. Figure 10 demonstrates how an L2TP packet is assembled prior to transmission. As in the case of PPTP, each layer or process adds its own header to the datagram generated by the application. The chief difference between L2TP and PPTP is that L2TP adds a UDP header to the packet and PPTP adds a GRE header to the packet.

3.1.3 IP Security (IPSec)

IPSec is an Internet Engineering Task Force (IETF) layer three protocol standard that supports the secured transfer of information across an IP network. As the name implies, it protects IP based services or applications by using cryptography. IPSec is expected to emerge as the preferred protocol for VPNs because it has a complete security solution, which includes packet authentication, encryption, and key management.

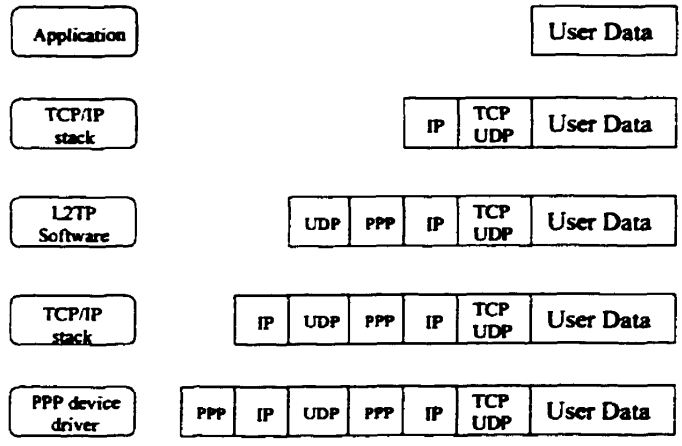

Figure 10. Construction of an L2TP Packet

To deliver security, the IPSec standard provides two security strategies: Authentication Header (AH) and Encapsulating Security Payload (ESP).

3.1.3.1 Authentication Header (AH)

The IP Authentication Header [12] provides connectionless integrity and data origin authentication for IP datagrams, and also offers protection against replay. Data Integrity is assured by the checksum generated by a message authentication code; data origin authentication is assured by including a secret shared key in the data to be authenticated; and replay protection is provided by use of a sequence number field within AH. The IPSec authentication header provides no data encryption.

Some fields in the IP header change en-route and the receiver cannot predict their value. These fields are called mutable and are not protected by the AH. The mutable IPv4 fields are:

- Type of Service (TOS),
- Flags,
- Fragment Offset,
- Time to Live (TTL), and
- Header Checksum

The payloads of IP packets are considered immutable and are always protected by the AH.

The position of the AH header in the IP packet and the header fields are shown in Figure 11. These fields are explained in Table 4.

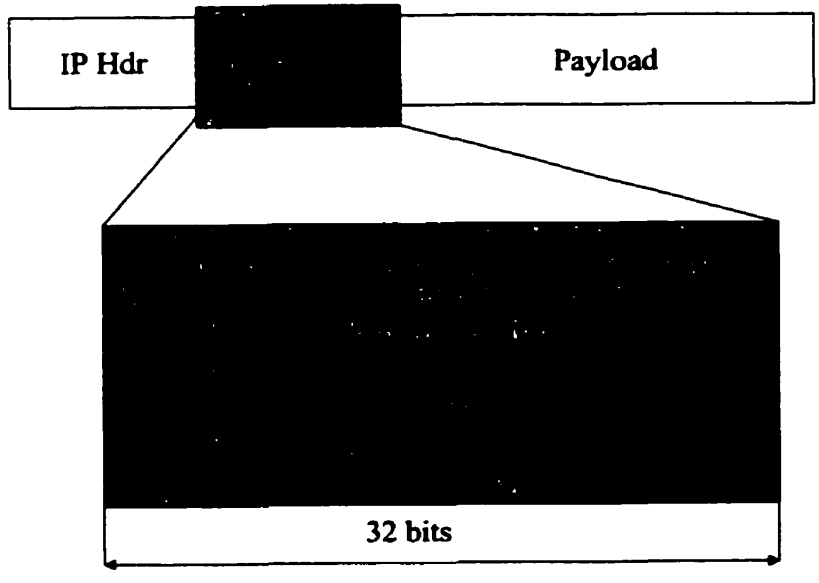

Figure 11. AH Header Format

Table 4. The explanation of fields in AH format

Field Name	Field Len. (bits)	Explanation
Next Header	8	The type of the next payload after the AH. The value is chosen from the set of IP protocol numbers.
Payload Length	8	The length of the AH header expressed in 32-bit word, minus 2. The default value is 4.
Reserved	16	Reserved for future use. It is currently set to zeroes.
Security Parameter Index	32	An arbitrary 32-bit number assigned to a security association (SA) (discussed later).
Sequence Number	32	Serves as a counter, which is used for replay protection. The sequence number has to be unique.
Authentication Data	Integral multiple of 32 bits	Used by the receiver to verify the integrity of the incoming packet.

3.1.3.2 Encapsulating Security Payload (ESP)

The ESP [13] header provides a mechanism to encrypt the IP payload. In doing so, it provides confidentiality, data origin authentication, connectionless integrity, and anti-replay protection. Therefore ESP headers are an alternative to AH headers in IPSec packets.

The format of the ESP packet is more complicated than that of the AH packet. There is not only an ESP header, but also an ESP trailer and ESP authentication data. The payload is located between the header and the trailer. Figure 12 shows the structure of the ESP header and trailer. The security parameter index (SPI) and the sequence number fields are the same as those in AH. The explanations of the remaining fields are listed in Table 5.

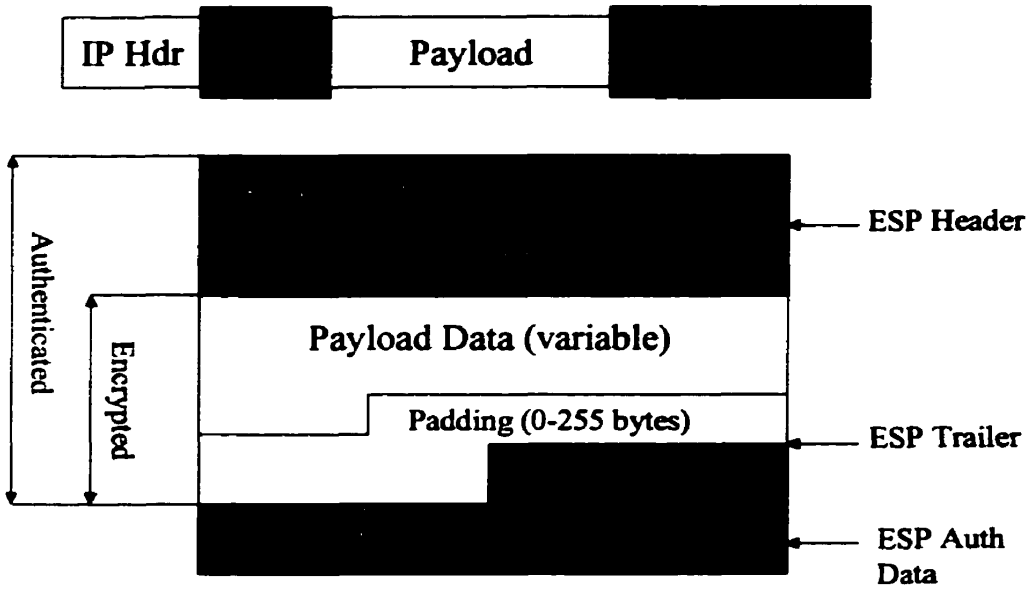

Figure 12. ESP Header and Trailer

Table 5. Explanation of fields in ESP

Payload Data	A data of variable length that consists of data described by the Next Header field
Padding	Used to make the input data an integral number of blocks
Pad Length	An 8-bit field that contains the number of padding in bytes.
Next Header	An 8-bit field that shows the data type carried in the payload.
Authentication Data	A variable length field which is calculated from the SPI to the Next Header field.

Comparing ESP to AH, only ESP provides encryption, while either can provide authentication, integrity checking, and replay protection.

3.3.1.3 Security Association (SA)

A security association (SA) [14][15] is a data structure that contains information about which transformation is to be applied to an IP datagram. An SPI, as well as an IP destination address and a security protocol identifier (AH or ESP) uniquely identify each SA.

Two types of SA can be used: transport mode SA and tunnel mode SA. Figure 13 illustrates the AH and ESP in transport mode. As shown in the figure, the IP header for the transformed packet is the original packet's IP header. Therefore, transport mode only provides protection for upper layer protocols, but not for the IP header. In the case of AH, protection is provided to the IP header and the IP payload. In the case of ESP protection, the original packet except the IP header is encrypted.

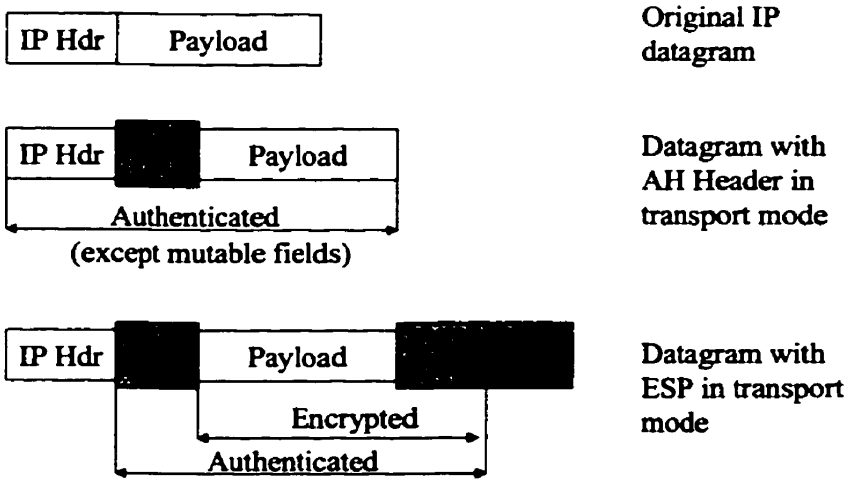

Figure 13. AH and ESP in Transport Mode

Tunnel Mode protects the entire inner IP packet, including the IP header. Figure 14 shows AH and ESP in tunnel mode. For AH format, the entire original packet is appended after a new IP header. The original IP header carries the ultimate source and destination addresses. For ESP, the entire original IP datagram is enclosed within the ESP payload. A new IP header is generated and attached to the ESP payload. Upon receipt of the datagram, a tunnel server processes and discards the plain text IP header and then decrypts its contents to retrieve the original payload IP packet. The payload IP packet is then processed normally and routed to its destination on the target network.

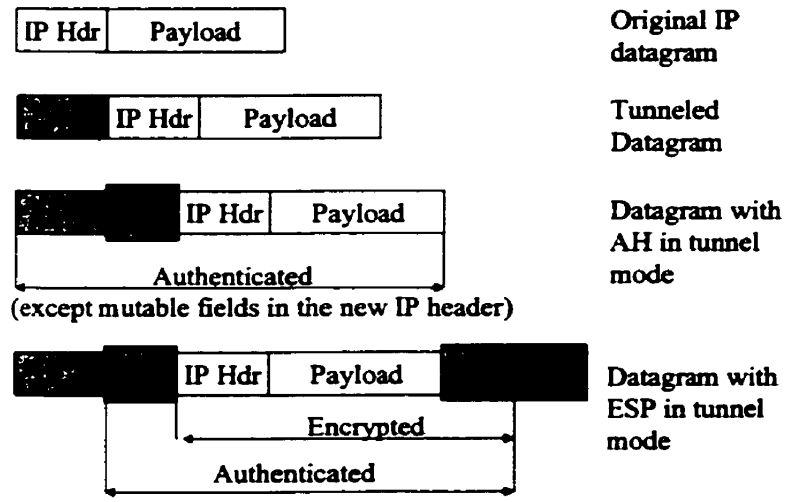

Figure 14. AH and ESP in Tunnel Mode

Either AH or ESP can be specified in an SA, but not both. If both AH and ESP protection is needed, then two or more SAs must be created. To secure a bi-directional

communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

3.1.3.4 ISAKMP/Oakley (IKE)

The Internet key exchange (IKE) [16] combines the Internet Security Association and Key Management Protocol (ISAKMP) with the Oakley key exchange. It is used to establish, negotiate, modify and delete SAs. ISAKMP requires that all information exchanges must be both encrypted and authenticated. No one can eavesdrop on the key material, and the keying material will be exchanged only among authenticated parties.

ISAKMP/Oakley uses a two-phase approach. The purpose of phase 1 is to establish a master secret key from which all cryptographic keys will subsequently be derived for protecting the users' data traffic. It establishes an SA for the ISAKMP itself. Phase 2 is used to establish an SA for IPSec. Phase 2 negotiations generally occur more frequently than Phase 1 since the master key can be used for a longer time.

3.2 Security

VPNs primarily protect data as it tunnels through the Internet. This process, though very effective with information in transit, does not offer protection to an organization's network itself. As in the case of private networks, a firewall offers the best potential for VPN security. Firewalls achieve network protection and privacy through access control. Access control specifies the amount of freedom a VPN user has, restricts certain types of traffic, and controls access to applications in various network domains. Since almost every organization connected to the Internet has a firewall installed, all that is needed is to add VPN software to the firewall. The Internet firewall is also the ideal location for deploying

World Wide Web and FTP servers. The firewall can be configured to allow Internet access only to those services, while prohibiting external access to other systems on the protected network.

When two sites, each behind different corporate firewalls, want to communicate with each other via a VPN, the VPN software on each firewall creates keys that will be used to encrypt messages. When an application being used by the user generates a packet, the packet leaves for the destination via the firewall. The VPN software, which resides in the firewall, encrypts the packet according to the protocol (AH or ESP) then the firewall forwards it to its destination. The firewall/VPN at the other end receives the packet, decrypts it and then forwards it to the final destination, which is located in the local network. The responding packet generated by the receiving end goes through the same procedure in the opposite direction.

3.3 VPN Management Requirements

VPNs offer enterprises an alternative approach to interconnecting employees, remote facilities, and external business partners. Though it extends a private network to the Internet, a VPN is still an end-to-end network that exhibits most of the same characteristics as private WAN infrastructures. As a business expands, the network grows. A VPN must remain scalable to address the increased needs. Moving from a dedicated infrastructure to the Internet challenges network administrators to maintain the confidentiality and integrity of the data while opening the network to Internet accesses. The move to a shared infrastructure (the Internet) also presents new challenges in delivering and monitoring the reliability of the network service provided.

To ensure smooth integration of private WANs and VPNs, network managers need to exchange configuration information with the service provider. This includes validating that the service levels provided by the service provider meet the corporation network requirements. To provide scalable network-wide management, the network administrators can use policy-based network management to configure the network instead of configuring the network on a device-by-device basis.

Since VPNs are an extension to enterprise networks, the existing enterprise management environments can be extended and enhanced with VPN management capabilities to provide network administrators with control over the VPN end users. As VPNs gain popularity, the need for open, standard, replicated, and integrated management services becomes essential. Using directory-based policy management will enable integrated user information and network states. By leveraging the DEN standard, the directory will become the repository for the critical information required for interoperable, policy-based networking including VPNs.

4. MODELING A SIMPLE NETWORK MANAGEMENT SYSTEM

A program was written in Java to model a simple network and its management system. The program demonstrates the use of DENs in network management. A CIM is used to implement some of the network and network management components. XML is used as the communication vehicle between devices. Network sockets are used to enable objects running in different virtual machines to communicate with each other.

4.0 Java

To demonstrate the use of a CIM in a network application, an application [17] [18] was developed in Java which utilizes Java Remote Method Invocation (RMI). RMI creates Java applications that can talk to other Java applications over a network. RMI allows an application to call methods and access variables inside another application, which may be running in different Java environments or different operating system altogether, and to pass objects back and forth over a network connection.

4.0.1 Advantages of Java as a programming language

As a programming language, Java has a number of advantages:

- Platform independence— A Java program can run on any platform that has a Java virtual machine (JVM) because Java programs are compiled into bytecodes, which in turn are interpreted by the interpreter in a JVM.

- High productivity—Java was designed to incorporate the latest object-oriented software engineering techniques. Further, it does not have constructs such as multiple inheritance and operator overloading that are prone to usage mistakes.
- Built-in network support—Java's built-in network library package (java.net) provides cross-platform support for simple networking operations, including connecting and retrieving files by using common Web protocols and creating basic UNIX like sockets.
- Support for international characters—Java uses Unicode as the character set. It is especially important in today's global economy.

4.0.2 Java RMI

The goal for RMI is to integrate a distributed object model into Java without disrupting the language or the existing object model, and to make interacting with a remote object as easy as interacting with a local one. It also includes additional exceptions for handling network errors that may occur while a remote operation is occurring. RMI is protocol independent.

The Stub and Skeleton classes, which reside on the client and server respectively, hide the remoteness of the method call from the application implementation classes. When a server application registers an object with the RMI Registry, the skeleton class is also registered. When a client application or applet requests a server object, an instance of the stub class is created and connected to the server application. Stubs and skeletons allow the client and server classes to behave as if the objects they are dealing with were local.

RMI is simple to use because most of the complexity is hidden in the stub and skeleton classes. A normal Java object can be transformed into an RMI server object by applying a few simple modifications and generating stubs. The client invokes methods on the server object through a Java interface, so there are even fewer modifications required there.

Although RMI is easy-to-use and simple-to-implement, it is a Java only solution. RMI can not be used unless all communicating programs are written in Java.

4.1 The IBM XML parser

Using XML as a standard message format in business to business transactions requires a mechanism that can read and interpret the XML documents. An XML parser can be used for this purpose. It reads the document and re-arranges it in a structured form that allows each element to be accessed and manipulated.

IBM's XMLJ4 parser is written in Java, and is therefore portable to any operating systems with a JVM. The parser uses both DTD and the XML documents to create a Document Object Model (DOM) tree, which presents the document hierarchically. The DOM has a group of API's, that allow easy access to the elements within the tree. Using the DOM APIs, any element within the XML document can be accessed, changed, or added.

The package "com.ibm.xml.parser", which is primarily used in the project, contains classes and methods for parsing, generating, manipulating, and validating XML documents.

4.2 Converting MOF files to JAVA

As mentioned earlier (Section 2.2), a CIM model is described in a language called Managed Object Format (MOF). The MOF file is basically made up of a series of class and instance declarations. Components of a MOF file include:

- Keywords,
- Comments,
- Validation Context,
- Naming of schema elements,
- Class declarations,
- Qualifier declarations,
- Instance declarations,
- Method declarations,
- Compiler directives, and
- Initializers.

Figure 6 illustrated an example of expressing a CIM class in MOF. The following specifies a class:

1. **The qualifiers of the class.** This may be empty or contain a list of qualifier name/value pairs separated by commas and enclosed within square brackets ("[" and "]").
2. **The class name.**
3. **The name of the class from which this class is derived** (if any).
4. **The class properties,** which define the data members of the class. A property may also have an optional qualifier list, expressed in the same way as the class

qualifier list. In addition, a property has a data type and (optionally) a default initial value.

5. **The methods** supported by the class. A method may have an optional qualifier list. A method has a signature consisting of its return type, plus its parameters and their type.

The most common qualifiers used are listed in Table 6 [9].

Table 6. The most common qualifiers

ALIAS	Establishes an alternate name for a property or method in the schema.
ARRAYTYPE	Indicates the type of the qualified array. Valid values are "Bag", "Indexed", and "Ordered".
DESCRIPTION	Provides a description of a Named Element.
KEY	Indicates that the property is part of the key. If more than one property has the KEY qualifier, then all such properties collectively form the key (i.e. a compound key).
MAPPINGSTRINGS	A mapping string for a given provider or agent.
MAX	Indicates the maximum number of values a given multi-valued reference can have. A value of NULL implies no limit.
MAXLEN	Indicates the maximum length, in octets, of a string property. When overriding the default value, any unsigned integer value (uint32) can be specified. A value of NULL implies unlimited length.
MIN	Indicates the minimum number of values a given reference can have.
MODEL CORRESPONDENCE	Indicates a correspondence between an object's property and other properties in the CIM Schema. Object properties are identified using the syntax: *<schema name> "_" <class or association name> "." <property name>*
OVERRIDE	Indicates that the property, method, or reference in the derived class overrides the similar construct in the parent class in the inheritance tree or in the specified parent class.
PROPAGATED	The propagated qualifier is a string-valued qualifier that contains the name of the key that is being propagated. READ Indicates that the property is readable.

REQUIRED	Indicates that a non-NULL value is required for the property.
UNITS	Provides units in which the associated property is expressed. For example, a Size property might have Units ("bytes").
VALUEMAP	Defines the set of permissible values for this property. For example: [ValueMap ("1", "2", "3", "4", "5") , Values ("Other", "Unknown", "Enabled", "Disabled", "Not Applicable")] uint16 StatusInfo;
VALUES	Provides translation between an integer value and an associated string. If a ValueMap qualifier is not present, the Values array is indexed (zero relative) using the value in the associated property. If a ValueMap qualifier is present, the Values index is defined by the location of the property value in the ValueMap.

One qualifier that needs to be stressed is KEY. Keys in the CIM are used to provide a way of uniquely identifying an instance within the scope of a given namespace. Only properties and references may be used as keys, and keys are unique within a namespace. If a new subclass is derived from a superclass and the superclass or any of its parent classes have key properties, the new subclass cannot define any additional key properties. New key properties in a subclass can be introduced only if all of the superclasses of a given class have no keys. A class can have more than one key. Those keys consist of a composite key, which enables any instance of any subclass of the class to be identified.

Since the CIM is expressed in MOF files, a program was developed to convert MOF files to Java program files. To take advantages of XML, the conversion is done in two steps. The first step involves converting MOF files to XML files. A program—*ProcessMOF.java* was written to accomplish this task. The program takes one string argument, i.e. the name of the MOF file being processed. The conversion is done according to the XML DTDv2.0.0, which is posted on the DMTF's web based enterprise

management (WBEM) standards web site (http://www.dmtf.org/spec/wbem.html). Each class of the CIM has it own XML file that means that for every MOF file, there could be many XML files. The second step is to convert XML files to Java Program files. The program *ProcessXML.java* was developed for this purpose. It takes the name of a XML file as input. The output of *ProcessXML.java* is a series of Java program files, which can be used as building blocks to develop network applications.

To convert a MOF file to Java program files, first *ProcessMOF.java* is used to convert the MOF files to XML files. Once the conversion is done, another program—ProcessXML.java can be used to produce Java program files using the outputs of the first step as its input.

4.3 Modeling a simple network management system

A management program developed by Dr. D. Blight at Fujitsu is used as the basis for the project. Figure 15 shows the high level structure interpretation of the management system.

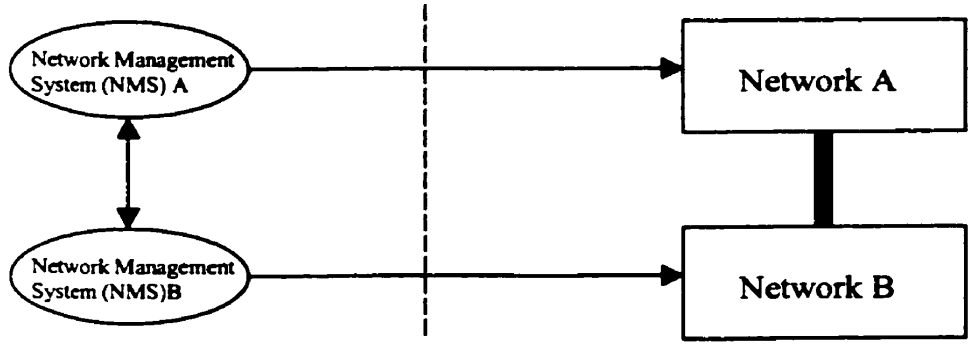

Figure 15. Graphic Representation of the Network Management Program

Network management system (NMS) A manages Network A and NMS B manages Network B. The two networks can be connected or disconnected. If the two network are connected and A needs to send some data to B, NMS A wants to know the state of Network B. To accomplish this, NMS A needs to communicate with NMS B. The information exchanged between the two network management systems can be the policies or state deciding which policy to use . To exchange the information effectively, a standard format should be used. Figure 16 illustrates a possible solution where CIM is used to describe the information. XML is used as the encoding schema because of the benefits described in Section 2.3. A network administrator can access the information through web browsers using HTTP or through text editors depending on his/her preference and the location of the files.

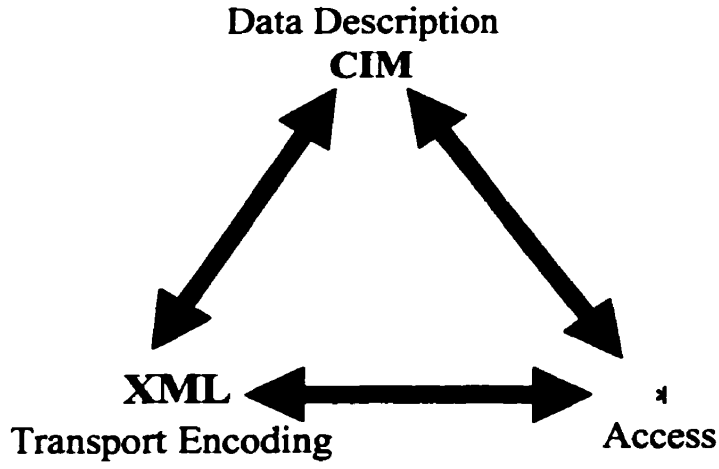

Figure 16. Communications between NMSs, Networks

4.3.1 Modeling a simple network

To fully model a network, we need to implement network objects, such as routers, connections, applications, application systems and policies. A class—NetObject was used as the base class for all the network components. A *RemoteObject* class was developed to identify objects with their hostnames, object names, and its *NetObject* names. The physical aspects of a network are modeled by classes: *Connection*, *Router*, *Path*, *Network*, *Device* and *Port*. The full class hierarchy is shown in Figure 17.

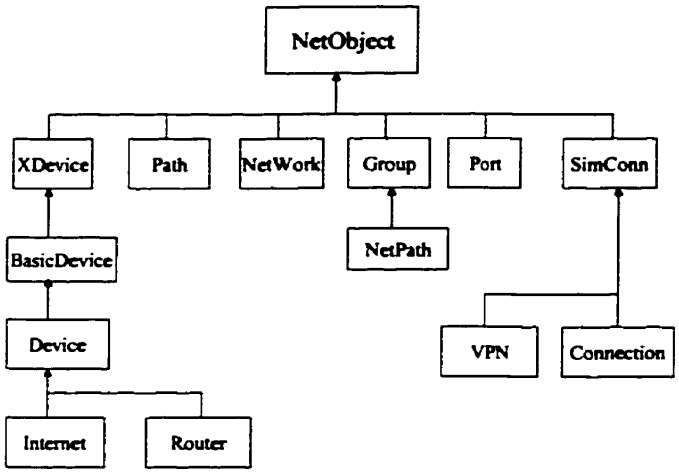

Figure 17. Physical Class Hierarchy of Implemented Network Components

The *NetObject* class defines a set of static variables that are used to identify different components of a network, which are modeled in the simple network. The values of these static variables are listed in Appendix B. The class defines the following attributes:

- "Name" is a string that is the key of this class. This attribute is used to distinguish different objects

- "State" is an enumerated integer that defines the status of the network object. There are eighteen possible values, which are listed in Appendix B
- "Description" is a string that provides a textual description of the object. Each and every object has its own description
- "InstallDate" is a Date value indicating when the object was installed. It can be updated whenever a network manager updates a system
- "Caption" is a string that provides a short textual description (one-line string) of the object

The *NetObject* class is modeled after the CIM's ManagedSystemElement class. Because the project is not intended to model a complex network system, the devices that are modeled in the program are generic. The CIM_PhysicalElement is skipped since it is mainly concerned with manufacturing information, such as manufacturer, stock number, serial number, part number and version. Since CIM_LogicalElement is an empty class whose sole purpose is to separate logic elements from physical elements, it is not adopted in this project.

The class *Xdevice* is a subclass of the class *NetObject*, and it defines methods: neighbors(), Connections(), addConnection() and deleteConnection(). All these methods will be used in the implementation of a simple network. In the *BasicDevice* class, the methods: neighbors() and Connections() are overridden. The class *Device* is a subclass of the *BasicDevice* class; it can be used to represent physical components such as routers, switches, and terminals. The difference between a *Router* object and a *Device* object is that a router object has a loopback connection, whereas a device object does not. A much

simplified version of the CIM hierarchy is used because of the complexity involved in modeling a whole CIM.

The class *Path* is a subclass of the class *NetObject*, and it represents connections between different network devices. It is composed of path segments (Objects of PathSeg, which is a class that defines attributes: Device, Application, Connection, Port and Queue). The transmission rate of the path can be set by user to simulate network under different conditions.

The *Network* class defines methods that establish a network. The network includes devices, paths, applications, and routing protocols. The queues that network devices use are implemented as well in the *Network* class. This class also includes methods that can be used to distinguish data source and data sink. All the devices in the network are stored in a vector. Any device in the network can be found by using the method: findDevice(String name). As indicated in the above, all the instances of *NetObject* has name as one of its attributes. Different instances should have different value for name.

In the *Group* class, a vector is used to store the instances of class *NetObject*. The group then can be treated as a whole during the simulation. The class *NetPath* implements methods that can associate devices and connections. The class *NetPath* is the subclass of the *Group* class, and thus it inherits all the methods in the *Group* class.

The local IP address, local port, foreign IP address and foreign port identify the two endpoints of a connection. The class *Port* implements the methods that realize the functionality provided by ports. It includes the method findQueue(String QN) which takes

a string as an input and finds the corresponding queue, selectQueue(Device dst, IPv4Header IP) which selects a queue according to the device and its IP address which is implemented by the class IPv4Header, and removeQueue() which removes contents in a queue. Since in this simple network all the instances of the class *NetObject* have distinct names, ports, connections, and devices can also be identified by their names. The data flow rate can also be calculated using the information contained in the queue. The method calcFlow() is implemented for this purpose.

The class *SimConn* implements the basic functions of a connection. It identifies the two ends of a connection: the source device and the destination device. It also contains method—isVPN() that returns a boolean, which distinguishes VPN connection from other connections. A VPN connection is between two instances of *NetObject*. The class Connection is used to implement general-purpose network connections.

The logical aspects of a network are described by the classes: *Application*, *ApplicationSystem* and *Policies*. These classes are all subclass of *NetObject*. The class *Application* and *ApplicationSystem* are classes used to model applications and the system they reside. An application may be in one of four states: deployable, installable, executable and running. To simplify the model, only the later two stages were implemented in this thesis. Applications are run assuming a client-server model as most of the applications used in networking are under it. The *ApplicationSystem* resides in the host and the Application resides in the *ApplicationSystem*. The instances of *ApplicationSystem* and *Application* can be identified by their name since they are all subclass of *NetObject*.

The package Policy contains classes that can be used to implement policies for connections, paths, QoS, and VPNs. Since the main purpose of this thesis is to demonstrate the use of XML in network management and the benefits of a CIM, not all the policies are fully implemented. The classes are however, written so that they can be extended at a later time. The base class of this package is *flaPolicy*, which is a subclass of *NetObject*. An important method in *flaPolicy* is activate(Network N), which activates a policy in a given network.

There are several supporting packages, which implement all the other necessary functions a network supports. These include IP, server, and RemoteReference. There are two classes in the IP package: *IPv4Address* and *IPv4Header*. IPv4 addresses are expressed as four decimal numbers separated by decimal points. Each decimal number represents one of the 4 bytes of the 32-bit address. The class *IPv4Header* includes fields: source address, destination address, source port, destination port and Type of Service (ToS). All the other fields are ignored in the model. The package RemoteReference contains class—RemoteReference, which links host and object, host and remote host. Objects can be anything, such as an application, a connection, or an application system.

The class TNOM which is a server side application utilizing RMI. To implement an RMI-based client/server application, an interface, which contains all the methods the remote objects support, must be defined. *ManagedObject* is such an interface. All the methods in this interface potentially throw *RemoteException* to handle potential network problems. The class *TNOM* implements the remote interface *ManagedObject* and extends the class *UnicastRemoteObject*. The methods contained in the interface are implemented

inside the TNOM class. The remote application is also registered, which binds it to a host and port.

To complete the network, a client side application—*Sdemo* is used. The class takes one argument, which is a file name, from the input line. The input file, which is written in XML, contains the configuration of a particular network. The simulated network then configures itself according to the input. Thus, the configuration of a network can be easily changed to model different situations. Since the configuration files are written in XML format, the program can search for a particular input if necessary. All the configuration files are contained in a well known directory which enables programs running on different machines to locate desired files easily.

An example of the graphical user interface of the program is given in Figure 18. The configuration file for the output shown in the figure is listed in Appendix C. The DTD for the configuration file is listed in Appendix D.

The example network consists of four gateway routers, each of which has distinct name. The IP address for a router can either be auto-assigned or specified by network administrators. The routers are connected to five different datasources. The bandwidth of each connection is initially set at 1500 bps, which can be changed during a simulation. The connections are arranged such that the routers can still perform routing functions even if some connections are down. This scenario can be simulated by disabling selected connections. The center of the network consists of two interconnected Internets. A user can customize the network configuration by clicking on the buttons located at the bottom of the window (see Figure 18). Each component of the network has its own configuration panel.

They are invoked by double clicking on the specific component. Users might need to change some parameters to meet their specific needs. Each panel shows changeable parameters, such as the IP address of a device. The process of configuring a network and the simulation are shown in a separate window so that a user can monitor the progress.

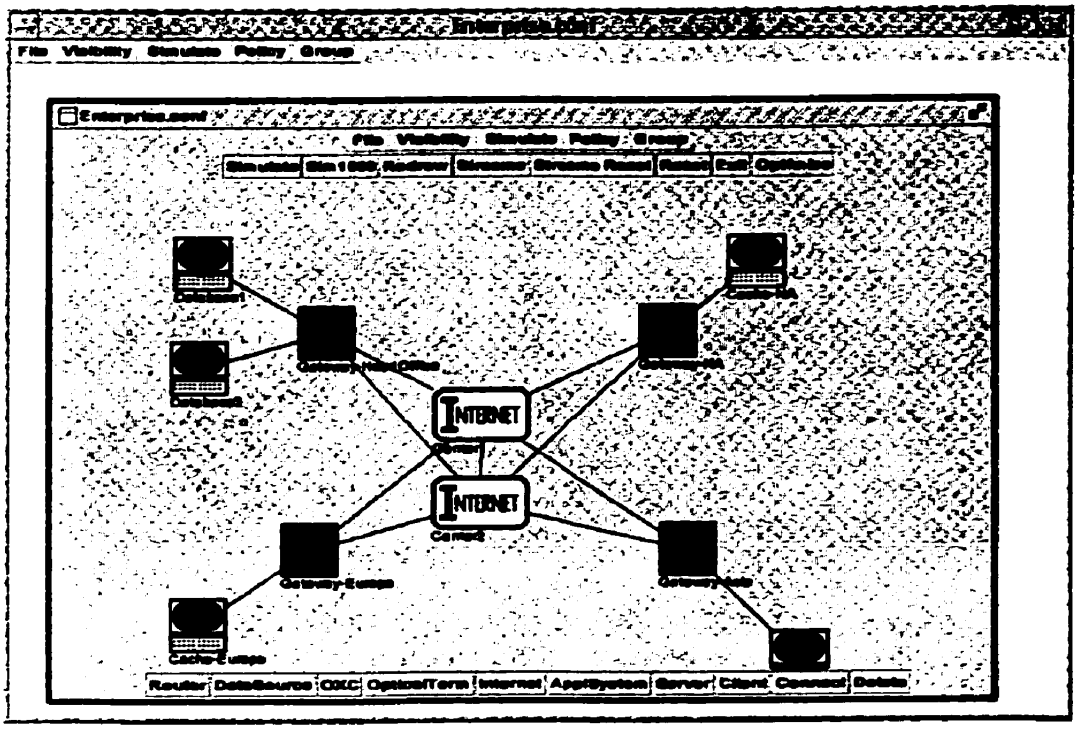

Figure 18. Graphic Interface of the Program

4.3.2 Modeling a management system

To communicate, we need the information to be exchanged and an encoding scheme. As shown in Figure 16, the data is described using CIM, which is written in MOF. The ideal choice of encoding scheme is XML because it is system-independent.

The other benefits of XML are that new tag and attribute names can be defined at will and document structures can be nested to any level of complexity. The program *ProcessMOF.java* (described in Section 4.2) was implemented and incorporated into the system to convert MOF file to XML files.

A policy server was added to the above program (Section 4.3.1), which listens to requests from the users. A user invokes a request for a policy by clicking on the Policy menu (shown in Figure 18). Once the server receives the request, it generates a MOF file according to the status of the network. The generated file is then converted to an XML file, which is sent to the requester.

A VPN policy file that instructs the network to realize a VPN connection using IPSec is used in this program. Several classes are added to the package pbn.policy, which contains classes that specifies policies for different purposes. The class *IPSecPolicyList* uses a string vector to store a set of policy names that are contained in the system. Methods that can be used to access, add, and delete a particular policy are implemented in this class. The class *VPNpolicy* is used to send a request for a VPN connection to a policy server. The class is used as a network management system. It tells the network of the existence of VPN connections and security associations. It also identifies whether encryption should be used in a VPN connection or not. The class also includes methods can be used to generate keys for encryption. The class *Server* is used as the policy server. It listens to the users' requests and sends the requested policy files to the users. Since XML is just an open way to express data, XML can be transported with any protocol. Network sockets are used to transfer information between user and policy server in this program.

A flag was added to the class *Network*, which indicates the existence of a VPN. If it is true, a VPN is used somewhere in the network. A vector—*SA*, which is used to store all the security associations in the network, was also added to the *Network* class. A security association is unidirectional. Two security associations are needed for two parties to communicate. All the security associations could be stored in a separate file so that other management systems could have access to the file.

Class *VPNtunnelPolicy* is used to indicate the existence of a VPN between two end-points. After receiving the policy file, the user creates a new instance of the class *VPNtunnelPolicy*. It checks to see if a connection between two endpoints of the VPN exists. If the connection exists, it sets the flag "*isVPN*" in the class *Connection*. Otherwise, it creates a new connection between the two endpoints. It then adds the VPN connection to the vector *SA* in the class *Network*. A network manager can keep track of VPN connections by checking this vector. If the protocol ESP is used, the class *VPNpolicy* generates keys which can be distributed to the communicating parties.

4.3.3 Using the program

While running the program, a user invokes a VPN policy request by clicking on the menu item VPNPolicy, which in turn creates a new instance of the class *VPNpolicy*. The method—configure(Network N) in the class *VPNpolicy* is called. This method creates a network socket connection to the policy server and sends a VPN policy request to the server. The server generates a simple policy file which includes the two endpoints of the VPN connection, the protocol and the transport mode to be used in the VPN connection. The server then converts the MOF file to XML and sends it back to the user.

After receiving the policy file, the user creates a new instance of the class *VPNtunnelPolicy* using the two endpoints specified by the policy file. It first checks to see if the connection between the two points exists. If it does, it sets the flag *isVPN* in the class *Connection* to true. Otherwise, it creates a new connection between the two endpoints and then sets the flag. The class Connection uses vector *sas* to store the two security associations (one for each direction).

The flag *VPN_EXIST* in the class *Network* is set to true to indicate the existence of the VPN connection in the network. The class *Network* uses vector *SA* to store all the VPN connections in the network. A network manager can keep track of the VPNs by checking this vector. To simplify the management, all the VPN connections are output to a text file, which can be accessed by other network management systems if necessary.

Figure 19 demonstrates the program's process.

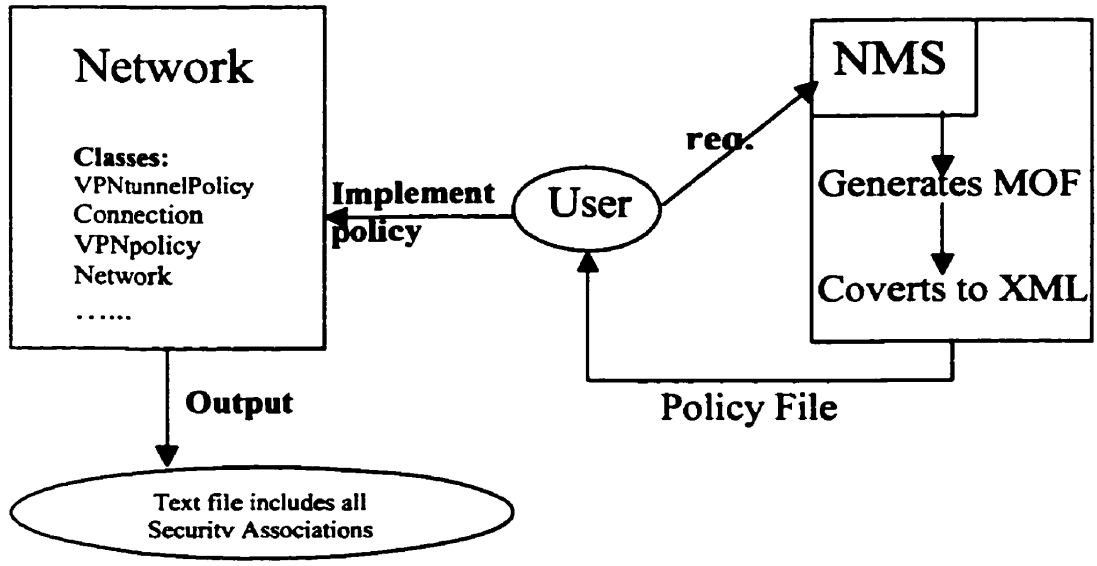

Figure 19. Graphical representation of the NMS management program

4.4 Modeling IPSec at the Packet Level

Because IPSec is expected to emerge as the dominant protocol for VPNs due to its encryption ability, I modeled it at the packet level. The Diffie-Hellman algorithm is used in the IKE phase. The algorithm DES in ECB mode is used to encrypt and decrypt the packets. HMAC-MD5-96 [19][20] is used to authenticate data origin and to ensure packet integrity.

The Diffie-Hellman key exchange protocol allows two parties to agree on a shared key, even though the messages are transferred through public media. Since there is no secure channel in the earliest key negotiation session, it can be used to negotiate shared secret keys. Figure 20 shows the steps involved in establishing shared secret keys. These secret keys will be used in the next steps of the key negotiation protocol to derive keys that will be used in DES. In the Diffie-Hellman key exchange, both parties share two public values, a modulus m, which is a large prime number, and an integer g. Each party has a private number (a and b respectively) that should be large.

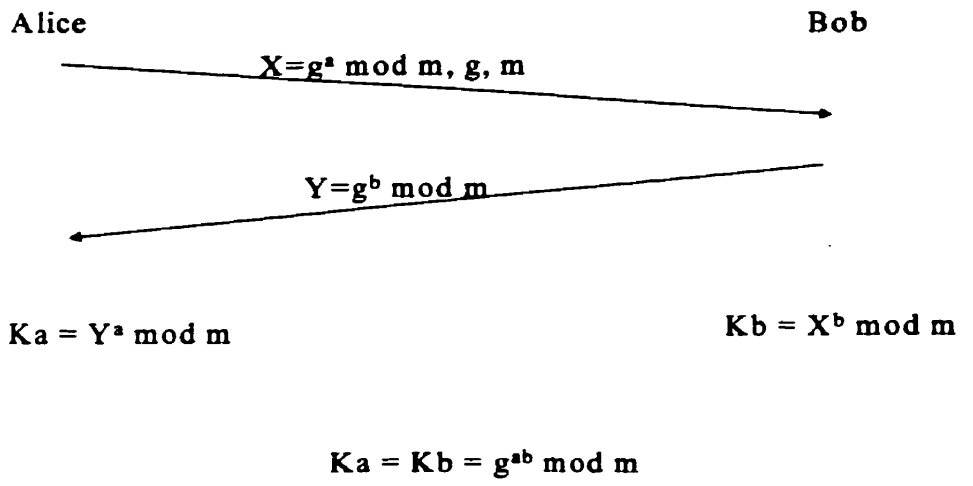

Figure 20. Diffie-Hellman key exchange

DES [21] in ECB mode is used in ESP to encrypt and decrypt IP packets. The DES encryption process, which consists of three steps, is shown in Figure 21. The input is 64-bit plaintext and the key is 56 bits in length. First, the input goes through a permutation. The output is then fed to 16 iterations, each with a different key. The right and left halves of the output of the last iteration are swapped before going through an inverse permutation function. In ECB mode, each block of ciphertext is encrypted independent of every other block.

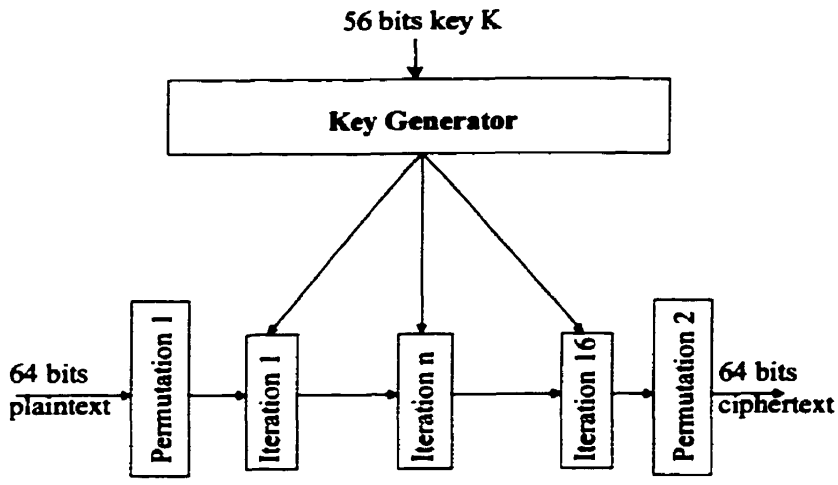

Figure 21. The process of DES encryption

HMAC-MD5-96 is used as the authentication mechanism to provide data origin authentication and integrity protection. The HMAC-MD5-96 process is shown in Figure 22. The base function is applied twice in succession. The leftmost 96 bits of the resulting hash value are used as the MAC for the datagram, which is stored in the authentication data field of AH (Figure 11) and ESP (Figure 12).

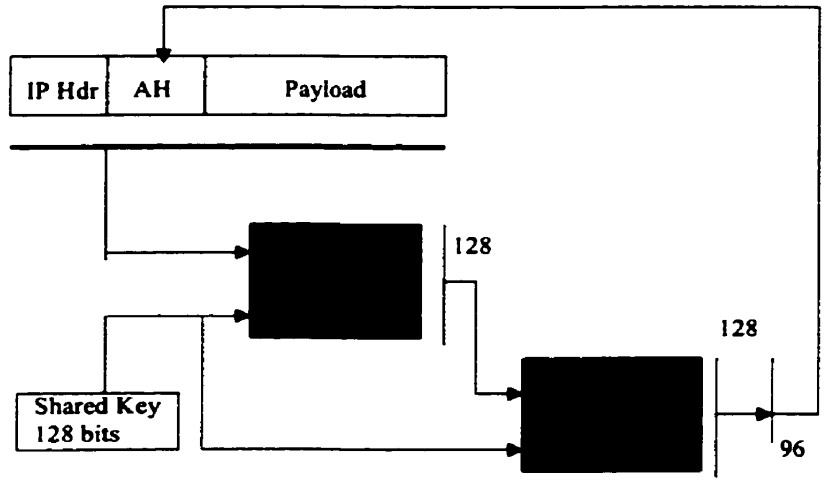

Figure 21. HMAC-MD5-96 Processing

4.4.1 Implementing in the management system

There were several classes added to the program described in Section 4.3 for handling encryption and VPNs. the package Key contains three classes that implement IKE key exchange, encryption key generation, and authentication. Because only Diffie-Hellman key exchange is implemented for this thesis, the class *DHKeyGen* is the only subclass of the class *IKE*. Other key exchange algorithms can be added by extending the class IKE. The class *DES* and the class *MAC* are used for DES and MAC generation.

Three main components of the protocol IPSec are contained in the package VPN. The class *AH_Header* realizes the AH header format shown in Figure 12. Next_Hdr and Payld_Len are bytes; Reserved is a short; SPI and sequence number are integers. The MAC is a byte array that could use a different length for a different algorithm. The Reserved field

is set to zeroes and it is included in the data authentication calculation. Because a 96 bits authentication value is used, Payld_Len is set to 4. The implementation of ESP is more complex than that of AH_Header. Three classes *ESP_Header*, *ESP_Trailer* and *ESP_Auth* are used to implement header, trailer and authentication data (shown in Figure 12), respectively. The class *ESP* has four instance variables: header, payload, trailer and authentication. The class *SA* uses properties: SPI, protocol name, and destination IP address to identify unique security associations used in IPSec.

Package pbn.VPN.Action contains the classes *ahAction* and *espAction*, which are subclasses of the class *vpnAction*. AH can be employed in two ways: transport mode and tunnel mode. In transport mode, AH is inserted after the IP header and before the upper layer protocol. When AH is used in a security gateway such as firewall, tunnel mode must be used. The method *generateAHPacket* in the class *ahAction* handles the generation of AH packets. The method takes one argument—payload, it then checks the mode and the destination. It uses a security gateway (a default router) for tunnel mode. The inner IP header contains the ultimate source and destination address, while the outer IP header uses the address of the security gateway as its destination address. A SA is associated with each session of the AH process. The sequence number is set to zero when a SA is first established. The first packet sent using the given SA will have a sequence number of 1. The class *espAction* handles the process of ESP. The method *generateESP* takes one string argument and generates the ESP header and trailer. The string is padded to ensure that the resulting ciphertext terminates on a 4-byte boundary so that the Authentication Data field is aligned on a 4-byte boundary, as illustrated in Figure 14. For illustrative purpose, a random byte is used for property Next_Hdr to identify the type of data contained in the Payload

Data field. Like AH, ESP can be employed in two ways: tunnel mode and transport mode. The transport mode is only applicable to host implementations. The ESP is placed after the IP header, but before the upper layer protocol information. Thus it provides protection to the upper layer protocol, but not to the IP header. The padded payload and ESP trailer are encrypted before being added to the final ESP packet. Authentication is performed on the ESP header and the encrypted payload and ESP trailer. Tunnel mode ESP may be employed in either hosts or security gateways. The program uses a default router for the security gateway. If tunnel mode is used, the method *generateESP* adds the original IP header to the padded payload. The resulting string and ESP trailer are encrypted. The method *composeESPPacket* adds an IP header to the output of *generateESP*. If tunnel mode is used, the IP header has the router as its destination. If transport mode is used, the IP header uses the original destination as its destination address. The method *keyExchange* is used to generate keys that will be used for encryption and decryption. The generation of sequence numbers is the same as in the case of AH.

Several classes were also added to the package pbn.policy. The class *IPSecPolicy* includes the name of the algorithm that will be used to exchange keys during the IKE negotiation and the specifics of IPSec, such as the protocol (AH or ESP) and the transport mode (Transport or Tunnel). It contains one flag, which is set whenever the current policy is enabled. Since I only used Diffie-Hellman key exchange, the name of the algorithm is set to DH (for Diffie-Hellman) by default. The class *VPNpolicy* is used to send a request for a VPN policy to the server and create a new instance of the class *vpnPolicyAction* after receiving the policy file. The method *executeAction* in the class *vpnPolicyAction* creates an SA for the session and it uses AH or ESP to transmit packets according to the policy file.

The class *Device* is modified to add encryption and decryption capabilities to the hosts and routers. For IPSec in tunnel mode, a router checks the MAC field to see if the packet is valid. If it is not, it drops the packet without further action. If AH is used, it sends the packet to the final destination directly. It decrypts the packet, then sends it to the intended destination if ESP is used. For IPSec in transport mode, the host checks the MAC field and performs necessary functions before sending it to the upper layer.

A server is used to listen to the request for the VPN policy file from users. The class *Server* extends the class *Thread*. A server object can accept requests from different users. When the server receives a request for a VPN policy file from a user, it checks to see if it is available. If the requested file is available, the server pulls the policy file, which is written in MOF format, from a directory, converts it to a file in XML, and sends it to the requester. The directory that contains the policy files and server can be located on different hosts. A user invokes a VPN solution by clicking on the VPNPolicy menu item. The class *VPNPolicy* reads the XML file and executes the VPN solution according to the specific policy. The server and the requester communicate via network sockets in this model.

The program was run to see the time it takes to generate Diffie-Hellman key pairs. Two users exchange public information (g, m) first. Each user then computes a shared secret value based on the public information (g, m) and their own secret information (a, b) (see Figure 20). These secret values can be used as session keys or as encryption keys for encrypting randomly generated session keys. The graphic interface of the program is shown in Figure 18. A string "This is a test for VPN protocol." was used in the simulation. The simulation was run several times. Ten randomly chosen results are shown

in Table 7. The average time for the key exchange is 41052 ms. Depending on the server's condition, it could take a lot more than 41052 ms to generate Diffie-Hellman key pairs. Thus, it makes sense to use a single Diffie-Hellman key for a long period of time. Unlike DES key pairs, which change for different session, Diffie-Hellman key pairs usually valid for a day, so they will not overly burden the server for key generation.

Table 7. Results for simulation of Diffie-Hellman key exchange

Time used to generate Diffie-Hellman key pairs (in ms)									
40700	38780	37960	38330	39000	37790	41030	38500	55310	43120

To demonstrate that DES takes less time to generate keys, I also collected data for DES key generation. The results are shown in Table 8. The average time for a client to generate a DES key is 368 ms, which is less than one percent of that of Diffie-Hellman key exchange. To maximize the security of data transmission, it is feasible to generate different DES keys for different sessions to provide enhanced data integrity.

Table 8. Results for simulation of DES key generation

Time used to generate DES key (in ms)									
330	380	330	330	390	380	390	380	390	380

5. CONCLUSION AND FUTURE WORK

5.0 Conclusion

The Internet is becoming a universal way of providing connectivity and application services. New applications are being conceived that have their own special requirements but are being adapted to run on the Internet due to market demand. People are demanding more robust services for less cost, which may be able to be achieved by using Internet technologies. These three factors are primarily responsible for driving an increased demand for more intelligent networking.

Directory Enabled Networking (DEN) was conceived as the foundation for building an intelligent network. DEN defines a means to store information describing the services that the users of the networks need and the capabilities of the devices that make up the network. The information is stored in a common repository in an agreed-upon format (e. g. MOF). DMTF's CIM is a standard information model, which could be used as a building block to build network applications. Detailed implementation is then the responsibility of the applications that build upon the model.

A simple network management program based on the previous work done by Dr. Blight, which is a much simplified CIM implementation, was developed in this thesis. The information stored in the directory is in MOF. A policy server can generate policy files, which are in MOF. The policy server then converts the MOF file to XML to leverage the portability of XML and sends it to the user. All the components used in the project are generic in nature. The program demonstrates the use of CIM and DEN in the

implementation of an intelligent network management system. In this system, CIM enables sharing information between different management systems; DEN integrates knowledge about network to provide users with end-to-end network services.

IPSec at the packet level is also modeled within the program. Once a VPN policy is invoked, it sends a VPN request to a server, which is listening for the requests from users. The server generates a VPN policy file in MOF, then converts the file to XML format and sends the resulting file to the user. The user reads the file and takes appropriate actions accordingly.

In the test that I have done, it takes about 41420 ms, which is significant long, to generate Diffie-Hellman key pairs. If several requests are sent simultaneously, the server may be overloaded. The solution to this problem is to use the same Diffie-Hellman key for a longer period, such as one per day. It only takes 368 ms to generate DES key. To maximize the protection for data, DES key can be changed from session to session.

5.1 Future Work

Because only VPN policy files are used, I did not implement methods which can be used to replicate and extend the directory service. As the network grows, each separate management domain must be configured individually. Adding support for replication could reduce the network administrator's workload by automatically replicating the values for common network components' parameters. A directory service must be able to grow and expand as new standards and/or applications evolve. Or the cost of constantly implementing new services and upgrading the system will be very high in the long run.

In this program, policy requests were sent in response to a user manually clicking on a button. This could be modified to add an application layer. Instead of sending a request directly, the user could then launch an application, which would then sends policy requests to the policy server.

Because the project is only a simplified version of CIM, it may not meet the needs of some of the network managers. To use the program for the specific needs of different networks, a network manager needs to extend the program. Some common areas that network managers might want to expand are the Policy class and IP address format for simple network management simulations. In the case of complex network programs, other classes, such as Applications and Users should be added.

The policy server can be used to coordinate the creation, modification, validation, administration, management, and installation of policies. A network can be informed about changes to existing policies or the additional of a new policy via the policy server. The policy server can also be used to communicate updates of policies to other network systems. This work also remains to be done.

Today, IP has established itself as the primary vehicle for global system of networking. As more and more people are connected to the Internet, the available IPv4 addresses will not be able to meet the growing demands for new IP addresses. IPv6 increases the IP address size from 32 bits to 128 bits. IPv6 addresses performance, scalability, security, ease-of-configuration, and network management issues that are central to the ongoing competitiveness and bottom-line performance of all types of network-dependent businesses. The project only implements support for IPv4 address

format. To meet future demands of network managers, the program can also be extended to include IPv6 address.

Only VPN policy files are used in this project. A policy file, in this program, only includes the parameters that are essential to the creation of VPN. The policy file can be extended to include user profile and application profile. A user profile can include the service level the user is entitled to and the access level the user has. An application profile can include the time the application can be run and the service level the application is entitled to. Policy can also be extended to incorporated enterprise's business goal to ensure maximum use of limited network resources.

The access protocol (Figure 17) was not implemented in this project. A network manager can add a servlet to this program so that the information can be accessed via web browsers.

Reference:

1. Judd, S., and Strassner, J., "Directory Enabled Networks", www.cisco.com/warp/public/cc/cisco/mkt/enm/diren/prodlit/densp_ai.pdf, Cisco, 1998.
2. Goncalves, M., "Directory Enabled Networks", McGraw Hill, 1999.
3. Weider, C., Reynodls, J., and Heker, S., "Technical Overview of Directory Services using the X.500 protocol", RFC 1309, 1992.
4. Yeong, W., Howes, T., and Kille, S., "Lightweight Directory Access Protocol", RFC 1777, 1995.
5. "Policy-based Networking—*creating the business-driven network*", Lucent Technologies.
6. Strassner, J., "Directory Enabled Networks", Macmillan Technical Publishing, 1999.
7. "CIM Tutorial", www.dmtf.org/spec/cim_tutorial/, Distributed Management Task Force, 1998.
8. "Common Information Model *Core Model White Paper*, version 1", www.dmtf.org/spec/cim_core/, Distributed Management Task Force, 1998.
9. "Common Information Model (CIM) Specification v2.2", www.dmtf.org/spec/cim_spec_v22/, Distributed Management Task Force, 1999.
10. "Understanding the Application Management Model, version 0.9". www.dmtf.org/spec/cim_apps/, Distributed Management Task Force, 1998.
11. "Specification for CIM Operations over HTTP". Distributed Management Task Force, 1999.
12. Kent, S., and Atkinson, R., "IP Authentication Header", RFC 2402, November 1998.

13. Kent, S., and Atkinson, R., "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

14. Kent, S., and Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

15. Maughan, D., Schertler, M., Schneider, M., and Turner, J., "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

16. Harkins, D., Carrel, D., "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

17. Hamada, T., Blight, D. C., and Czezowski, P. J., "Active Policies in Knowledge Hyperspace: Intelligent Agents and Policy Based Networking", KICS KNOW Review, vol 2, no 2, PP.31-41, December 1999.

18. Blight, D. C., Czezowski, P. J., "Issue in IP over DWDM Management", Optical Network Workshop (ONW 2000), Richardson, USA, January 2000.

19. Krawczyk, H., Bellare, M., and Canetti., R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

20. Madson, C., Glenn, R., "The Use of HMAC-MD5-96 within ESP and AH", RFC 2403, November 1998.

21. Stallings, W., "Data and Computer Communications", 5th Edition, Prentice Hall, 1997.

22. McConnell, J., "Building a Policy-Driven Infrastructure". McConnell Associates, 1999.

23. "Private Use of Public Networks for Enterprise Customers *New Standards-Based Virtual Private Networks Offer Cost Savings and Business Opportunities*". www.3com.com/technology/tech_net/white_papers/500651.htm, 3COM.

24. Ryan, J., "Managing the Costs and Complexities of VPN Deployment". www.techguide.com.

Appendix A: XML representation of CIM_ManagedSystemElement

```
<?xml version="1.0"?><!DOCTYPE CIM SYSTEM "cim.dtd">
<CIM>
 <CLASS NAME="CIM_ManagedSystemElement">
  <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
   <VALUE>CIM_ManagedSystemElement is the base class for the System Element hierarchy. Membership Criteria: Any distinguishable component of a System is a candidate for inclusion in this class. Examples: software components, such as files; and devices, such as disk drives and controllers, and physical components such as chips and cards</VALUE>
  </QUALIFIER>
  <QUALIFIER NAME="Abstract" TYPE="boolean" TOSUBCLASS="false">
   <VALUE>TRUE</VALUE>
  </QUALIFIER>
  <PROPERTY NAME="Caption" TYPE="string">
   <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
    <VALUE>The Caption property is a short textual description (one-line string) of the object</VALUE>
   </QUALIFIER>
   <QUALIFIER NAME="MaxLen" TYPE="sint32" TOSUBCLASS="false">
    <VALUE>64</VALUE>
   </QUALIFIER>
  </PROPERTY>
  <PROPERTY NAME="Description" TYPE="string">
   <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
    <VALUE>The Description property provides a textual description of the object</VALUE>
   </QUALIFIER>
  </PROPERTY>
  <PROPERTY NAME="InstallDate" TYPE="datetime">
   <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
    <VALUE>A datetime value indicating when the object was installed. A lack of a value does not indicate that the object is not installed</VALUE>
   </QUALIFIER>
   <QUALIFIER NAME="MappingStrings" TYPE="string" TOSUBCLASS="false">
    <VALUE.ARRAY>
     <VALUE>MIF.DMTF|ComponentID|001.5</VALUE>
    </VALUE.ARRAY>
   </QUALIFIER>
  </PROPERTY>
  <PROPERTY NAME="Name" TYPE="string">
   <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
    <VALUE>The Name property defines the label by which the object is known. When subclassed, the Name property can be overridden to be a Key property</VALUE>
   </QUALIFIER>
   <QUALIFIER NAME="MaxLen" TYPE="sint32" TOSUBCLASS="false">
    <VALUE>256</VALUE>
   </QUALIFIER>
  </PROPERTY>
  <PROPERTY NAME="Status" TYPE="string">
   <QUALIFIER NAME="description" TYPE="string" TOSUBCLASS="false">
    <VALUE>A string indicating the current status of the object. Various operational and non-operational statuses are defined. Operational statuses are \OK\, \Degraded\, \Stressed\ and \Pred Fail\. \Stressed\ indicates that the Element is functioning, but needs attention. Examples of \Stressed\ states are overload, overheated, etc. The condition \Pred Fail\ (failure predicted) indicates that an Element is functioning properly but
```

predicting a failure in the near future. An example is a SMART-enabled hard drive. Non-operational statuses can also be specified. These are \Error\, \NonRecover\, \Starting\, \Stopping\ and \Service\. \NonRecover\ indicates that a non-recoverable error has occurred. \Service\ describes an Element being configured, maintained or cleaned, or otherwise administered. This status could apply during mirror-resilvering of a disk, reload of a user permissions list, or other administrative task. Not all such work is on-line, yet the Element is neither \OK\ nor in one of the other states</VALUE>

```
    </QUALIFIER>
    <QUALIFIER NAME="MaxLen" TYPE="sint32" TOSUBCLASS="false">
     <VALUE>10</VALUE>
    </QUALIFIER>
    <QUALIFIER NAME="ValueMap" TYPE="string" TOSUBCLASS="false">
     <VALUE.ARRAY>
      <VALUE>OK</VALUE>
      <VALUE>Error</VALUE>
      <VALUE>Degraded</VALUE>
      <VALUE>Unknown</VALUE>
      <VALUE>Pred Fail</VALUE>
      <VALUE>Starting</VALUE>
      <VALUE>Stopping</VALUE>
      <VALUE>Service</VALUE>
      <VALUE>Stressed</VALUE>
      <VALUE>NonRecover</VALUE>
     </VALUE.ARRAY>
    </QUALIFIER>
  </PROPERTY>
 </CLASS>
</CIM>
```

Appendix B: Static variables used to identify different components of a network

```
ROUTER                          = 1;
DATASOURCE                      = 2;
OPTICALTERM                     = 4;
INTERNET                        = 8;
NETMARKER                      = 16;
DEVICE                         = 32;
APPLSYS                        = 64;
APPL                          = 128;
CONNECTION                    = 256;
SCONNECTION                   = 512;
STRINGLABEL                  = 1024;
MARKER                = 0x00000400;
MOVEABLE              = 0x00000800;
POLICY                = 0x00001000;
GROUP                 = 0x00002000;
OXC                   = 0x00004000;
XDEVICE               = 0x00008000;
```

Appendix C: Configuration File used In Figure 18

```
<CIM Name=Enterprise>
        <AgentFilter>
                <FilterEntry>
                        <Agent>Agents.QoS</Agent>
                </FilterEntry>
        </AgentFilter>
        <Router>
                <Name>Gateway-HeadOffice</Name>
                <GInfo>
                        <LOC><X>208</X><Y>113</Y></LOC>
                        <Line/>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                </GInfo>
        </Router>
        <Router>
                <Name>Gateway-Europe</Name>
                <GInfo>
                        <LOC><X>194</X><Y>310</Y></LOC>
                        <Line/>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                        <NFILE>images/r1.gif</NFILE>
                </GInfo>
        </Router>
        <Internet>
                <Name>Carrier1</Name>
                <GInfo>
                        <LOC><X>321</X><Y>188</Y></LOC>
                        <Line/>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                </GInfo>
        </Internet>
        <Internet>
                <Name>Carrier2</Name>
                <GInfo>
                        <LOC><X>320</X><Y>268</Y></LOC>
                        <Line/>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
                        <NFILE>images/internet.gif</NFILE>
```

```
        </GInfo>
</Internet>
<Router>
        <Name>Gateway-NA</Name>
        <GInfo>
                <LOC><X>497</X><Y>112</Y></LOC>
                <Line/>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
        </GInfo>
</Router>
<Router>
        <Name>Gateway-Asia</Name>
        <GInfo>
                <LOC><X>514</X><Y>310</Y></LOC>
                <Line/>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
                <NFILE>images/r1.gif</NFILE>
        </GInfo>
</Router>
<Connection>
        <Name>Conn0</Name>
        <Src>Carrier1</Src>
        <Dst>Carrier2</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>361</X><Y>212</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn1</Name>
        <Src>Gateway-HeadOffice</Src>
        <Dst>Carrier1</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>232</X><Y>137</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn2</Name>
        <Src>Carrier1</Src>
        <Dst>Gateway-NA</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>361</X><Y>212</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
```

```
<Connection>
        <Name>Conn3</Name>
        <Src>Gateway-Europe</Src>
        <Dst>Carrier2</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>218</X><Y>334</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn4</Name>
        <Src>Carrier2</Src>
        <Dst>Gateway-Asia</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>360</X><Y>292</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn5</Name>
        <Src>Gateway-HeadOffice</Src>
        <Dst>Carrier2</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>232</X><Y>137</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn6</Name>
        <Src>Carrier1</Src>
        <Dst>Gateway-Europe</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>361</X><Y>212</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn7</Name>
        <Src>Carrier1</Src>
        <Dst>Gateway-Asia</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>361</X><Y>212</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn8</Name>
        <Src>Carrier2</Src>
        <Dst>Gateway-NA</Dst>
        <BW>1500</BW>
        <GInfo>
```

```
                <LOC><X>360</X><Y>292</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Device>
        <Name>Database2</Name>
        <IPv4Address>192.168.2.0</IPv4Address>
        <GInfo>
                <LOC><X>101</X><Y>145</Y></LOC>
                <Line/>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
        </GInfo>
</Device>
<Device>
        <Name>Database1</Name>
        <IPv4Address>192.168.2.0</IPv4address>
        <GInfo>
                <LOC><X>104</X><Y>50</Y></LOC>
                <Line/>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
        </GInfo>
</Device>
<Device>
        <Name>Cache-Europe</Name>
        <IPv4Address>192.168.2.0</IPv4address>
        <GInfo>
                <LOC><X>100</X><Y>378</Y></LOC>
                <Line/>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
        </GInfo>
</Device>
<Device>
```

```
        <Name>Cache-NA</Name>
        <IPv4Address>192.168.2.0</IPv4Address>
        <GInfo>
                <LOC><X>571<?X><Y>49</Y></LOC>
                <Line/>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
        </GInfo>
</Device>
<Device>
        <Name>Cache-Asia</Name>
        <IPv4Address>192.168.2.0</IPv4Address>
        <GInfo>
                <LOC><X>607</X><Y>408</Y></LOC>
                <Line/>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
                <NFILE>images/noicon.gif</NFILE>
                <NFILE>images/d1.gif</NFILE>
        </GInfo>
</Device>
<Connection>
        <Name>Conn14</Name>
        <Src>Database2</Src>
        <Dst>Gateway-HeadOffice</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><X>125</X><Y>169</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn15</Name>
        <Src>Database1</Src>
        <Dst>Gateway-HeadOffice</Dst>
        <BW>1500</BW>
        <GInfo>
                <LOC><><X>128</X><Y>74</Y></LOC>
                <Line/>
        </GInfo>
</Connection>
<Connection>
        <Name>Conn16</Name>
        <Src>Cache-Europe</Src>
```

```
                <Dst>Gateway-Europe</Dst>
                <BW>1500</BW>
                <GInfo>
                        <LOC><X>124</X><Y>402</Y></LOC>
                        <Line/>
                </GInfo>
        </Connection>
        <Connection>
                <Name>Conn17</Name>
                <Src>Gateway-NA</Src>
                <Dst>Cache-NA</Dst>
                <BW>1500</BW>
                <GInfo>
                        <LOC><X>521</X><Y>136</Y></LOC>
                        <Line/>
                </GInfo>
        </Connection>
        <Connection>
                <Name>Conn18</Name>
                <Src>Gateway-Asia</Src>
                <Dst>Cache-Asia</Dst>
                <BW>1500</BW>
                <GInfo>
                        <LOC><X>538</X><Y>334</Y></LOC>
                        <Line/>
                </GInfo>
        </Connection>
</CIM>
```

Appendix D: DTD file for the configuration file in Appendix C

```
<!ELEMENT Class (AgentFilter, Router+, Internet+, Connection+, Device+)>
<!ATTLIST Class Name CDATA #REQUIRED>

<!ELEMENT AgentFilter (FilterEntry)>
<!ELEMENT FilterEntry (Agent)>
<!ELEMENT Agent (#PCDATA)>

<!ELEMENT Router (Name, GInfo)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT GInfo (LOC, Line, NFILE+)>
<!ELEMENT Line EMPTY>
<!ELEMENT LOC (X, Y)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT NFILE (#PCDATA)>

<!ELEMENT Internet (Name, GInfo)>

<!ELEMENT Connection (Name, Src, Dst, BW, GInfo)>
<!ELEMENT Src (#PCDATA)>
<!ELEMENT Dst (#PCDATA)>
<!ELEMENT BW (#PCDATA)>

<!ELEMENT Device (Name, IPv4Address, GInfo)>
<!ELEMENT IPv4Address (#PCDATA)>
```