WEB-based Network Monitoring Using SNMP, CGI and CORBA

By

Jizong Li

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of

Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-41732-8

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

WEB-BASED NETWORK MONITORING USING SNMP, CGI AND CORBA

BY

JIZONG LI

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

ABSTRACT:

The rapid development of the Internet poses a huge task for network management and makes it one of the hottest topics of the networking world. With the development of Web technology and its universal accessibility and platform independence, there is a trend to deploy Web browsers to monitor and manage networks. Since static Web pages do not provide the rich information compared to native windows based management applications, Java applets implementing graphical interfaces which show the real time state of the network are attractive alternatives.

The Simple Network Management Protocol (SNMP) was introduced in 1988 and is still widely used. It was initially designed as a short-term solution to manage TCP/IP based networks. But since TCP/IP is dominant in the world and unlikely give way to OSI (Open Systems Interconnection) based protocols, implementing and deploying SNMP for network management is still important and meaningful today.

Common Object Request Broker Architecture (CORBA) is defined by the Object Management Group (OMG) as providing a common architectural framework for object-oriented applications. It provides many advantages such as object service local/remote transparency, high-level language bindings and static and dynamic method invocations.

The motivation of this thesis is to take the advantage of WWW and CORBA technologies to create a Web-based network monitoring tool based on the SNMP protocol. Within this thesis, SNMP, WWW and CORBA technologies are reviewed and deployed creating a Web-based network monitoring tool. With this system the Object Identifier (OID) values under MIB-II can be retrieved and a real time graph displayed presenting data such as the total number of bytes in and out for all interfaces, IP packets in and out, on machines in a local network. The performance of monitoring via both CORBA and CGI middleware are compared and discussed.

The network monitoring client/server system is implemented in Java with the AdventNet basic SNMP API chosen as the Java SNMP library for this application. Also Visibroker from Insprise Corp. was selected to construct the CORBA environment.

ACKNOWLEDGMENTS

First I would like to express my sincere gratitude to my thesis advisor Dr. McLeod, for his valuable advice and continuous encouragement throughout my thesis research, also for his financial support during this research. I would like to give thanks to Dr. Ferens and Dr. Maheswaran for their interest to be my examining committee members and their time to read my thesis.

I would like to extend my thanks to lots of my colleagues for their help in technical aspect and in English language usage. Special thanks to our system administrator Guy Jonatschick, for his technical support in my research.

Furthermore I would like to express my thanks to IIC (Internet Innovation Centre) for giving me the access to their PC workstations and doing research tests on it.

Finally I would like to express my great thanks to my elder brother Jisong Li, for his support, continuous encouragement and thoughtful suggestions throughout my master program process. And also, sincere thanks to my lovely wife Sufang Hou, for her understanding and support, and our cute daughter, Shari.

ABBREVIATIONS:

API	Application Program Interfaces
ASN.1	Abstract Syntax Notation 1
BOA	Basic Object Adapter
CCITT	International Telegraph and Telephone Consultative Committee
CGI	Common Gateway Interface
CLTS	Connectionless Transport Service
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DBMS	DataBase Management System
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Invocation
HTML	HyperText Markup language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Organization for Standardization
JAR	Java ARchive
MIB	Management Information Base
MIME	Multipurpose Internet Mail Extensions

NMS	Network Management System
OID	Object IDentifier
OMG	Object Management Group
OO	Object-Oriented
OOSE	Object-Oriented Software Engineering
ORB	Object Request Broker
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
RFC	Request For Comment
RMON	Remote Network Monitoring
RFP	Request For Proposal
SGMP	Simple Gateway Monitoring Protocol
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
STDIN	Standard Input
STDOUT	Standard Output
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modelling Language
URL	Uniform Resource Locator
WWW	World Wide Web

TABLE OF CONTENTS:

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

In resent years, networks have been growing larger and more powerful, brought convenience to business, government, and individuals. The problem is that the network management task is also becoming increasingly complicated. In response to this, standards that deal with network management have been developed, covering services, protocols, and management information bases. Among them, Simple Network Management Protocol (SNMP) is the most widely used one for TCP/IP based networks.

1.1 Basic Concepts

SNMP was introduced in 1988 and was initially designed as a short-term solution to manage TCP/IP based networks. With SNMP's *Get, Set* and *Trap* operations, monitoring and controlling can be realized in TCP/IP networks [Stallings 96]. Since TCP/IP is dominant, implementation and deployment of SNMP management systems are important.

Because of the limitations and deficiencies in the original SNMP suite, SNMP v2 was introduced and published in 1993. To address the security and remote configuration capabilities issues, a recent set of RFCs, known collectively as SNMP v3, has also been recently introduced [Snmpv3] [Stallings 98].

A network management or monitoring system must have a management station or manager. The management station serves as the interface for the human network manager into the network management system so that the network manager can monitor and control the network management processes. Another key element in network management is the management agent. Any node in the network to be managed, such as PCs, workstations, servers, bridges and routers, should be equipped with an agent so that they can be managed from a management station. The agent gathers and records management information for one or more network elements and communicates that information to the manager. The communication is implemented according to a common network management protocol which is shared by all the management stations and agents.

Since the agent has a function of collecting and maintaining information for its local environment, the management information base (MIB) was introduced. The MIB contains current and historical information about its local configuration and traffic. The management station will maintain a global MIB with summary information from all the agents.

There are two techniques used for making the management information collected and stored by agents available to manager systems. One is polling, a process by which the manager queries the information from the agent and the agent responds by looking at its MIB. The other process is event reporting, which indicates that the manager listens for the event reports generated by the agents.

The heart of the network management system is a set of applications that meet the needs for network management. At a minimum, a system will include basic applications for performance monitoring, configuration control, and accounting. This thesis focuses on monitoring the system specified by the user in the local network and presenting the information via text or real time graph in the client's Web browser.

Common Object Request Broker Architecture (CORBA) is defined by the Object Management Group (OMG) to provide middleware for object-oriented applications. With a membership of over 800 companies, OMG represents the main spectrum of the computer industry except for

Microsoft. For the majority of the industry, the next generation of middleware is CORBA.

CORBA allows applications to communicate with one another no matter where they are located or who has designed them. Using CORBA, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other aspects of the system that are not part of an object's interface. The ORB takes care of everything and provides interoperability between objects on different machines in heterogeneous distributed environments [OMG].

More detail about SNMP and CORBA is provided in Chapter 2 and 3.

1.2 Challenges and Motivation

Since WWW technology is able to provide network management operators with universal accessibility, platform independence, friendly management behaviour as well as secure management operations, using Web technology to handle network management is becoming a trend. "Several large corporations have built custom browser interfaces to existing management applications, giving network managers a window into the state of the network..." [Corcoran 96]. It is no longer sufficient to present only static information through Web pages. Instead, we have to create Java applets that graphically show the state of the network and the applications running on it in near real time. More and more application packages, initiatives and companies address Web-based network management solutions, such as Web NMS from AdventNet, JMAPI from Sun, WBEM initiative from BMC Software, Cisco, Compaq, Intel and Microsoft [WBEM FAQ] [WBMP].

Another approach is led by Subrata Mazumdar at Bell Laboratories, where they proposed a

CORBA/SNMP Gateway between management applications in a CORBA domain and agents in a SNMP domain. The main function of the gateway is to dynamically convert the method invocations on object references in the CORBA domain to SNMP messages for MIB entries at the remote agents [Mazumdar 96].

CORBA has great advantages for dealing with distributed and heterogeneous systems, so it is very useful and natural to apply CORBA in the network management area. Also it brings a new architecture to integrate legacy systems and new development to the current client/server systems. More details will be provided in Chapter 3.

The objective of this thesis is to demonstrate the advantage of WWW and CORBA technologies in creating Web-based network monitoring tools based on the SNMP protocol. First, the AdventNet basic SNMP API was studied and chosen as the SNMP class library to construct the basic client/server system, from which the OID values under MIB-II can be retrieved and results displayed either in text or real time graph (e.g. the total number of bytes in and out for all interfaces, IP packets in and out, of any machine in the local network). Secondly, the basic client/server was extended with CGI (Common Gateway Interface) such that the network information can be browsed via the Web. Thirdly, instead of CGI, the basic client/server is extended with CORBA. Finally CORBA's power and advantages are illustrated by the comparison of the CGI and CORBA approaches.

The network monitoring client/server system is implemented in Java and the CORBA environment was constructed by Visibroker from Inprise Corp.

1.3 Structure of the thesis

This thesis is organised in several chapters as follows:

Chapter 2 gives the overview of SNMP and network management and more information about

SMI, MIB and SNMP specifications. SNMP limitations are also reviewed.

Chapter 3 introduces two middleware solutions: CGI and CORBA, and describes how they function in a 3-tier client/server model. Some discussion and comparison is given. CGI and CORBA programming are also introduced.

Chapter 4 introduces the AdventNet SNMP API software packages and then provides detail about how to construct a Web-based network monitoring tool using Java and CGI programs. An object-oriented approach based on UML is also introduced.

Chapter 5 introduces VisiBroker from Inprise Corp. (one of the leading CORBA products) and then explains how to construct a Web-based network monitoring tool using Java and CORBA. Also object-oriented models are provided.

Chapter 6 explains the functions of the constructed network monitoring tools, and provides discussion about the performance for network monitoring via CGI and CORBA.

Chapter 7 concludes this thesis with some final thoughts and comments. Also future work is given for the further research in this field.

CHAPTER 2

SNMP and Network Management Review

In this chapter, I provide a brief history about SNMP and then introduce SMI, MIB and SNMP specifications. Network management technology is then reviewed.

2.1 SNMP Basic

2.1.1 A Brief History of SNMP

SNMP was initially designed as a short-term solution to manage TCP/IP based networks. It was developed from the Simple Gateway Monitoring Protocol (SGMP) in 1988 and enhanced to meet the need for more general-purpose network management. CMIP (Common Management Information Protocol) was defined as the long-range solution in the case of a network transition from TCP/IP to the OSI architecture [Cerf 88]. At that time, it was thought that within a reasonable period of time, TCP/IP installations would give the way to OSI-based protocols. But due to the flourishing of TCP/IP applications and development in 1990s, it seems that TCP/IP will last much longer than all predictions. Therefore, SNMP became the most widely used network management protocol on TCP/IP-based networks and appears to remain in use for the long term. At the same time, the CMIP effort languishes [Stallings 96].

The set of specifications that define SNMP and its related functions and data bases is a series of related RFC (Request For Comment) documents which is still growing. Among them, there are three foundation specifications which are made of the basic SNMP set. They are:

- RFC 1155 - Structure and Identification of Management Information for TCP/IP based internets.
- RFC 1157 - A Simple Network Management Protocol (SNMP).

- RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II

Since limitations and deficiencies existed in the original SNMP suite, it was enhanced by the publication of SNMP v2. However, both versions of SNMP lack security features, notably authentication and privacy, which are required to fully exploit SNMP. A recent set of RFCs, known collectively as SNMPv3, now correct this deficiency and add security and remote configuration capabilities to SNMPv2 [Snmp FAQ] [Stallings 98].

Another important addition to the basic set of SNMP standards (SMI, MIB, SNMP) is RMON (Remote Network Monitoring) which defines a remote monitoring MIB for remote management of networks. Since this thesis does not deal with RMON, it will not be explored further.

2.1.2 SNMP Model

The SNMP set of standards provide a framework for the definition of management information and a protocol for the exchange of that information.The SNMP model of a managed network consists of four key elements: management stations, managed nodes (agent), management information base and a network management protocol. These pieces are illustrated in Figure 2.1.

Network management is done from the management station, which is equipped with manager software that is responsible for managing part or all of the configuration on behalf of network management applications and users. It may provide a graphical user interface to allow the network manager to inspect the status of the network and take action when required.

The managed nodes can be hosts, routers, bridges or any other devices capable of communicating status information to the outside world. To be managed by a SNMP manager, a node must be capable of running an SNMP agent, which is a software module in a managed device responsible for maintaining local management information and delivering that information to a manager by request. The structure of information is determined by the Management Information Base (MIB).

Each system in a network maintains a MIB that reflects the status of the managed resources in that system.

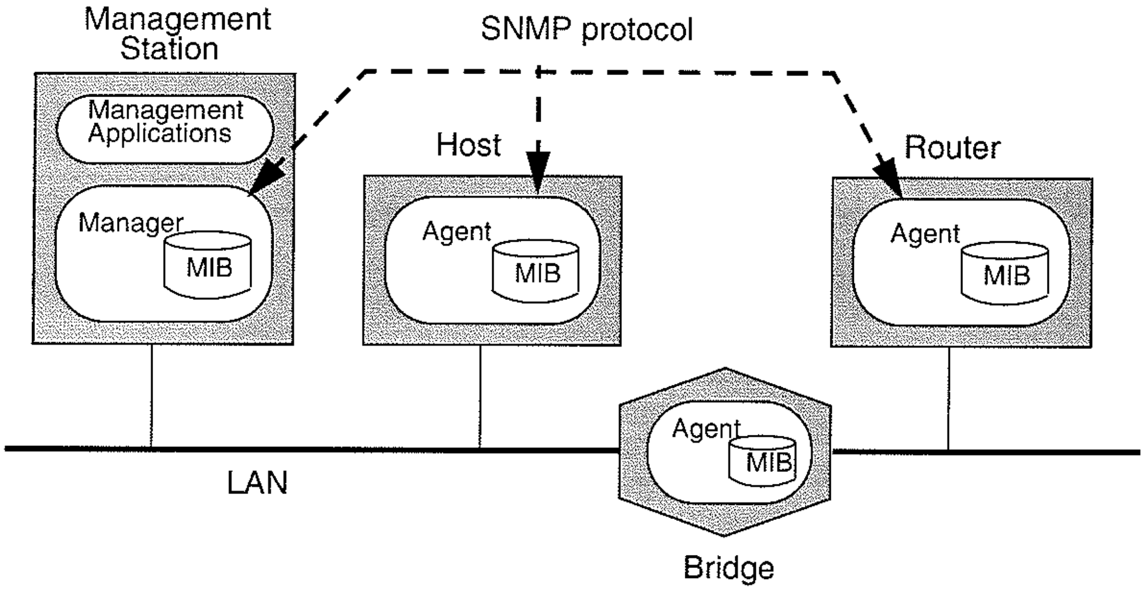


Figure 2.1 Components of the SNMP Management Model

The communication between the manager and the agents is based on the SNMP protocol. This protocol allows the manager to query the state of an agent's local resource objects, which are defined in MIB, and change them if necessary. There are three general-purpose operations [Stallings 96]:

- Get: A management station retrieves the value of objects from a managed station.
- Set: A management station updates the value of objects in a managed station.
- Trap: A managed station sends an unsolicited scalar object value to a management station to notify it of some significant events.

Based on SNMP, the Get operation has two options: GetRequest and GetNextRequest, which indicates a request of the variable following the current one. In SNMP v2, there is one more option provided: GetBulkRequest which is useful for getting a large table value of variables.

Also, SNMP v2 provides InformRequest operation to enable one manager to send trap type information to another.

When used in TCP/IP networks, SNMP is an application-level protocol and uses the UDP transport, so it is connectionless. No ongoing connections are maintained between a manager and its agents. An illustration of SNMP protocol and applications within network architecture [Stallings 96] is shown in Figure 2.2, where network-dependent protocols can be Ethernet, FDDI and X.25, etc.

SNMP accommodates the management of devices that do not implement the SNMP software by means of proxies. A proxy is an SNMP agent that maintains information for one or more non-SNMP devices for which it is responsible and communicates with the management station on behalf of them.

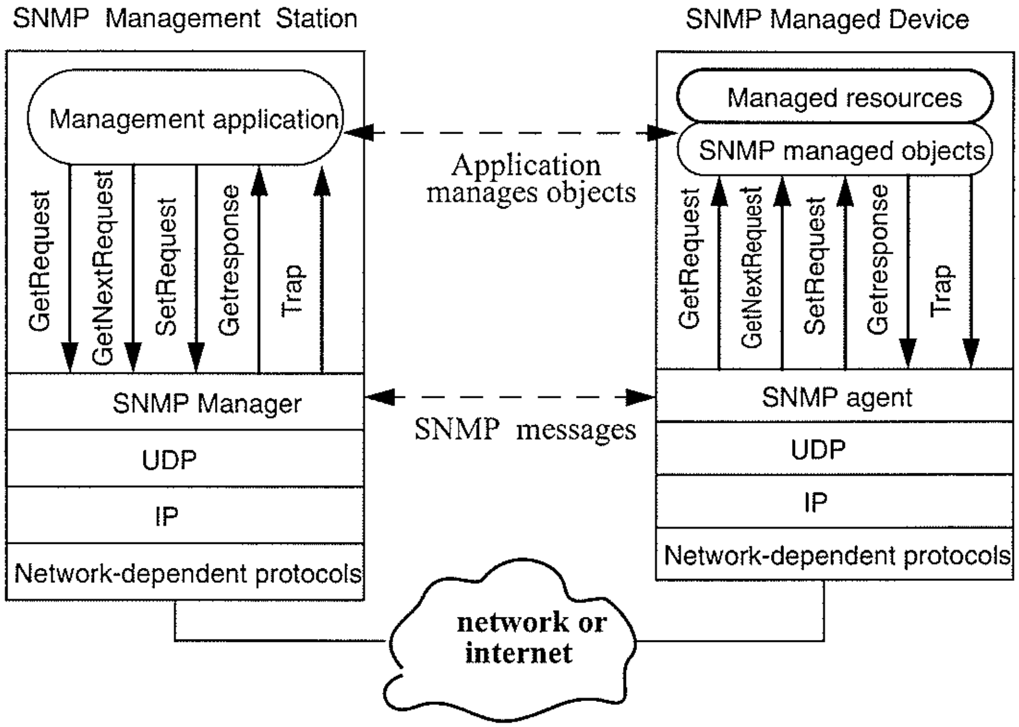


Figure 2.2 The Role of SNMP in Network

2.2 Introduction to SMI and MIB

2.2.1 ASN.1 — Abstract Syntax Notation 1

ASN.1 is a formal language developed and standardized by CCITT (X.208) and ISO (ISO 8824). It is the most widely used language to define abstract syntaxes of application data. It is used to define the formats of the PDUs (Protocol Data Unit) that are exchanged by SNMP entities and the management information base for both SNMP and OSI systems management.

ASN.1 mainly deals with abstract syntax, data type, encoding rules and transfer syntax. In ASN.1, you can define modules, which are collections of ASN.1 descriptions, each description referring to an object. Possible objects can be data types and macros. A type may be simple, which is defined by specifying the set of its values, or structured, which is defined based on one or more simple types. There are several simple types available for SNMP, which are: Integer, Bit String, Octet String, Object Identifier and NULL.

All versions of SNMP are based on ASN.1:1988 (that is X.208). [Snmp FAQ]

2.2.2 SMI and MIB

The Structure of Management Information (SMI), which is specified in RFC 1155 [Rose 90], defines the general framework within which a MIB can be defined and constructed. The SMI identifies the data types that can be used in the MIB and specifies how resources within the MIB are represented and named.

For network management based on SNMP, all SNMP manageable resources or entities are described as objects and grouped in management information base (MIB). The original SNMP MIB (MIB-I) for managing a TCP/IP Internet was defined in RFC 1066 in 1988. And later in 1991, MIB-II was published in RFC 1213 which added some much-needed objects, and since then has become the standard SNMP MIB.

The MIB is structured as a tree. In this tree structure, all SNMP variables or objects are represented as branches and leaves, and named according to the type OBJECT IDENTIFIER of ASN.1. The managed objects are grouped into logically related sets from the root. The root of the tree is the object referring to the ASN.1 standard and unlabelled. Starting from the root, there are three nodes at the first level and labelled as: **iso** (1), **ccitt** (0) and **joint-iso-ccitt** (2). Among them **iso** stands for International Organization for Standardization, **ccitt** for International Telegraph and Telephone Consultative Committee, and **joint-iso-ccitt** for jointly administered by the ISO and the CCITT.

Under the **iso** (1) node, the ISO has designated one subtree for use by other organizations, **org** (3), and one child of it is the U.S. Department of Defence, **dod** (6). RFC 1155 makes one subtree under **dod** allocated for administration by the Internet Activities Board as follows:

```
internet   OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) 1 }
```

Further down from the internet node is *mib-2* which we will use for network management, and the object identifier of it is derived as follows:

```
mib-2  OBJECT IDENTIFIER :: = { iso(1) org(3) dod(6) internet(1) mgmt(2) 1}
```

Thus the *mib-2* node has the object identifier value of 1.3.6.1.2.1, which is the global root prefix of every SNMP MIB object.

There are eleven nodes attached under *mib-2*. They are the important network information groups for SNMP except *cmot* (1.3.6.1.2.1.9) now. The Figure 2.3 shows this structure.

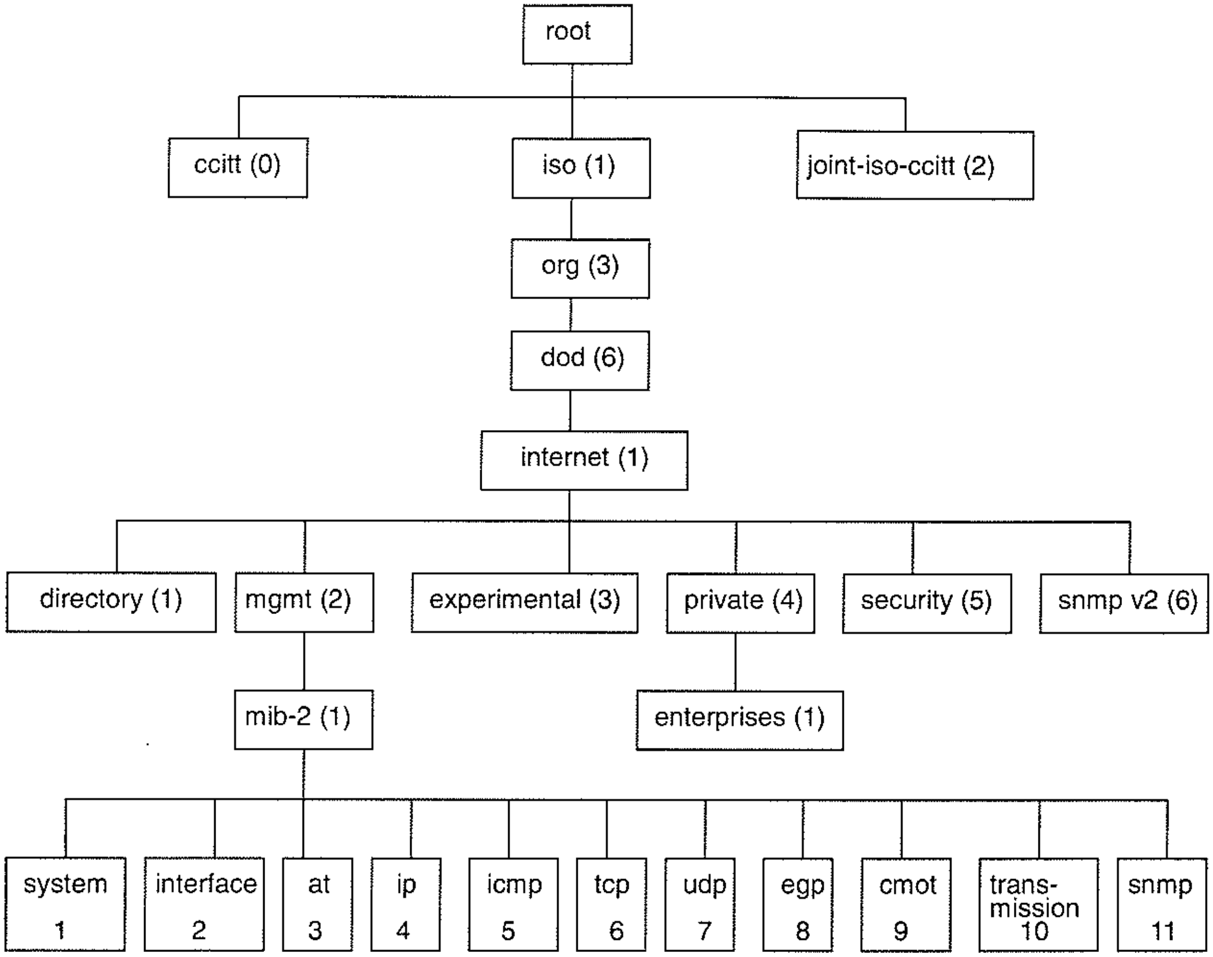


Figure 2.3 The Structure of the MIB

2.2.3 The system group in MIB-II

The information in the system group is very important for network management. In this thesis, the codes for displaying the system information via the Web were developed and the result can be displayed by the Web browser. A brief explanation follows.

As RFC 1213 defined, the system group provides general information about the managed system. It contains seven objects (the structure is shown in Figure 2.4 and definitions are given in Table 2.1) [Stallings 96]. If an agent is not configured to have a value for any of these variables, a string of length 0 is returned [McCloghrie 91].

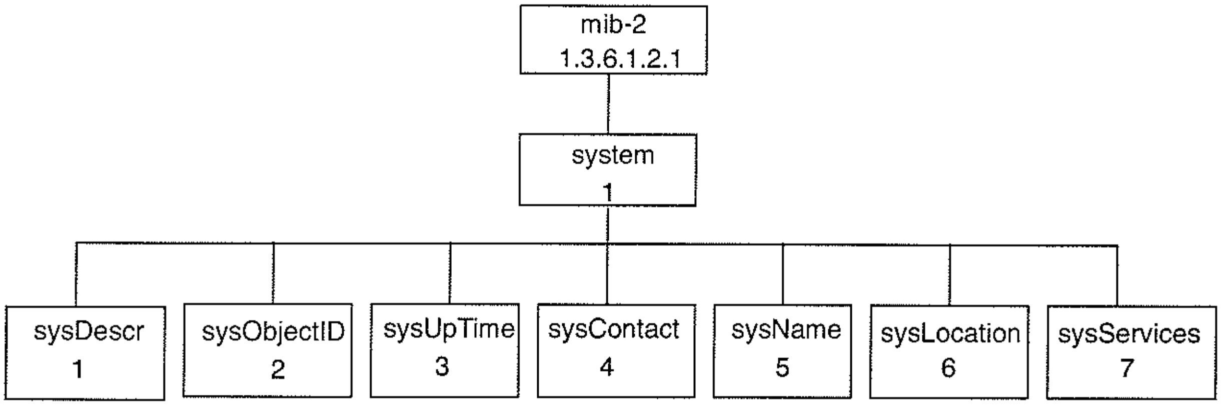


Figure 2.4 MIB-II system Group

Table 2.1 The *system* Group Objects

Object	Syntax	Access	Description
sysDescr	DisplayString (size (0 ... 255)	RO	A description of the entity, such as the full name and version identification of the system's hardware type, software operating-system, and networking software.
sysObjectID	OBJECT IDENTIFIER	RO	The vendor's authoritative identification of the network management subsystem contained in the entity. It is allocated within the SMI enterprises subtree (1.3.6.1.4.1).
sysUpTime	TimeTicks	RO	The time since the network management portion of the system was last re-initialized.
sysContact	DisplayString (size (0 ... 255)	RW	The identification and contact information of the contact person for this managed node.
sysName	DisplayString (size (0 ... 255)	RW	An administratively-assigned name for this managed node.
sysLocation	DisplayString (size (0 ... 255)	RW	The physical location of this node.
sysServices	INTEGER (0 ... 127)	RO	A value which indicates the set of services that this entity primarily offers.

* RW -- read & write * RO -- read only

According to RFC 1213, sysServices value is interpreted as a seven-bit code. Each bit of the code corresponds to a layer in the TCP/IP or OSI architecture, with the least significant bit corresponding to layer 1 (physical layer). For each layer L in the range of 1 through 7, which this node performs transactions for, 2 raised to (L - 1) is added to the sum. According to the OSI model, the layers and functionality are shown Table 2.2:

Table 2.2 The Layers in OSI Architecture

Layer	Functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., IP gateways)
4	end-to-end/transport (e.g., IP hosts)
5	session
6	presentation
7	applications (e.g., mail relays)

2.2.4 The enterprise group in MIB-II

In Figure 2.3, the enterprise object is located under the private (4) subtree. The enterprise group is used to allow enterprises (venders, etc.), who provide networking subsystems, to register models of their products and make them public so that network managers can use them to manage some products from this enterprise node.

A branch within the enterprise subtree is allocated to each enterprise that registers for an enterprise object identifier. Many enterprises have their own enterprise MIB, such as Proteon, IBM, CMU, ACC, Cisco, et al. There is a file named "enterprises.oid" provided by Sun Microsystems, Inc., which provides the enterprise's name and its OID in the Structure of Management Information. Some familiar enterprise names in this file are listed in Table 2.3.

Table 2.3 Enterprise Names and their OIDs

Name of Enterprise	OID
Reserved	1.3.6.1.4.1.0
Proteon	1.3.6.1.4.1.1
IBM	1.3.6.1.4.1.2
...	
cisco	1.3.6.1.4.1.9
NSC	1.3.6.1.4.1.10
HP	1.3.6.1.4.1.11
Novell	1.3.6.1.4.1.23
...	
Sun Microsystems	1.3.6.1.4.1.42
3Com	1.3.6.1.4.1.43
...	

Each enterprise MIB is also defined according to SMI and ASN.1. For example, the file CISCO-SMI.my from Cisco System Inc. shows:

```
...
ciscoProducts OBJECT IDENTIFIER ::= { cisco 1 }
-- OBJECT-IDENTITY
    Status:  mandatory
    Descr:
        ciscoProducts is the root OBJECT IDENTIFIER from which sysObjectID values are assigned.
        Actual values are defined in CISCO-PRODUCTS-MIB.

local OBJECT IDENTIFIER ::= { cisco 2 }
-- OBJECT-IDENTITY
    Status: mandatory
    Descr:
        Subtree beneath which pre-10.2 MIBS were built.
...
```

In my application of system monitoring, the developed programs can be run to obtain system information from the SNMP daemon (agent) according to system group in MIB-II. This program will interpret the part of that *sysObjectID* value to the enterprise name based on the file "enterprise.oid" and display it on the Web browser.

2.2.5 The interface group in MIB-II

Important information is contained in the interface group, such as the number of interfaces and the type of interfaces on the queried machine, the number of them that are up (ready to work) and the number of them down. The Figure 2.5 illustrates the OID tree under the interface group and Table 2.4 gives the definition of some of them [Stallings 96].

Table 2.4 The *interfaces* Group Objects

Object	Syntax	Access	Description
ifNumber	INTEGER	RO	the number of network interfaces
ifTable	sequence of ifEntry	NA	a list of interface entries
ifEntry	SEQUENCE	NA	an interface entry containing objects at the subnetwork layer and below for a particular interface
ifIndex	INTEGER	RO	a unique value for each interface
ifDescr	DisplayString (size (0 ... 255))	RO	information about the interface, including name of manufacturer, product name, and version of the hardware interface
ifType	INTEGER	RO	type of interface, distinguished according to the physical/link layer
ifMtu	INTEGER	RO	the size of the largest protocol data unit, in octets, that can be sent/received on the interface
ifSpeed	Gauge	RO	an estimate of the interface's current bandwidth in bits/sec.
ifPhysAddress	PhysAddress	RO	the interface's address at the protocol layer immediately below the network layer
ifAdminStatus	INTEGER	RW	desired interface state (up (1), down (2), testing (3))
ifOperStatus	INTEGER	RO	current operational interface state (up (1), down (2), testing (3))
ifLastChange	TimeTicks	RO	value of sysUpTime at the time the interface entered its current operational state
ifInOctets	Counter	RO	total number of octets received on the interface, including framing characters
ifInUcastPkts	Counter	RO	number of subnetwork-unicast packets delivered to a higher-layer protocol
...	...	...	...
ifOutOctets	Counter	RO	total number of octets transmitted on the interface, including framing characters
...	...	...	...

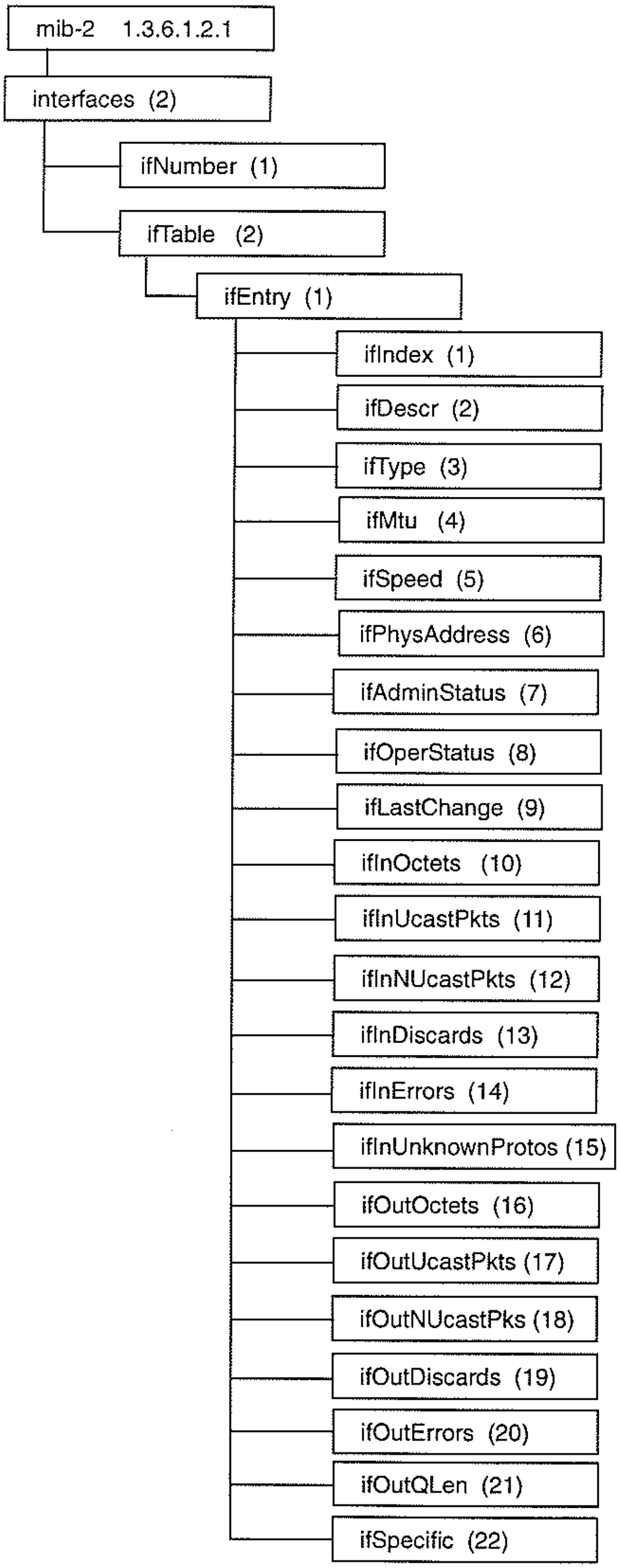


Figure 2.5 The Interfaces Group Structure in MIB-II

The information in this group is a starting point for many network management functions. For example, we can detect which interface is up or down. Also the total number of octets into or out of the system can be used to detect congestion in the network. Once congestion has been detected, other group objects can be examined to find out if this happened at the IP level or the TCP level, etc.

In the application developed here, the OID values in this group can be retrieved and a real time graph displayed, showing the total number of bytes in and out for all interfaces on machines in the network. The accompanying Description Window gives the descriptions of the OIDs in this group.

2.2.6 The ip group in MIB-II

According to RFC 1213, implementation of the IP group is mandatory for all systems. There are 23 objects within this group and the information about them is relevant to the implementation and operation of IP at a node. Figure 2.6 shows the structure of this group. Some objects of this group contain basic counters of traffic flow into and out of the IP layer. The definitions of them are given in Table 2.5 [Stallings 96]. Two of them are chosen in my application, one is *ipInDelivers* and the other *ipOutRequests*. The values of these two objects on machines in our network can be retrieved and displayed on a real time graph. As mentioned in 2.2.5, the objects which contain traffic flow information in this group will be considered with the objects in the interface group when there is congestion in the network.

For more information about the objects in this group, please refer to [McCloghrie 91].

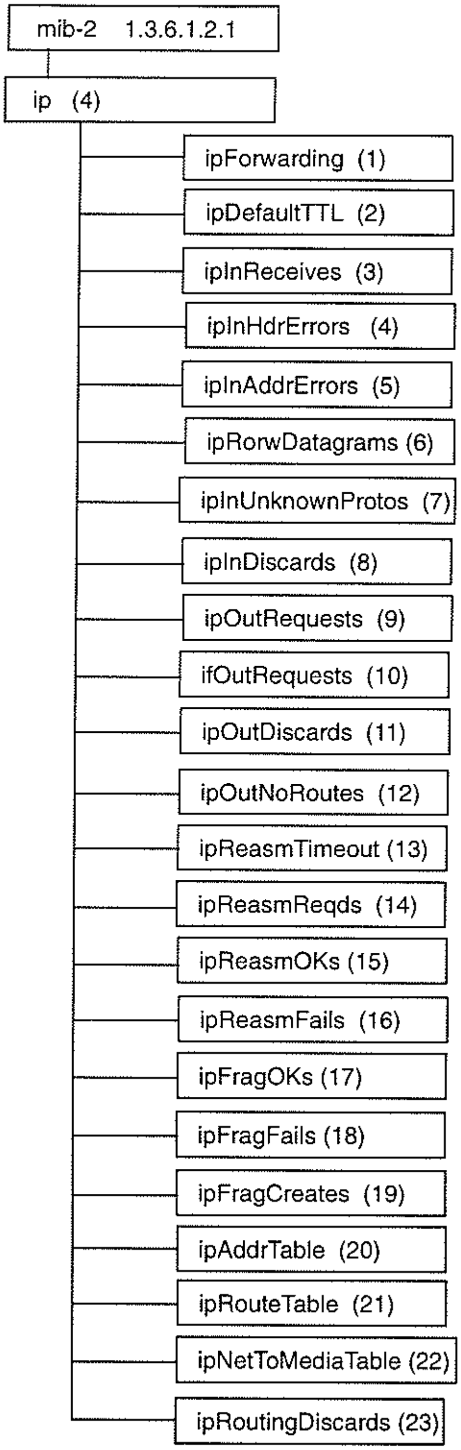


Figure 2.6 The ip Group Structure in MIB-II

Table 2.5 Part of Objects in *ip* Group

Object	Syntax	Access	Description
ipForwarding	INTEGER	RW	acting as an IP gateway (1), not acting as an gateway (2)
ipDefaultTTL	INTEGER	RW	default value inserted into Time-To-Live field of IP header of datagrams originated at this entity
ipInReceives	Counter	RO	total number of input datagrams received from interfaces, including those received in error
ipInHdrErrors	Counter	RO	number of input datagrams discarded due to errors in IP header
ipInAddrErrors	Counter	RO	number of input datagrams discarded because the IP address in the destination field was not valid to be received at this entity
ipForwDatagrams	Counter	RO	number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to forward
ipInUnknownProtos	Counter	RO	number of locally addressed datagrams received successfully but discarded because of an unknown or unsupported protocol
ipInDiscards	Counter	RO	number of input IP datagrams for which no problems were encountered to prevent their continued processing but which were discarded (e.g., lack of buffer space)
ipInDelievers	Counter	RO	total number of input datagrams successfully delivered to IP user protocols
ipOutRequests	Counter	RO	total number of IP datagrams that local IP user protocols supplied to IP in requests for transmission
ipOutDiscards	Counter	RO	number of output IP datagrams for which no problems were encountered to prevent their continued processing but which were discarded (e.g., lack of buffer space)
ipOutNoRoutes	Counter	RO	number of IP datagrams discarded because no route could be found
...	...	...	...

2.3 SNMP Specifications

In RFC 1157, the network management protocol is defined as an application protocol by which the variables of an agent's MIB may be inspected or altered. SNMP provides three general-purpose operations: *get*, *set* and *trap* so that the manager can get or set the variables in an agent's MIB and be notified by a trap if some significant events happen.

2.3.1 SNMP Formats

With SNMP, communication among protocol entities is accomplished by the exchange of messages which is represented within a single UDP datagram using the basic encoding rules of ASN.1. Each message includes a version number indicating the version of SNMP, a community name to be used for this exchange, and one of five types of protocol data units (PDU) [Case 90]. This structure is depicted informally in Figure 2.7 [Stallings 96]. Some of the message fields are explained in Table 2.6.

Version	Community	SNMP PDU

(1) SNMP message

PDU type	requestID	0	0	variable-bindings

(2) GetRequest PDU, GetNextRequest PDU, and SetRequest PDU

PDU type	requestID	errorStatus	errorIndex	variable-bindings

(3) GetResponse PDU

PDU type	enterprise	agentAddr	genericTrap	specific Trap	time stamp	variable-bindings

(4) Trap PDU

name1	value1	name2	value2	...

(5) variable-bindings

Figure 2.7 SNMP Format

Note: the meaning of SNMP PDU is different from most other protocol usages [Blight 97]

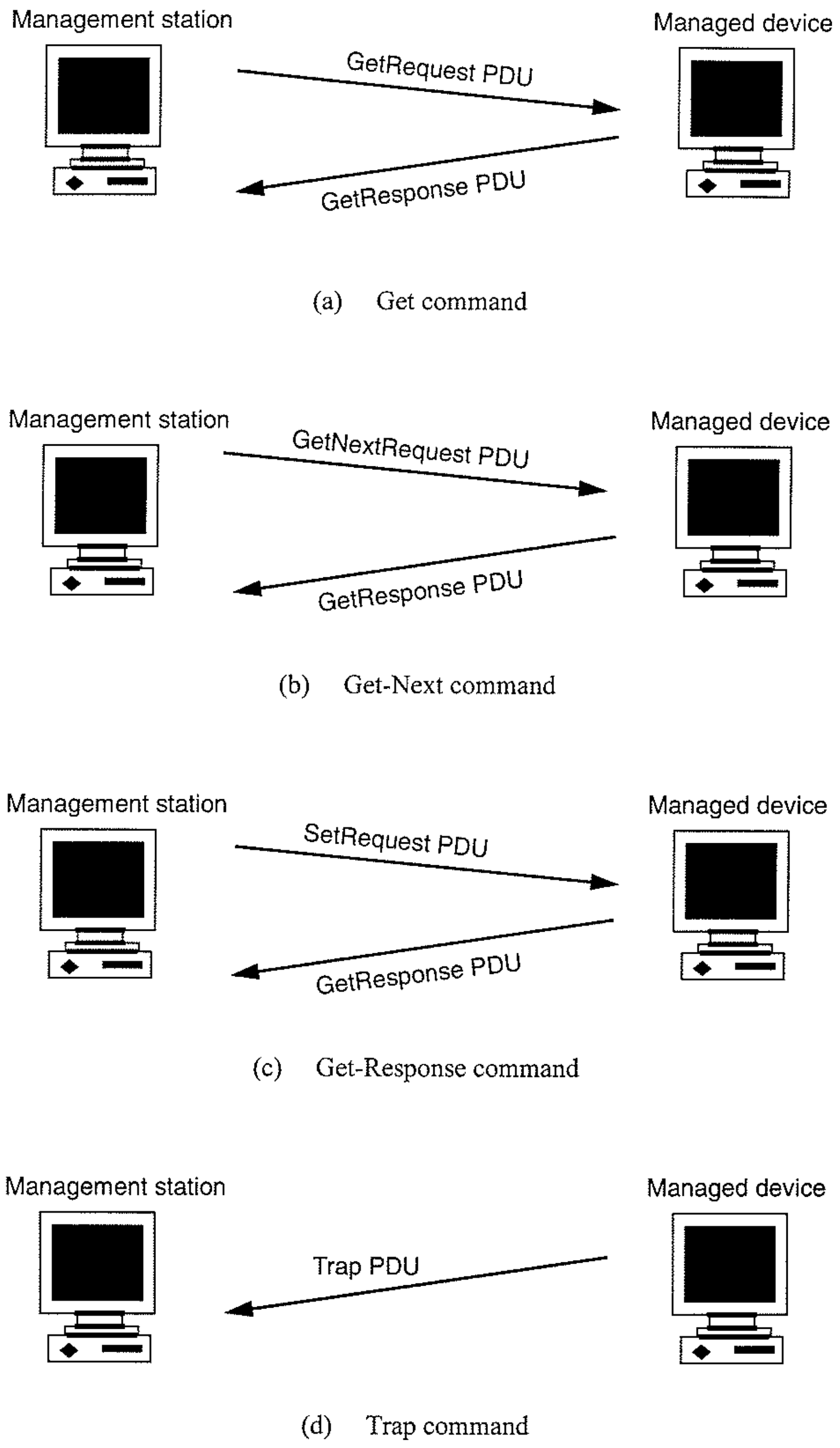
- SNMP PDU refers to the information portion only
- Common PDU refers to the whole message

Table 2.6 SNMP Message Fields

community	the name of it acts as a password to authenticate the SNMP message, the public community is used to access most information.
errorStatus	the integer value used to indicate if an exception occurred while processing a request, they are: noError (0), tooBig (1), noSuchName (2), badValue (3), readOnly (4) and genErr (5).
errorIndex	the integer value used while errorStatus is nonzero to provide additional information by indicating which variable in a list caused the exception (the variable is an instance of the managed object).
genericTrap	the integer value indicates which event happened in managed device, they are: • coldStart(0): the sending protocol entity (SPE) is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered • warmStart(1): the SPE is reinitializing itself such that neither the agent configuration nor the protocol entity implementation is altered. • linkDown(2): one of the communication links has failed. Variable-bindings will refer to ifIndex of the failed interface. • linkUp(3): one of the communication links has come up. Variable-bindings will refer to ifIndex of up interface. • authenticationFailure(4): signifies that a received protocol message is not properly authenticated. • egpNeighborLoss(5): signifies that an EGP (Exterior Gateway Protocol) neighbour has been disconnected. • enterpriseSpecific(6): signifies some enterprise-specific event has occurred.
specificTrap	the integer value indicates the particular trap occurred which is not enterprise Specific.

2.3.2 SNMP commands and sequences

There are five basic SNMP commands: *Get, Get-Next, Get-Response, Set* and *Trap*. Correspondingly, there are five PDU types: *GetRequest PDU, GetNextRequest PDU, GetResponse PDU, SetRequest PDU* and *Trap PDU*. The format of them have been previously mentioned, the mechanism of how it works is illustrated in Figure 2.8.



(a) Get command

(b) Get-Next command

(c) Get-Response command

(d) Trap command

Figure 2.8 SNMP PDU Sequences

2.3.3 Connectionless Transport Service and Port Number

SNMP was intended for use over a connectionless transport service. The reason for this is that various kinds of failures and outages of device and entities may happen in a network. If SNMP relies on the use of the connection-oriented transport service, then the loss of that connection could impair the effectiveness of SNMP exchanges. So SNMP implementations use UDP (User Datagram Protocol) transport service in the TCP/IP architecture. In OSI architecture, it is also possible to support SNMP using the connectionless transport service (CLTS).

UDP segments are transmitted in IP datagrams. The UDP header includes source and destination port fields, enabling SNMP entities to address each other. A protocol entity receives all messages at UDP port 161 except for trap message. Management stations listen for incoming Traps on UDP port 162. Implementation of this protocol need not accept messages whose length exceeds 484 octets. However, it is recommended that implementations support larger datagrams whenever feasible [Snmp FAQ].

2.3.4 SNMP Limitations

There are some limitations within SNMP [Ben-Artzi 90], listed as follows:

(1) One packet of information per request. Not suitable to handle complicated networks where large volumes of data need to be polled. In that case CMIP scores better than SNMP.

(2) SNMP traps are unacknowledged. In this way, an agent cannot be sure that a critical message has reached the management station.

(3) Poor security. The name of the community acts as a password to authenticate the SNMP message. Basic SNMP is better suited for monitoring than control.

(4) Only simple data structure. Not suited to query object values or types.

(5) Does not support manager-to-manager communications.

(6) Not directly supportive of imperative commands.

Many of these deficiencies were improved or addressed in SNMP v2 and SNMP v3.

2.4 Network Management Features

A functional classification of network management is defined by the ISO as the following five categories: fault management, accounting management, configuration and name management, performance management, and security management.

Performance management of a computer network comprises two broad functional categories -- monitoring and controlling. Monitoring is the function that observes and tracks the events and activities on the network, and the controlling function enables performance management to make adjustments to improve network performance.

Also, according to Stallings, there are three components in performance monitoring: performance measurement, performance analysis and synthetic traffic generation [Stallings 96]. Among them, performance measurement deals with the gathering of statistics about network traffic and timing. It is often accomplished by the agents running on the devices on the network (hosts, routers, bridges, etc.). These agents are in a position to observe the amount of traffic into and out of a node, the number of connections (network level, transport level, application level) and the traffic per connection, and other measures that provide a detailed picture of the behaviour of that node.

Performance analysis is responsible for analysing and presenting the data, and synthetic traffic generation for the observation of a network under a controlled load. With proper performance analysis, the network manager can monitor network availability, response time, throughput, and resource usage. With SNMP's Get and Set operations, the network manager can do the performance management in the network.

Fault Management deals with detecting, isolating and correcting network problems. With

SNMP's Trap operation, the manager station can get the problem report from the SNMP agent running on that machine. The network manager can then decide how to deal with it, by correcting or isolating that problem entity.

Perkins mentioned, "With properly designed MIBs, SNMP can be used to manage network configuration, performance, faults, accounting, and security." [Perkins 96]. He also gave the general guidelines for defining objects and MIBs. Since this thesis only focused on monitoring the network using the existing MIBs, it did not involve any MIB designs.

2.5 Network Management Architecture

Figure 2.1 gives a simple picture of the local network management environment. It includes the network management station and managed nodes (agent). Both use a management protocol (such as SNMP) to communicate values of variables stored in a MIB. Since the centralized computing model gives way to the distributed computing architecture, network management is also becoming distributed. A distributed network management system places several interoperable network management stations (managers) throughout the internet where each of these departmental-level managers (or sub-managers) is responsible for the management of those downsized applications and networks. One central workstation either interacts with sub-managers or goes directly to each agent to manage all the resources in the network. Figure 2.9 illustrates the basic structure used for distributed network management systems based on the above idea.

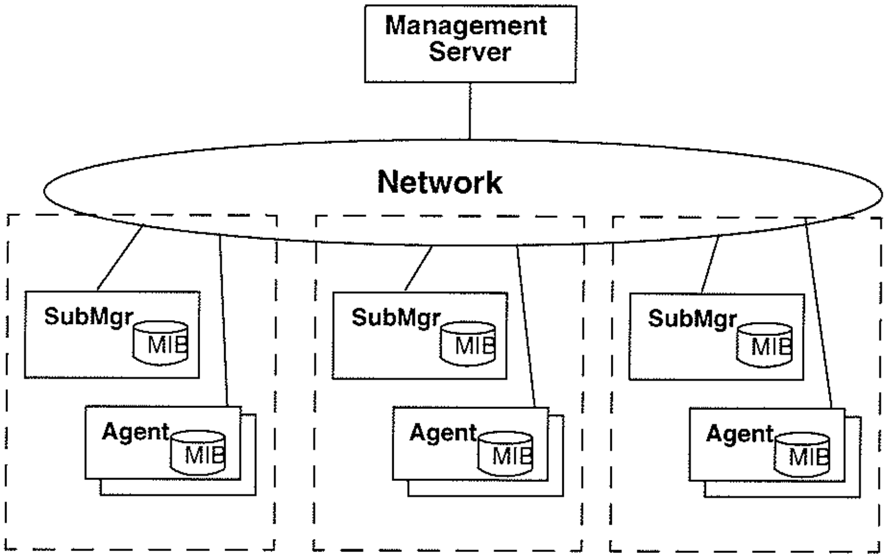


Figure 2.9 Typical Distributed Management System Architecture

CHAPTER 3

CGI and CORBA

Today, many people are familiar with the World Wide Web, as well as the meaning of HTTP, URLs and Web servers. If a user clicks a link pointing to a special URL and this URL is a location of a HTML page, this HTML page will be returned, even combined with images, sound clips, or Java applets. Furthermore, URLs also provide the naming scheme used to locate programs on the Web Server side.

For Java applets, we know that Web browsers impose two types of security restrictions on them (also called Java sandbox security):

- They allow applets to only connect back to the host where the applet was downloaded.
- They allow applets to only accept incoming connections from the host where the applet was downloaded.

If Java applets want to communicate with the agents residing in other machines behind the Web server (such as SNMP daemon running on the machines in the local network), they need to work around the above sandbox. This can be realized using CGI and CORBA middleware.

3.1 CGI Review

3.1.1 What is CGI

CGI stands for Common Gateway Interface. It defines a method for the HTTP server and an outside program to share information. The Web server only knows how to handle HTML documents. When it receives a request from a client to run a gateway program (often called a CGI script), it simply invokes the program named in the URL which takes care of the request. The

Web server summarizes the pertinent information about the request in a standard set of environment variables and passes them to that program. During this time it is using a protocol (or standard)* called the Common Gateway Interface (CGI).

The script then uses the useful information passed from the server and executes the request, and returns the results in HTML format to the Web server using the CGI protocol. The Web server treats the results like a normal document that it returns to the client. In a sense, the Web server acts as conduit between the Web client and a back-end program that does the actual work.

3.1.2 3 - Tiered Client/Server Model Using CGI

Usually a simple client/server system is called 2-tiered system. In this system, the user interface screens and associated logic residing on the client side constitute the first tier. The business logic functions and associated data on the server side constitute the second tier. According to Harmon, the tiers are the platforms, and "networking hardware and software that links the client and server platforms are middleware" * [Harmon 98]. Without CGI or other middleware involved on the Web server side, Web client and server just constitute a 2-tiered client/server system.

CGI bridges the gap between the Web server and other Internet services, such as updating databases on the back-end server or managing the machines in the network behind the Web server. In this situation it functions as the middleware between WWW servers and external databases or other information services. With CGI involved, we have the third tier in client/server architecture, which is constituted by the external database and information service systems. Figure 3.1 shows

* According to William E. Weinman's "The CGI Book", "CGI is not really a language or a protocol in the strictest sense of those terms. It is really just a set of commonly named variables and agreed-upon conventions for passing information from the server to the client and back again."

this new 3-tier client/server architecture. Usually the first tier belongs to the client's screen, the middle tier is composed by the server objects which represent the persistent data and the business logic functions and the third tier belongs to traditional servers (legacy systems). In Figure 3.1, our middle tier is the Web Server augmented with the CGI application servers *.

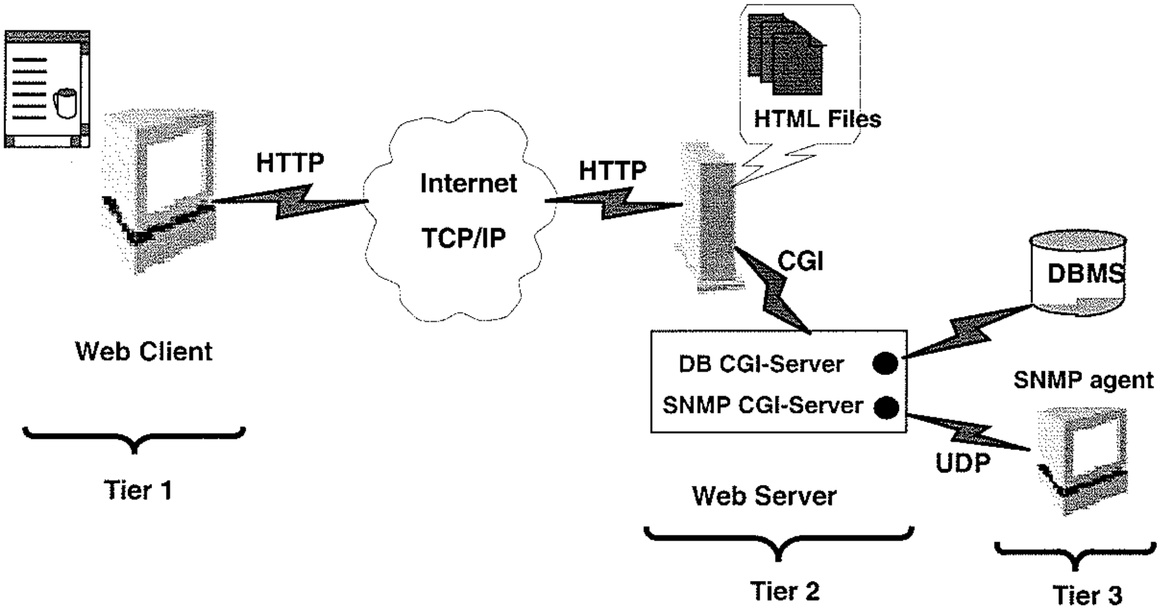


Figure 3.1 The Web Client/Server 3-tier model using CGI

Vendors in the network area are using this technology to connect Web browsers to their element managers to oversee their equipment. According to Jander, "Bay, for example, offers browser access to several of its Optivity management applications. 3Com sells a suite of tools that allows its Transcend software to be accessed over the Internet..." [Jander 96].

* In [Enck 97], "middleware sits between the client and server and forms a third tier in the client/server network."

3.1.3 CGI Specification -- Extending HTTP

The CGI specification was created and documented by the main HTTP server authors: Tony Sanders, Ari Luotonen, George Phillips, and John Franks. Originally the service from the HTTP server was very limited and it could only return static HTML documents to the Web client. In order to meet increasing needs for more functions such as outputing dynamic information to the client, the above authors provide a way to extend services and capabilities from the core of the WWW server. The result is CGI.

CGI is a simple interface for running external programs (CGI scripts) under an HTTP server. There are four major ways for servers to communicate with the CGI script. They are listed below. The first three are for CGI script obtaining information from the server and the last one for CGI script sending information to the sever.

Environment variables

Environment variables are variables that are set by the server and accessible by external programs. These variables contain a wealth of information about the server, external programs, and the client's request. They are listed in Table 3.1 [Tittel 96].

The command line

This approach is typically used only for HTML ISINDEX queries. Information can be passed to external programs as command line arguments when the program is executed.

Standard input

While the clients using POST or PUT methods to send requests to external programs, the information will be passed to an external program on standard input by server. If the GET method is used, the data will show up in the QUERY_STRING environment variable [Patchett 97]. The server will also set the CONTENT_TYPE and CONTENT_LENGTH environment variables to the MIME type of the data and the length of the data in bytes, respectively.

Table 3.1 Standard Environment Variables Reference

Environment Variable	Description
AUTH_TYPE	access authentication type
CONTENT_LENGTH	size in decimal number of octets of any attached entity
CONTENT_TYPE	the MIME type of an attached entity, such as: application/octet-stream, text/plain, etc.
GATEWAY_INTERFACE*	revision of the server's CGI
HTTP_(string)	client header data
PATH_INFO	extra path information to be interpreted by CGI script
PATH_TRANSLATED	virtual to physical mapping of file in system
QUERY_STRING	URL-encoded search string
REMOTE_ADDR	IP address of agent making request
REMOTE_HOST	fully qualified domain name of requesting agent
REMOTE_INDENT	identity data reported about agent connection to server
REMOTE_USER	user ID sent by client
REQUEST_METHOD	request method by client, for HTTP, this is "GET", "POST", etc.
SCRIPT_NAME	URI** path identifying a CGI script
SERVER_NAME*	server name; host part of URI**; DNS alias
SERVER_PORT	server port where request was received
SERVER_PROTOCOL	name and revision of request protocol
SERVER_SOFTWARE*	name and version of server software answering the request

* Not request-specific (set for all requests)

** URI : Universal Resource Identifier

Standard output

When external programs are sending information back to the server, which in turn pass it back to the client's browser, the information is just written to standard output. Also the output information should be written in HTML format.

3.1.4 Web-based network monitoring scenario using CGI

Figure 3.2 shows a Web-based network monitoring scenario using CGI. As mentioned above, Web servers pass information to back-end CGI programs via environment variables and STDIN and executes the program (SNMP server program) according to the request. When this program is running, it reads environment variables and data from STDIN and contacts the SNMP agent to get the information back. Then it returns the information in HTML format via STDOUT to the Web server. Finally the Web server returns this HTML file to the Web client.

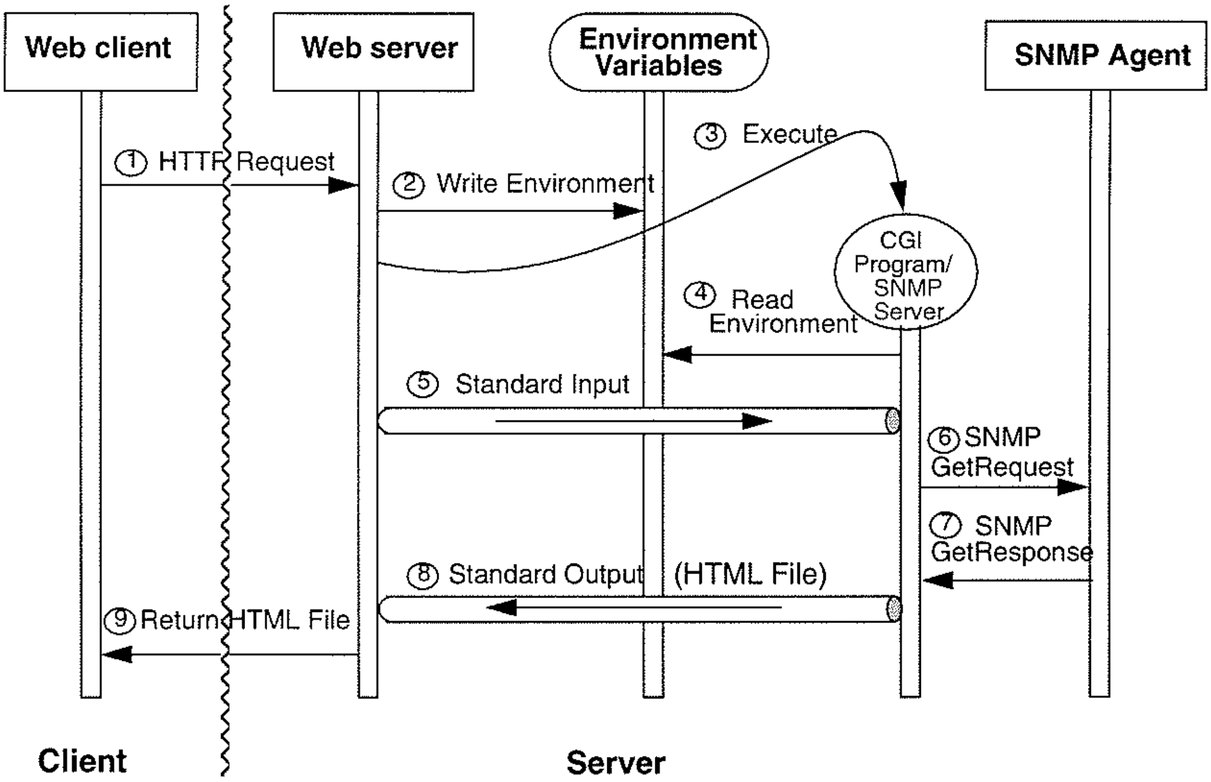


Figure 3.2 Client/Server Interaction Scenario for Network Monitoring

3.1.5 CGI Programming

CGI programs are most commonly written in C, C++, Perl, and Bourne and C shell scripts. Other languages such as TCL and Python are also now becoming popular for CGI programs.

When writing CGI programs, there are two rules which should be followed:

(1) write any data or text for the response to the client to the Standard Output (STDOUT);

(2) output must be preceded with a Content-type line, followed by a blank line.

Both of the above rules are very important. Data which we want the client to receive should be written to the STDOUT, preceded by its MIME type, and the server will do the rest.

The following illustration [Weinman 96] will return "Hello, World!" to the client's browser when invoked by the client. It is written in Perl and illustrated in Figure 3.3.

```
#!/usr/bin/perl

print "Content-type: text/html\n\n"; #print to STANDOUT, followed by a blank line!

# Title it
print "<HTML><HEAD><TITLE>Hello!</TITLE></HEAD>\n";    #print to STANDOUT

# Here's the HTML body
print "<BODY><H1>Hello, World!</H></BODY></HTML>\n";
```

Figure 3.3 A Simple Perl Program for CGI

As mentioned above, the program simply prints the message to the STANDOUT, and it is preceded by a MIME type: text/html, followed by a blank line.

3.1.6 Installing CGI Programs

The following is an introduction of how to install CGI programs on the server and how to run them.

First, the user must have access to a Web server, and write access to a directory named *cgi-bin,* so that CGI programs can be stored in that directory. The CGI programs will be automatically executed by the Web server when it get the request from the client. Secondly, the CGI program is made executable and accessible to the public. This will be done using *chmod* in the Unix system if the programs are written in a shell script or Perl. Also for Perl programs, Perl must be installed and available in your network.

If the CGI programs are written in Java (such as in this application), simply write an executable file in the *cgi_bin* directory with the following line:

```
java program_for_cgi
```

`program_for_cgi` is the name of the class program. In Unix, this file can be written in a shell script. For a PC, a batch file containing this line should be written. In both cases, Java must be available to the current directory.

After this is completed, the CGI programs are ready to be invoked from the applet codes on the Web client side. Further details will be given in Chapter 4.

3.1.7 Summary

From the above we have seen that a CGI program will be invoked each time the Web client sends the request for that service. Each time the Web server passes the parameters to CGI program via Environment Variables and/or Standard Input. After the CGI program is finished, the result will be sent back to the Web server via Standard Output. So although the Web server starts a totally new process for each request, it is clumsy and takes time. Even though CGI is one of the predominant ways of constructing 3-tier Web-client/server model in the Internet today, it introduces a lot of overhead on top of the HTTP protocol.

3.2 Introduction to CORBA

Common Object Request Broker Architecture (CORBA) is defined by the Object Management Group (OMG) to provide a common architectural framework for object-oriented applications. It targets heterogeneous and distributed computing environments and provides the communication mechanism for objects within this environment. According to [Mowbray 97], CORBA "addresses the two most prominent problems faced in the software industry today: (1) the difficulty of developing client/server applications (2) integration of legacy systems, off-the-shelf applications, and new developments".

3.2.1 A little history about CORBA

According to OMG, CORBA is the OMG's answer "to the need for interoperability among the rapidly proliferating number of hardware and software products available today".

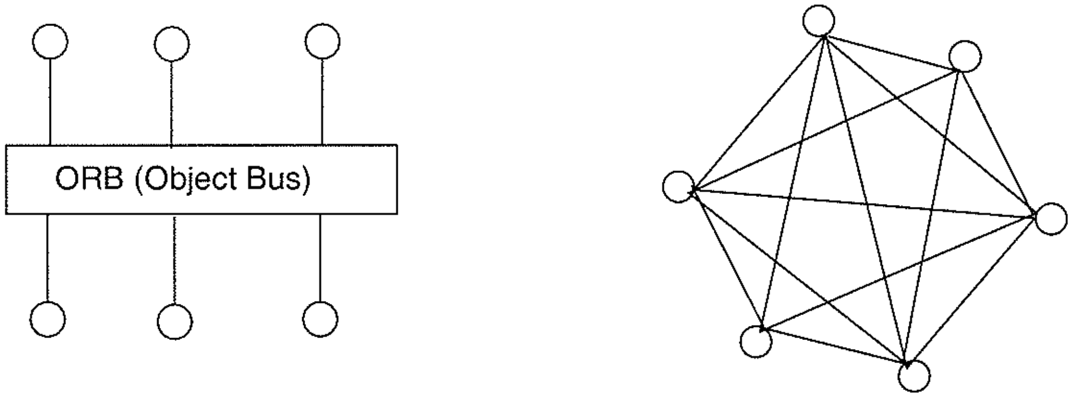
Similar to RFC (Request For Comments) for the Internet Engineering Task Force (IETF), there is the Request For Proposals (RFP) for the OMG. The "Common" in CORBA stands for the merging of two important API proposals, one from HyperDesk and Digital which supports dynamic API and the other from Sun and HP (Hewlett Packard) which supports static APIs. The result is CORBA which has both these two features and will be elaborated upon elsewhere.

In CORBA's history, the first important version is CORBA 1.1, which was introduced by OMG in early 1991. CORBA 1.1 defines the Interface Definition Language (IDL) and the Application Program Interfaces (API). IDL is a neutral language and can be mapped to other high level languages, therefore providing the capabilities for the objects written in different languages to communicate with each other. The API enables client/server object interaction within an Object Request Broker by providing the interface to it.

The next important step made by OMG for CORBA was the coming of CORBA 2.0 in December of 1994. The previous versions of CORBA did not specify a standard protocol through which the ORBs from different vendors could communicate with each other, but CORBA 2.0 does. That protocol is named the Internet Inter-ORB protocol (IIOP) which applies only to TCP/IP based networks.

3.2.2 The Object Request Broker (ORB)

The core of CORBA is the Object Request Broker (ORB), which is also referred to as the object bus or virtual object library. With an ORB, each component only needs to interface itself to the ORB with the help of CORBA's API and then all subsequent interactions among them are handled by the ORB. If there are N components within the application environment, only N interfaces are required to be defined. But without an ORB, custom interfaces must be defined for each interaction between components (writing with different languages, for different systems, etc.), therefore it is a N^2 problem. Figure 3.4 shows this situation.



(a) Components interact each other with an ORB

(b) Components interact each other without an ORB

Figure 3.4 Components Interact Each Other with/without ORB

Actually an ORB is a software component running on the machines with client and server, and provides the communication mechanism between them. It has two basic functions:

- provides object references for the client's request
- marshals and unmarshals the parameters to and from objects

In CORBA, we use an object reference to locate the object within an ORB. The ORB functions like a collection of objects and network resources, integrated with end-user applications, allowing those applications to locate and use objects within the ORB. When a component of an application wants to access a CORBA object, it should first obtain an object reference for that object. When used in CORBA/IIOP, it is called Interoperable Object Reference (IOR). IOR includes the ORB's internal object reference, Internet host address, and port numbers. It is managed by the interoperating ORBs and is not visible to application programmers.

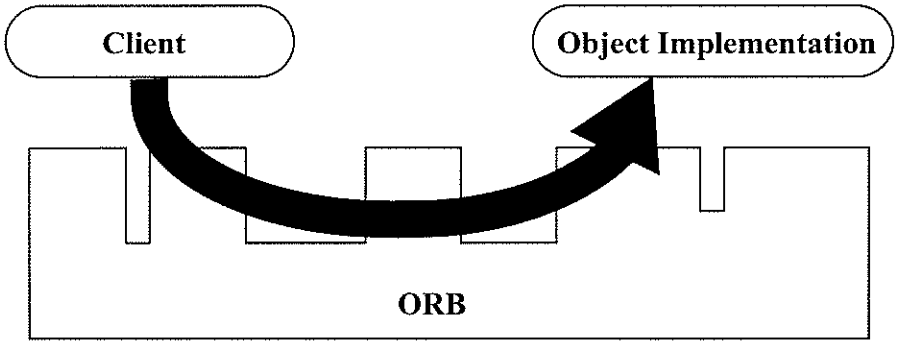


Figure 3.5 A Request Being Sent Through the Object Request Broker

Figure 3.5 shows a request being sent by a client to an object implementation through an ORB[CORBA]. The client is the entity that wishes to invokes an operation on the object, and has no knowledge about where the object is located and how the object is implemented. The ORB is responsible for finding the object implementation for the request (get the IOR for the request), and marshalling and unmarshalling the parameters to and from objects. Here marshalling refers to the process of translating input parameters from the client to a format that can be transmitted across a network. Unmarshalling is the reverse of marshalling; this process converts data from the network to the format that the client understands [Rosenberger 98]. The object implementation is the code and data that actually implements the object.

There are several ways for the client to make a request. The client can send the request to the server via the static IDL Stub, or the Dynamic Invocation Interface (DII). The client can also go directly to the ORB for some functions. Figure 3.6 shows these details.

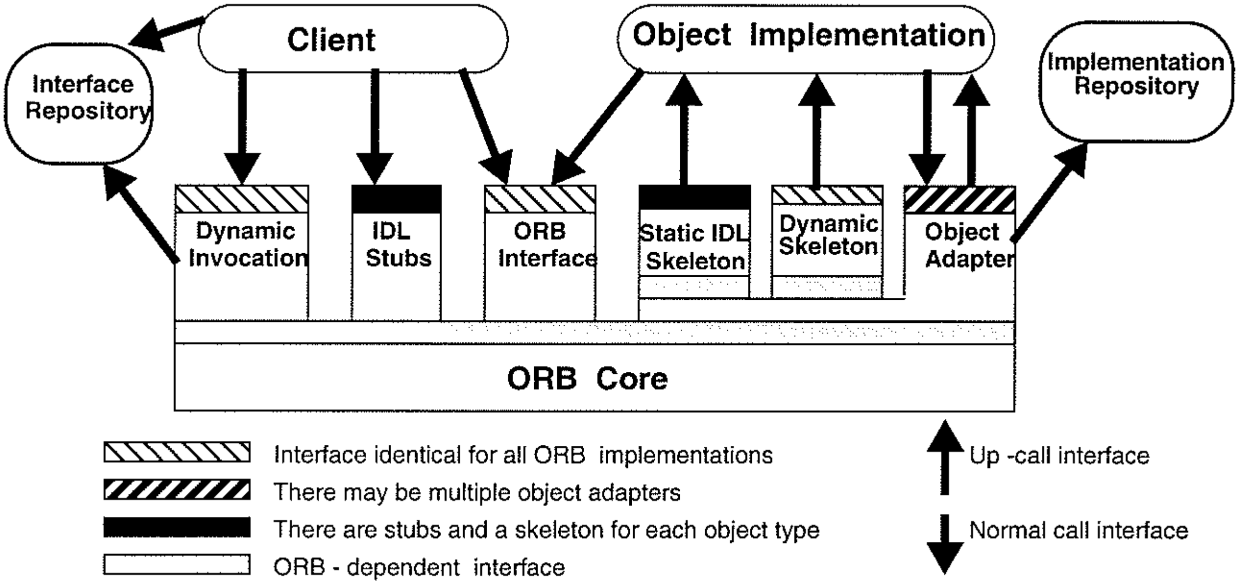

Client
Object Implementation
Interface Repository
Implementation Repository
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core
Interface identical for all ORB implementations
There may be multiple object adapters
There are stubs and a skeleton for each object type
ORB - dependent interface
Up -call interface
Normal call interface

Figure 3.6 The Structure of Object Request Interfaces

The IDL stub provides the static interface to the object service and is object specific (corresponding to specific object interface). Simply speaking, the IDL stub is a small piece of code which is compiled with the client application and gives the client application access to the server objects.

The Dynamic Invocation Interface (DII) provides a way for the client to discover server interfaces dynamically and invoke services from the object that the client does not know about at compiling time. The trade-off is that the complexity is significantly increased.

On the server side, which handles the object implementation, there are static IDL skeletons and

Dynamic Skeleton Invocation (DSI) corresponding to the IDL stubs and DII on the client side. Server skeletons are pieces of code generated when compiling IDL interface definitions. They provide static interfaces to each service exported by the server.

While processing a request, the object implementation may call the object adapter and the ORB interface for services from the ORB. The object adapter handles services such as generation and interpretation of object references and method invocation.

In order to handle the object implementation, the server may access two databases: interface repository and implementation repository. The interface repository provides persistent objects with well defined interfaces that represent the IDL information in a form available at run time. The implementation repository contains information that allows the ORB to locate and activate implementation of objects [CORBA].

3.2.3 OMG Interface Definition Language (IDL)

IDL is another chief component of CORBA and is used to define the types of objects by specifying their interfaces. IDL is not a complete programming language; it is limited to defining interfaces. An interface consists of a set of named operations (functions or methods) so that the clients can invoke them to receive the services.

Since IDL is a neutral language and can be mapped to other high level languages (such as Java, C/C++, Smalltalk, COBOL, etc.), after the object interfaces are defined in IDL, the user can finish the implementation code in other high level languages which can be mapped by IDL. In this way, CORBA separates interface definition from the object implementation and makes it possible to invoke the objects services across languages and operating systems.

The IDL is a descriptive language; it supports C++ syntax for constant, type, and operation declarations. However, IDL does not include any procedural structures or variables; therefore there are no flow-control statements (such as **if-else, while, for,** etc.). The IDL grammar is a subset of C++ (such as supporting naming space, preprocessing features, single and multiple inheritance), with some additional keywords to support the distributed concepts.

The main elements of CORBA IDL are:

- *module* — similar to Java package
- *interface* — CORBA's equivalent of Java interface
- *exception* — CORBA's equivalent of Java exception
- *attribute* — member variable, by which the implementation automatically creates **get/set** operations (**get** for *readonly* attributes, get and **set** for normal attributes)
- operations — CORBA's equivalent of methods
- basic data types (such as: *short, unsigned short, char*, ...)
- constructed types (such as: *array, sequence, struct, enum*, ...)

The sequence type is similar to array type, but the number of elements in an array is fixed and known in advance, the size of sequence can be dynamically changed. So in this way it is similar to Java's vector type. The difference is that in CORBA all values of a sequence must be of the same type or derived from the same type. In Java we can add different objects to the vector.

After writing IDL files, the user can rely on a precompiler which will interpret the IDL files to the language which the user will use to finish the implementation coding. Examples of precompiler include *idl2java* from *VisiBroker for Java, idl2cpp* from *VisiBroker for C++,* etc. The precompiler will also directly generate client stubs and server implementation skeletons. More details about programming in CORBA will be introduced in the next chapter. Figure 3.7 illustrates a simple IDL example.

```
module Bank {
        interface Account {
            float balance();
        };
        interface AccountManager {
            Account open(in string name);
        };
};
```

Figure 3.7 A Simple IDL Program Named Bank.idl from [VBJ33]

The above example provides the Account and AccountManager interface definitions. The keyword **in** in **open** operation means that the value of name should be passed from client to server. According to IDL definition, the parameters within a operation can have a mode that indicates whether the value is passed from client to server (**in**), from server to client (**out**) or both (**inout**). This is different from Java and C++.

3.2.4 Web Client/Server 3-tier model with CORBA

We have already seen that how CGI plays a role in 3-tier Client/Server model and how inefficient it is (the Web server needs to invoke a totally new process for each request, whether it is related or not). On the other hand, CORBA's ORB provides object references for the client's request and marshals and unmarshals the parameters to and from the objects. How does CORBA function in a 3-tier client/server model? Figure 3.8 shows a Web-based network monitoring application in a 3-tier client/server architecture, with CORBA's service from VisiBroker (one of commercial CORBA products) [VisiBroker]. The circled numbers show the message sequence for one of the SNMP applications.

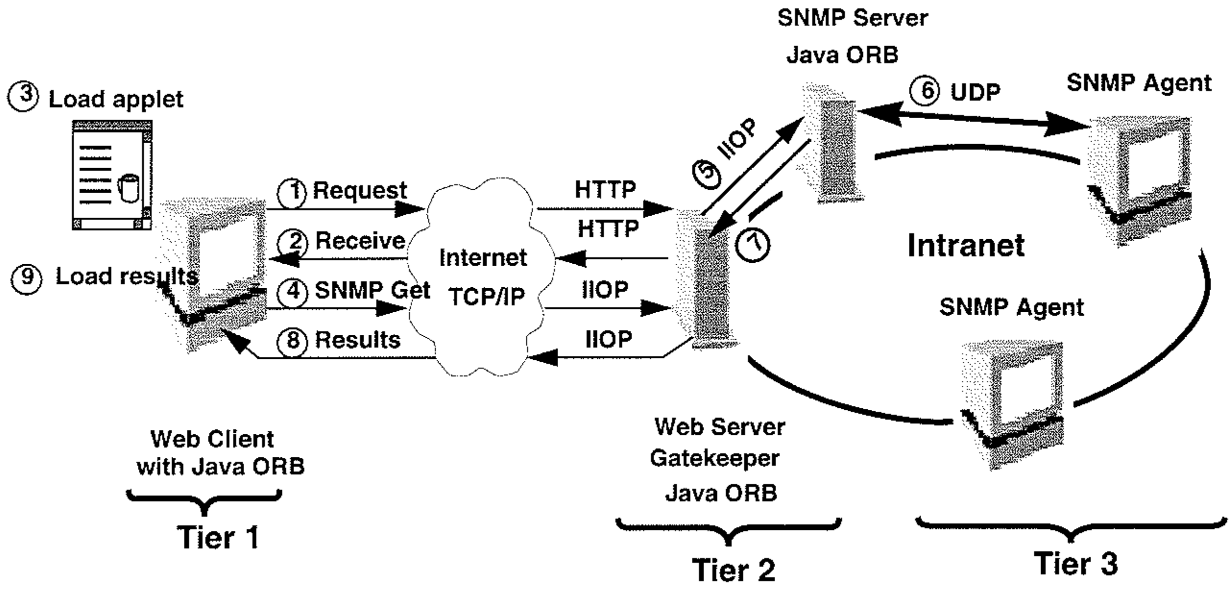


Figure 3.8 3-tier Client/Server Model with CORBA and SNMP Application

(1) The Web client sends the request for Network Management Homepage to the Web server. The communication is via HTTP.

(2) The Web client downloads the HTML page (which includes the reference to the embedded Java applet and Jar files) and Java applet from the Web server. This is also done via HTTP.

(3) The Web browser loads the applet and Jar files into the client machine's memory and starts the applet.

(4) The user clicks the button in the applet's window for receiving the network information, the request is sent to the Gatekeeper (running on the Web server) across the network via ORB/ IIOP.

(5) The VisiBroker Gatekeeper enables the applet to communicate with the SNMP server across networks while still conforming to the security restrictions imposed by Web browsers and firewalls. It serves as a gateway from an applet to server objects [VisiBroker].

(6) The SNMP Server contacts SNMP Agent in the local network for the information.

(7) The SNMP Server returns the result to the Gatekeeper residing on the Web Server.

(8) The Gatekeeper forwards the result automatically to the Web client via ORB/IIOP.

(9) The result is loaded and shown on the client's Web browser.

Note: the session between the Java applet and the SNMP server (or other object server) will persist until either side decides to disconnect.

3.2.5 Summary

From the above, we can reach some conclusions about differences between CORBA's applications and CGI's applications.

(1) For CGI's application, the Web server starts a totally new process for each request which is time-consuming and stateless. Also all messages between client and object server go over HTTP, which involves a great deal of overhead.

(2) For CORBA's application, the Web server and HTTP are involved for transferring the HTML page and Java applet at the beginning of the process. After that the client and object server communicate with each other via IIOP. IIOP introduces less overhead compared to HTTP.

(3) For CORBA's application, the object server keeps running in the back-end of the Web server. It is ready to provide desired service via the designed interface to the clients. Also the session between the applet and object server will persist, and state information can be kept until either side decides to disconnect.

(4) CORBA provides local/remote transparency for method invocation on the server side. It allows clients to directly invoke methods whether the server providing these services is running on the same machine or across a network.

(5) With a distributed object infrastructure, CORBA allows Java applets to communicate with other objects written in different languages across a network.

CHAPTER 4

Constructing Network Monitoring Tools Using CGI

In this chapter, construction of Web-based network monitoring tools using Java, SNMP and CGI is introduced. The SNMP API software package from AdventNet is also introduced.

4.1 Why Java

"Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language." [JavaSoft]. For this reason Java was selected for the network management software development.

As a computer language, Java is a good programming language to use. Since its introduction in May 1995, Java has become the most rapidly adopted programming language in computing history. Advantages of Java include: security, platform independence, full object orientation, multithreading, database access, network programming and Web browser integration. Network management application development can benefit from these features. Java has rich class libraries for network programming which makes it is easy to develop Java applications dealing with TCP and UDP. With the platform independence, the code developed can be run on any machine in the internet. Also with Java applets running on the Web browser, the system administrator can manage their networks remotely. This is because Web browsers are so prolific that they can be found everywhere in the world.

An applet is a mini-program that will run only under a Web browser. It is downloaded automatically as part of a Web page to the client's memory, and executed there. Typically the applet will create some kind of dynamic visual effect within the area of the page that is assigned to it. It provides you with a way to automatically distribute the client software from the server where there is a need to run it in the client side. We have already seen this scenario in Figure 3.1 and Figure 3.6.

4.2 The AdventNet SNMP Package

The AdventNet SNMP Package (version 1.2) is a comprehensive client-server toolkit for network management. It provides a good basic SNMP API library to facilitate building network management solutions which incorporate and integrate SNMP and Java technologies.

This package can be broadly divided into four categories:

1. SNMP Variable Classes
2. SNMP Communication Classes
3. SNMP MIB related Classes
4. Miscellaneous Classes

4.2.1 SNMP Variable Classes

There are several SNMP data types, such as: Integer, Bit String, Octet String, Object Identifier and NULL. In the AdventNet SNMP package, there are classes corresponding to them. First there is an abstract class named **SnmpVar** which is the ancestor of all SNMP variable classes. It contains abstract methods for printing, ASN encoding, ASN decoding, etc. The SnmpVar class has five direct sub-classes. The class hierarchy is shown in Figure 4.1.

There are also other two classes in this category, they are:

- SnmpVarBind: to deal with SNMP variable bindings with Object Identifiers.
- ASNTypes: for ASN parameters and methods used in the package, for encoding and decoding.

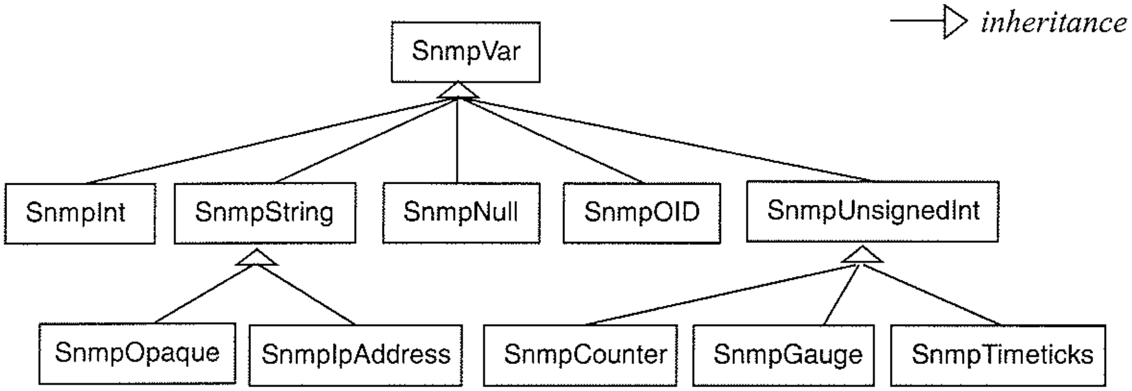


Figure 4.1 The Class Structure of SnmpVar

4.2.2 SNMP Communication Classes

There are five classes within this category dealing with SNMP communication. They are:

- SnmpAPI : to manage sessions created by the user application, manage the MIB modules that have been loaded, and store some key parameters for SNMP communication. This is very important class in the Advent SNMP Package. Before using the SnmpAPI, we should instantiate it and start it.
- SnmpSession : to manage a session with an SNMP peer. It provides functions to open sessions synchronously or asynchronously, send and receive SNMP requests, check for responses and timeouts, and close sessions. We should instantiate and open a SnmpSession instance first, then are able to use it to communicate with an SNMP peer.
- SnmpCallback : used when an asynchronous response to a snmprequest is to be delivered from a separate thread other than the SnmpSession receiver thread.
- SnmpPDU : to provide the variables and methods to create and use the SNMP PDU. The methods include adding variable bindings with specified OID and null variables to the PDU, printing all variable bindings, etc.
- SASClient: added for enabling applets on browsers to work around the security restriction in browsers that don't permit socket access to any host but the applet host. SASClient is not used in the

network management tools developed in this thesis.

4.2.3 SNMP MIB Related Classes

There are also five classes in this category which deal with the SNMP MIB.

- MibModule: provides a way to parse and use the data available in a MIB module file. Each MibModule instance is created from a MIB module file, and you can load and unload MIB modules by creating and deleting these instances. Via parsing the MIB module in the application, we can retrieve the values maintained by the agent.
- MibMacro: used to parse MIB macros, only OBJECT-TYPE and TRAP-TYPE macros are supported. Any parsing of MIB modules would instantiate the macros for the module being parsed.
- MibTrap: used to keep data on trap types defined in a module.
- MibNode: represents a node in the parsed MIB tree. A number of attributes and methods in it are provided to simplify development of applications using the MIB definitions.
- LeafSyntax: used to represent any unique syntax, including textual conventions, that is defined in a MIB module.

4.2.4 Miscellaneous Classes

Some classes do not belong to the above categories. They are:

- SnmpClient: this is an interface used to change the default behaviour on callbacks, authentication, and where to print debugging output. To use it, the user would implement this class and set the client variable in the SnmpAPI instance.
- MibException: this is an exception class that is used to throw MIB parsing exceptions.
- SnmpException: this is an exception class to throw SNMP exceptions, e.g. decode errors.

Next we will see how to develop network monitoring systems based on the AdventNet SNMP classes.

4.3 Constructing Network Monitoring System I Using CGI

4.3.1 Basic Requirements for Monitoring System I

The network monitoring system was designed starting from a simple one, named Network Monitoring System I using CGI (denoted *NMS1_CGI* for short), which only queries the basic information of the System group in MIB-II. As mentioned, the information in the System group in MIB-II is the basis for network management. The requirements for *NMS1_CGI* are:

(1) The user can use a Web browser (such as Netscape, Explorer, etc.) to monitor the specified system in the network. The monitoring system should provide a user-friendly graphical interface.

(2) The user can enter either the machine name or its IP address for monitoring*.

(3) According to the name or IP address, the *NMS1_CGI* should respond with either the information defined in the System group in MIB-II or a warning message when there is an error in the process.

(4) According to RFC1213, the syntax of *sysObjectID* is Object Identifier and syntax of *sysServices* is integer, so it is difficult for a user to directly understand them. The *NMS1_CGI* should interpret them to a more understandable representation.

(5) The query time is 15 seconds. If there is no result after this time, the system will show a timeout message.

Based on the above requirements, the system was designed using an OO approach and will be introduced next.

* Usually there is also a community name that should be entered by user. The community name is used to provide access to the SNMP agent for security reasons (refer to Table 2.6 and Figure 2.7). For this application this is hidden by the using default value: public.

4.3.2 Initial Approach — Use Case Diagram Model

Use case was introduced by [Jacobson 92] in 1992, within a general OO methodology, called OOSE (Object-oriented Software Engineering). Use cases are now a part of UML (Unified Modelling Language) for doing the initial system modelling. Figure 4.2 gives a high-level use case diagram of the *NMS1_CGI*. According to UML, a use case is a sequence of transactions performed by a system in response to a triggering event initiated by an actor to the system. The actor is an object outside the domain of the system that interacts directly with the system. Usually an actor is a human user of the system. We may consider the external system as the actor while it initiates the contact with the application system or gets a value from the application system [Pooley 99].

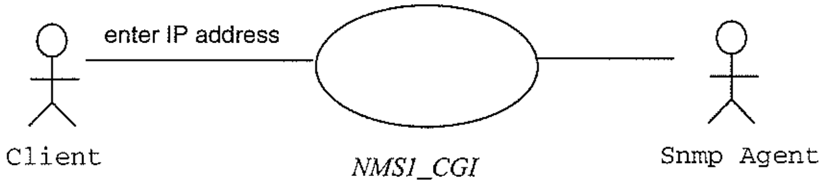


Figure 4.2 The Use Case Diagram of the Monitoring System

Within the figure, the use case itself is represented by an oval and an actor by a stickman. A line connects the actor and the use case which it interacts with. Figure 4.2 shows an abstract model of the *NMS1_CGI*, where one actor is the client who initiates the event, and the other is an SNMP Agent which is contacted by the *NMS1_CGI* and may get a value from it (for the SNMP get and set operations). The use case description of the *NMS1_CGI* is listed in Figure 4.3. It gives a generic, step-by-step written description of the interaction between the actor (such as the client or the SNMP agent in our case) and the use case (*NMS1_CGI*).

Use Case Description of the NMS1_CGI

Normal sequence:

(1) The Web client goes to the Web page of NMS1_CGI, enters the machine's assigned name or its IP address, clicks the Query button.

(2) According to user's request, NMS1_CGI builds an SnmpPDU and sends it to the queried machine.

(3) The SNMP agent in that machine checks its MIB and replies with the requested information.

(4) The NMS1_CGI interprets the information obtained from the SNMP agent into the desired format, and displays them on the client's browser.

Alternate sequence:

If the machine which the user checks does not exist in the network or does not support SNMP service, NMS1_CGI will return a warning message in the client's browser. If there is a typing error, the client can enter the machine name again for checking.

Figure 4.3 The Use Case Description of the NMS1_CGI

After doing this, the next step is to convert the use case to an OOSE Ideal Object Model (or Analysis Model), which describes an ideal system that would accomplish all of the requirements identified by the use case diagram [Jacobson 92]. The key to this approach is to assume that there are three broad types of objects: *Entity objects*, *Interface objects* and *Control objects*. UML provides *stereotypes* to attach extra classifications for different models, and in this way UML is made extensible. So for the above three object types, there are three "stereotype" names corresponding in UML, they are: *boundary class*, *entity class* and *control class* [Rumbaugh 98].

A *boundary class* describes objects that mediates between a system and outside actors, such as an order entry form or a sensor. The *entity class* is the class which holds the element information which the system deals with. The *control class* is responsible for dealing with transactions among the interface classes and entity classes. The classes in an ideal object model can be related to each other by means of association lines, which means that the objects of the two classes will send messages to each other. An ideal object model of the *NMS1_CGI* is illustrated in Figure 4.4.

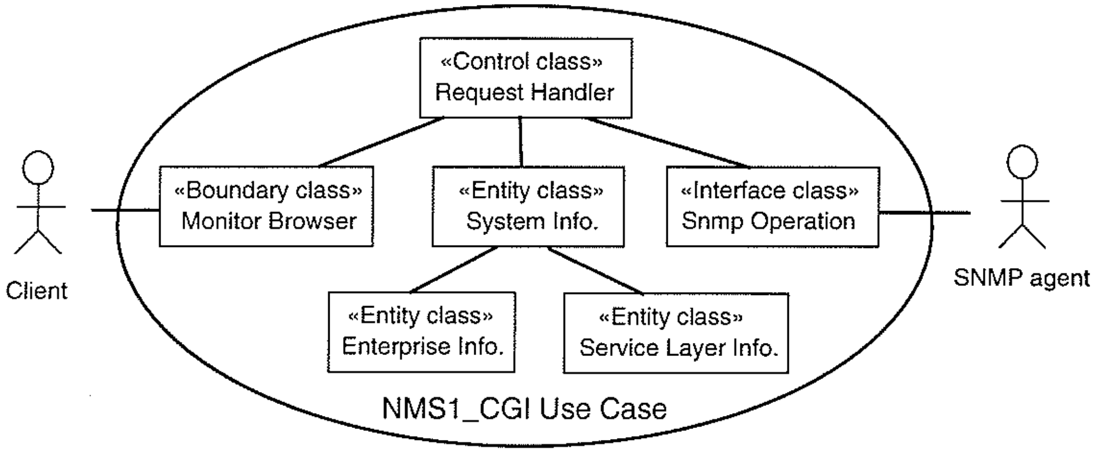


Figure 4.4 An Ideal Object Model of the NMS1_CGI

4.3.3 Class Diagram Design — Extending the Initial Object Model

Based on the ideal object model of the *NMS1_CGI*, the class diagram is designed. It is illustrated in Figure 4.5. The system classes with the main attributes (variables in Java) and operations (methods in Java) are illustrated in Figure 4.6. We can see that there is an Web Server object in Figure 4.5. This is because that there is no way for *ClientCGI* to *contact RequestHandler* directly while using CGI. While using CGI, the HTTP request sent by Web client first goes to Web server, and then Web server invokes and executes the CGI program. In Figure 4.6, the class RequestHandler was redesigned to include the functions of the SystemInfo class.

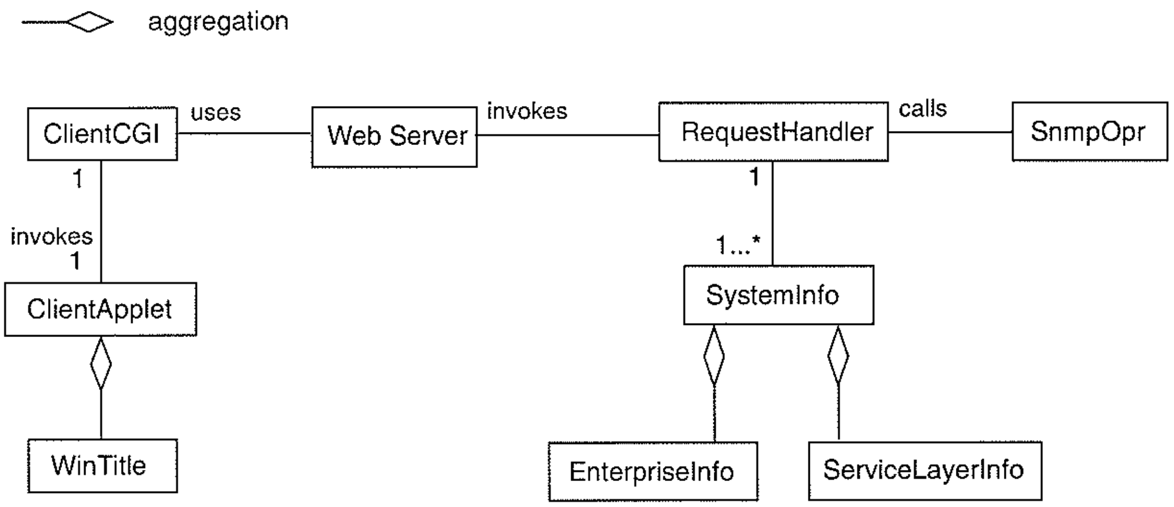


Figure 4.5 Initial UML Class Diagram of the NMS1_CGI

- *ClientApplet:* a Java applet which provides the main window in the Web browser for the Web client to interact with. The window is implemented using java.awt.GridBagLayout class.
  - *Attributes:*

hostname	a string represents the managed node name or its IP address
header	an object of WinTitle represents the header bar in the applet main window
message	information returned by RequestHandler
clientCgi	an object of ClientCGI (see class ClientCGI)

  - *Operations:*

init	called by applet viewer when Java launches the applet for the first time
actionPerformed	invoked when an action occurs in the applet window
refresh	refreshes the window to the default situation

- WinTitle: the class responsible for constructing the window title bar
  - *Attributes:*

textColor	the text colour in the window bar
textFont	the text font in the window bar

  - *Operations:*

draw	draw the graphic bar according to the specified text font and colour

- ClientCGI: the class responsible for interfacing the client application to the server side program via the CGI protocol
  - *Attributes:*

socket	an object of java.net.Socket, used to make a connection to RequestHandler
r_message	returned message from RequestHandler

  - *Operations:*

getMessage	the method for getting the queried message
getWarns	getting the warning message
servLayInfo	an object of ServiceLayerInfo (see class ServiceLayerInfo)
enterpriseInfo	an object of EnterpriseInfo (see class EnterpriseInfo)
snmpopr	an object of SnmpOpr (see class SnmpOpr)

- RequestHandler: responsible for handling the query from the Web client, instantiating an object of SnmpOpr to contact the SNMP agent to complete the SNMP operation, it is also the server side CGI interface responsible for interfacing the server to the client
  - *Attributes:*

message	the string message about the system getting from snmp
warning	the warning message got from the queried node

  - *Operations:*

main	responsible for the main operations of the RequestHandler

Figure 4.6 The System Classes and their Main Attributes and Operations

- Enterpriselnfo: responsible for interpreting the OID of sysObjectID in system group of MIB-II to the text information about the vendor who provides the management subsystem in the queried node (according to file "enterprises.oid" provided by Sun Microsystems)
  - *Attributes:*

oid_name	the string represents the enterprise name and its sub-ID

  - *Operations:*

getName	interprets the OID to the enterprise name and the sub-ID

- ServiceLayerlnfo: interprets the integer number of sysServices in system group of MIB-II to the text information about the set of services the queried node offers (refer to Chapter 2 for more information)
  - *Attributes:*

servNum	the integer number of sysServices returned by the SNMP agent
servlnfo	the text information about which service layers the system operates on

  - *Operations:*

calculate	for each servNum, calculates the service layers and returns this information

- SnmpOpr: responsible for SNMP operations such as building SNMP GetRequest PDU, sending it to the remote managed node, it is the class providing the interface to the SNMP agent.
  - *Attributes:*

api	an object of (advent)Snmp.SnmpAPI, needs to be instantiated and started in order to use the Snmp Package (see 4.2.2 for more information)
session	an object of (advent)Snmp.SnmpSession, needs to be instantiated and opened for sending requestPDU (see 4.2.2 for more information)
module	an object of (advent)Snmp.MibModule (see 4.2.3 for more information)
peer_name	the string represents the name of to be managed node
pdu	an object of (advent)Snmp.SnmpPDU
varbind	an object of (advent)Snmp.SnmpVarBind, contains an object identifier and an SnmpVar (the information returned by SNMP agent)
var	an object of (advent)Snmp.SnmpVar, can be any type of SNMP variables
errorMsg	the error information for the SNMP operation

  - *Operations:*

openSession	tries to open an SnmpSession and catch exceptions
buildPDU_1	instantiates an SnmpPDU with an instance of SnmpAPI and a request type, such as SnmpAPI.GET_REQ_MSG
buildPDU_2	adds variable bindings with specified OID and null variable to PDU
sendPDU	tries to send SnmpPDU synchronously via SnmpSession - returns response SnmpPDU and throws SnmpException on failure
getWarns	returns the errorMsg

Figure 4.6 The System Classes and their Main Attributes and Operations (con't)

4.3.4 Sequence Diagram Design

Usually it is good to diagram a system in a way that indicates the order in which things occur. UML offers several ways that can be used to show the dynamics of a system. Sequence diagrams show how the objects in a system send messages at a macro-level to one another. Currently sequence diagrams are one of the most popular UML diagrams and are used in conjunction with class diagrams to capture most of the information about a system.

Figure 3.2 illustrates the client/server interaction scenario for network monitoring. It is a sequence diagram with emphasis on the platform objects and give details about the message transmissions in the CGI scenario. Based on Figure 3.2, Figure 4.3, Figure 4.5 and Figure 4.6, the sequence diagram of the *NMS1_CGI* system was designed and is depicted in Figure 4.7. The step-by-step explanation of it is given in the following:

(1) Web user enters the queried machine name and clicks the query button.

(2) The applet collects the data which the user entered and forwards it to the object of ClientCGI.

(3) The object of ClientCGI (subsequently just the class name is used) constructs a socket to the Web server and then outputs the CGI script name and queried data to it.

(4) The Web Server parses the request from the client and then invokes the CGI program for it

(5) The RequestHandler (invoked by the Web server gets the data via the environment variables and Standard Input) parses the request parameters and invokes the SnmpOpr.

(6) The SnmpOpr then contacts the SNMP agent for the local information.

(7) The SNMP agent returns the information.

(8) SnmpOpr forwards the result back to the RequestHandler.

(9) — (12) Then RequestHandler contacts the EnterpriseInfo and ServiceLayerInfo separately to interpret two of the values (enterprise OID and service layer).

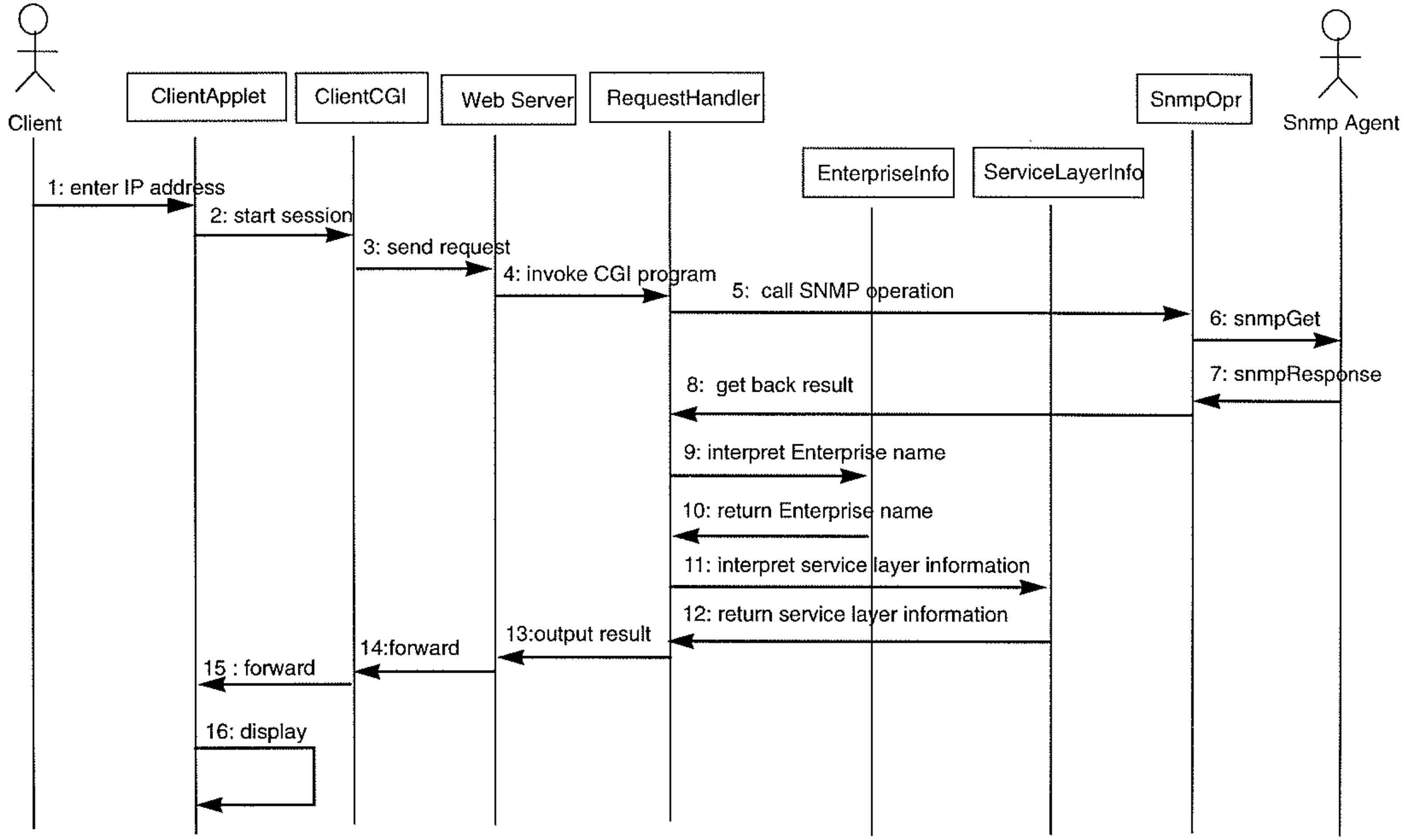


Figure 4.7 The Sequence Diagram of the NMS1_CGI

(13) RequestHandler then constructs a string with all requested information and returns it to the Web Server (via Standard Output).

(14) Web Server forwards the result back to the ClientCGI.

(15) ClientCGI divides the string into the needed information and returns this information to the ClientApplet.

(16) ClientApplet displays the result in the applet window, either useful information or exceptions.

4.3.5 The Code Development — CGI part

This section illustrates how this system is implemented in Java. In this section the code examples illustrating how to program for CGI are provided.

When a Web client clicks a link that points to a URL which locates a program on the Web server side, the server will invoke the program in the URL and transfer the request parameters via *the Standard Input* and *Environment Variables* to this program according to CGI protocol. After the program finishes, it will return the results via *the Standard Output* to the Web server and then Web server returns the results to the Web client. In the above scenario, HTTP client and server use the Internet's MIME (Multipurpose Internet Mail Extensions) data representations to describe (and negotiate) the contents of messages [Borenstein 93].

According to the HTTP protocol [Berners-Lee 96], the HTTP client and server must negotiate their data representations each time a connection is made. A typical HTTP request consists of the three parts, their syntax are shown below:

(1) the request line: `<method> <resource identifier> <HTTP version><crlf>`

- method is a HTTP method, such as GET or POST
- resource identifier specifies the name of the target resource

- HTTP version specifies which version the client uses, such as HTTP/1.0
- *<crlf>* stands for *carriage-return/line-feed*

(2) the request header fields: `<Header>: <value><crlf>`

- Header stands for a header name, and next is its value, followed a *<crlf>*
- with multiple use of these lines, the last one should be ended with two *<crlf>*s

(3) the entity body

- this part is used by clients to pass bulk information to the server

HTTP GET is used to retrieve the specified URL and returns the data to the client. POST is similar to GET, the difference is that they operate differently. You can not send data that is longer than 256 characters (or 1024 characters, system dependent) to the HTTP server via the GET method, so the use of the POST method is recommended [Orfali 97]. In this application, the POST method was used in the CGI program. The Figure 4.8 shows the code of ClientCGI.java. It uses the Java Socket class to construct a connection between the client and server. Referring to the explanation of HTTP request above, the important parts in the code are highlighted.

```
import java.lang.*;
import java.util.*;
import java.net.*;
import java.io.*;

class ClientCGI {
    Socket socket = null;
    private String[] msg = {"", "", "", "", "", "", "", "", ""};
    private boolean MSG = false, ERR = false;
    private String script = "/cgi-bin/jizong/snmpGet";
    private String line = "";
    ...

    public ClientCGI(String str) {
       String data = new String("hostname " + str);
       try {
           socket = new Socket("home.ee.umanitoba.ca", 80);
           DataOutputStream ostream = new DataOutputStream(socket.getOutputStream());
           DataInputStream istream = new DataInputStream(socket.getInputStream());

           ostream.writeBytes("POST " + script + " HTTP/1.0\r\n"
                 + "Content-type: application/octet-stream\r\n"
                 + "Content-length: " + data.length() + "\r\n\r\n");
           ostream.writeBytes(data);
           ostream.close();

           line = istream.readLine();

           if(line.equals("Warning!")) {
               ...  // do something
           } else if(line.equals("Message!")) {
               ...  // do something
           }
           istream.close();
       } catch (Exception e) {
           ...  // do something
       }
    }
    ...
}
```

Figure 4.8 The Part of ClientCGI.java with Usage of Java Socket

We can also take the advantage of the URL and URLConnection classes in java.net. With the use of URL and URLConnection classes, the program looks simple. Figure 4.9 shows the part of the code ClientCGI.java using URL and URLConnection. You can see that there is no need for you to send your request with the HTTP header fields but rather directly send your request to the server via the URL connection. In this way things are clear and easy.

Compared to the program ClientCGI.java, the server side CGI program looks simple. It simply takes the request from the Standard Input and returns the result via the Standard Output. Figure 4.10 illustrates this simplicity by showing the part of RequestHandler.java.

```
public ClientCGI(String str) {
   try {
       URL snmpServer = new URL("http://home.ee.umanitoba.ca/" +
                 "cgi-bin/jizong/snmpGet");
       URLConnection connection = snmpServer.openConnection();
       connection.setDoOutput(true);

       PrintStream ostream = new PrintStream(connection.getOutputStream());

       ostream.println(str);
       ostream.close();

       BufferedReader istream = new BufferedReader(
                 new InputStreamReader(connection.getInputStream()));

       line = istream.readLine();
       ...
   }
   ...
}
```

Figure 4.9 Part of the ClientCGI.java Utilizing of the Java URL Class

```
class RequestHandler {
    public static void main(String[] args) {
       String line = null, error = null, rdata;
       RequestHandler request_handler = new RequestHandler();

       try {
           BufferedReader istream = new BufferedReader(new InputStreamReader(System.i
           line = istream.readLine();

           if(line != null ) {
               ...; // invoke the method to query data, assign the data to rdata
               System.out.println("Content-Type: text/plain\n\nMessage!\n" + rdata);
           } else {
               System.out.println("Content-Type: text/plain\n\nWarning!\n + error");
           }
           istream.close();
       } catch (Exception e) {
           System.out.println("Content-Type: text/plain\n\nWarning!\n" + e);
       }

       System.exit(0);
    }
    ... // methods for querrying the machine in the network
}
```

Figure 4.10 Server-side CGI Program — Part of Code: RequestHandler.java

4.3.6 The Code Development — SNMP part

Based on the AdventNet SNMP package (see 4.2.1 ~ 4.2.3 for more information) and Figure 4.6 (listing the attributes and operations of SnmpOpr class), SnmpOpr.java was developed which is responsible for the SNMP operations in my application.

First consider the constructor of SnmpOpr. According to AdventNet SNMP package, any application that uses this package should instantiate and start the SnmpAPI class in the beginning. After that, the application should load the MIB modules in order to retrieve the expected values in the SNMP agent, and then instantiate and open a SnmpSession instance to be able to use it to communicate with an SNMP agent. According to [AdventNet], you can have as many sessions as you like, but be aware that each session is a thread, and you may not want to keep opening sessions unless you need them. Figure 4.11 shows these concepts combined with the attributes (variables) of the class.

```
class SnmpOpr {
    private MibModule   module = null;
    private SnmpOID     oid;
    private SnmpPDU     pdu = null, re_pdu = null;
    private SnmpVarBind varbind = null;
    private SnmpVar     var = null;
    private SnmpSession session = null;
    private SnmpAPI     api = null;
    private byte    command;
    private String  errMsg = "";
    private boolean estat = false;
    ...

    public SnmpOpr(String host, String community) {

       // Instantiate and start SnmpAPI
       api = new SnmpAPI();
       api.start();
       command = api.GET_REQ_MSG;  // change this for diff. operation

       // Load the MIB Module
       try {
           module = new MibModule("rfc1213-MIB", api, api.DEBUG);
       }
       catch (Exception e) {
           errMsg = "Exception: Reading/Parsing MIB URL: " + e;
           return;
       }

       // Instantiate SnmpSession
       session = new SnmpSession(api);
       session.peername = host;
       session.community = community;
       session.remote_port = 161;
       session.timeout = 15000;  // 15 seconds
       session.retries = 0;

       openSession();
    } // end of snmpOpr()
```

Figure 4.11 The Constructor for SnmpOpr.java

The constructor invokes the openSession() method which is a simple method that takes care of opening a SNMP session and catching the exceptions. Figure 4.12 illustrates this.

```
    // Open session
    private void openSession() {
       try {
           session.open();
       } catch (Exception e) {
           errMsg = "Unable to open SnmpSession: " + e.getMessage();
           estat = true;
       }
    }
```

Figure 4.12 The openSession Method of SnmpOpr.java

Now we need to see how to build a SNMP PDU and use it to send the request to the remote SNMP agent. We should first instantiate it and associate a command with it. This is illustrated in buildPDU_1 in Figure 4.13. The command constants are defined in the SnmpAPI class in byte format, such as *GET_REQ_MSG, SET_REQ_MSG,* etc. which respectively correspond to SNMP GET and SET operations.

After that, we need to add the specified OIDs to the PDU which will be sent. Referring to Chapter 2, all SNMP variables or objects are named according to the type OBJECT IDENTIFIER of ASN.1. Correspondingly, there is SnmpOID class in AdventNet SNMP package which is used to create and interpret Object IDs and relate SNMP variables to MIB data. The method buildPDU_2 in Figure 4.13 shows this picture.

```
// build SnmpPDU
private void buildPDU_1(byte cmd) {
   pdu = new SnmpPDU(api);
   pdu.command = cmd;
}

private void buildPDU_2(String poid) {
   SnmpOID oid = new SnmpOID(poid, api);
   pdu.addNull(oid);
}

//send SnmpPDU synchronously
private SnmpPDU sendPDU() {
   SnmpPDU response_pdu = null;

   try {
       response_pdu = session.syncSend(pdu);
   } catch (SnmpException e) {
       errMsg = "Sending PDU: " + e.getMessage();
       return null;
   }

   if (response_pdu == null) {
       ...
   } else {
       return response_pdu;
   }
}
```

Figure 4.13 The buildPDU_1, buildPDU_2 and sendPDU Methods in SnmpOpr.java

After we have opened an SNMP session and built a PDU, it is time for us to send it via the opened session. There are two ways to send the PDU, one is for sending and receiving PDU (messages) synchronously and the other asynchronously. My application applies sending and receiving PDU synchronously. In this case, it first sends the PDU via the opened session and waits until a response is received or a time out occurs. The method sendPDU in Figure 4.13 illustrates this scenario.

For more information, please refer to AdventNet SNMP Package Documentation [AdventNet].

4.3.7 Putting the system to work

After developing and debugging the codes, it is time now to put them to work. Refer to Chapter 3, the Web-based network monitoring is an application which will be used in 3-tiered client/server architecture. So first there should be an HTML file which can be downloaded by the Web client. A simple one is depicted in Figure 4.14 which embeds with the Java applet class and JAR file.

```
<HTML>
<HEAD><TITLE>SYSTEM MONITORING</TITLE></HEAD>

<BODY> <CENTER>

<APPLET CODEBASE="(give your code path here)", CODE="ClientApplet.class"
   archive=ClientAppletJar.jar width=510 height=360>
  <B>Sorry, your web browser should support Java1.1.</B> </APPLET>
</CENTER>
</BODY>
</HTML>
```

Figure 4.14 The Simple Code of NMS1_CGI.html

JAR stands for Java ARchive which is used to archive multiple files (such as Java class files) into a single JAR file. In this case, the browser only needs to make one connection to the Web server to download the JAR file for running the Java applet. Using JAR saves download time. In my application, I use JAR to archive all the class files of the client side to a single Jar file for downloading, and put this Jar file and HTML file into a folder which can be accessed by the Web client.

Next the following are placed in the cgi-bin folder, which can be located by the Web server for CGI applications,

(1) the server side class files, such as *RequestHandler.class, SnmpOpr.class, etc.*

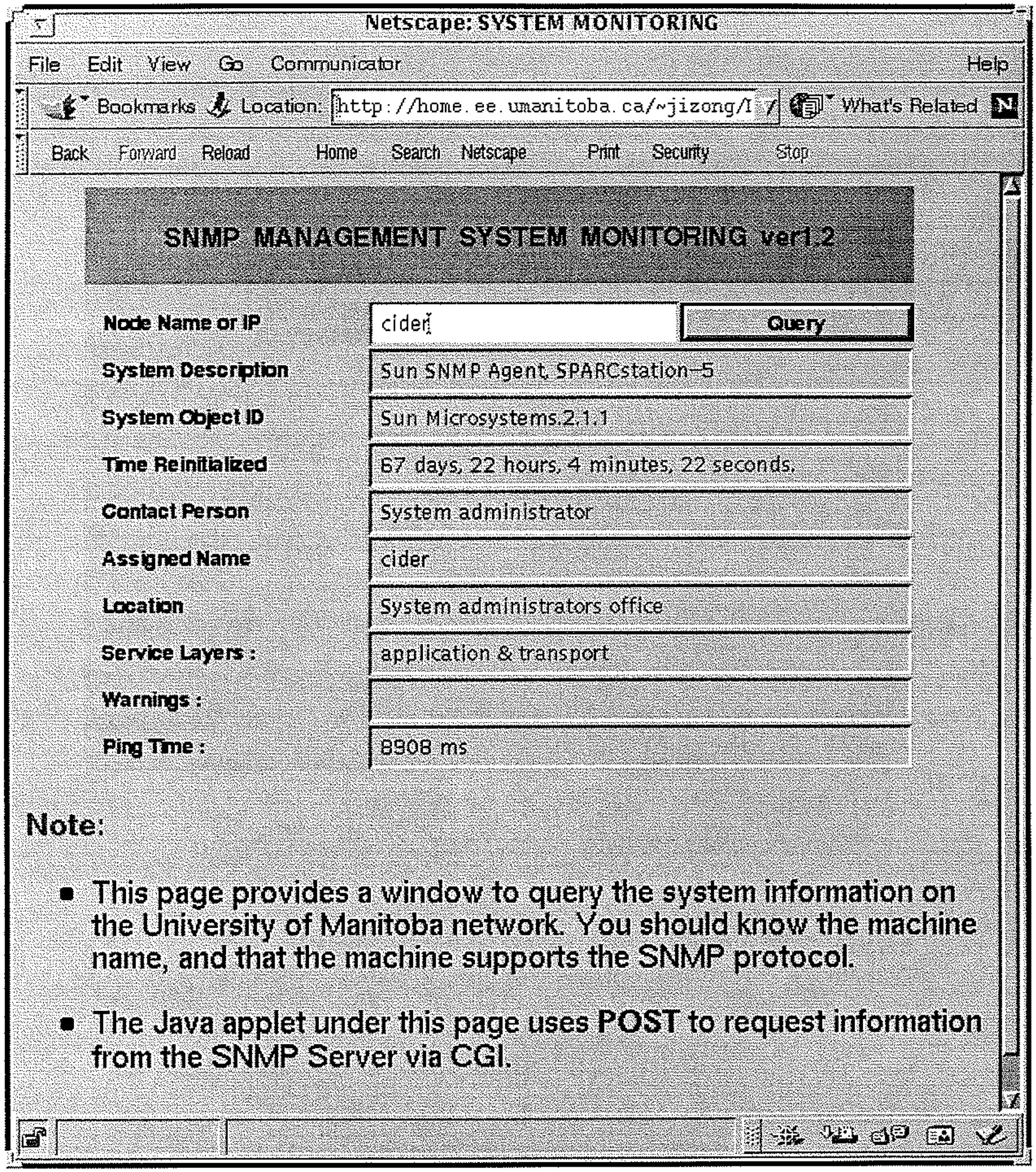
(2) the AdventNet SNMP package

(3) a shell script file named *snmpGet* with one line of code:

```
/usr/bin/java RequestHandler
```

Next, make sure the elements which will be managed by the *NMS1_CGI* are capable of running an SNMP agent. In our ee-network, the machines which are running Solaris 2.6 are automatically running an SNMP daemon, named *snmpd.* For PCs which are running Window NT and connected to our ee-network, we can install and start the SNMP service from the Service Package 3.

Figure 4.15 shows the ClientApplet in the Web browser. Since *cider* is a workstation in our ee-network, there is no need to add the full domain name to it. We can also use the IP address instead. If there is something wrong while querying, the warning line will show a warning message.



SNMP MANAGEMENT SYSTEM MONITORING ver1.2

Node Name or IP	cider
System Description	Sun SNMP Agent, SPARCstation-5
System Object ID	Sun Microsystems.2.1.1
Time Reinitialized	67 days, 22 hours, 4 minutes, 22 seconds.
Contact Person	System administrator
Assigned Name	cider
Location	System administrators office
Service Layers :	application & transport
Warnings :	
Ping Time :	8908 ms

Note:

- This page provides a window to query the system information on the University of Manitoba network. You should know the machine name, and that the machine supports the SNMP protocol.
- The Java applet under this page uses **POST** to request information from the SNMP Server via CGI.

Figure 4.15 The Graph Interface of NMS1_CGI in Web Browser

4.4 Constructing Network Monitoring System II Using CGI

After the long process of constructing *NMS1_CGI*, the construction of a more complicated monitoring system was undertaken, named Network Monitoring System II (denoted *NMS2_CGI* for short). The following are the basic requirements for it.

4.4.1 Basic Requirements for Monitoring System II

(1) The user can use a Web browser (such as Netscape, Explorer, etc.) to monitor the specified system in the network, same as *NMS1_CGI.*

(2) The user can enter either the machine name or its IP address for checking, same as *NMS1_CGI.*

(3) If user clicks the Help button, this should activates a pop-up window with the usage information for this monitoring tool.

(4) If user chooses an item from the choice list, this should activates an pop-up window with the description for that item according to RFC 1213.

(5) If user wants to get the information related to interface group and ip group, he/she can click "Get Text" to get the text information first and then click "Get Graph" to get two real time graphs about the total bytes in and out, IP packets in and out of the machine.

(6) While in the "Get Graph" operation, the system should open a pop-up window to provide the text information for the graph values such as total bytes in and out, IP packets in and out of the machine.

(7) While in "Get Graph" operation, there is a 15 seconds interval between each polling.

(8) While in "Get Graph" operation, user can click button "Terminate" to stop drawing the graph.

(9) There is a field to show the average poll time of 100 pollings for the "Get Graph" operation. This will be used for performance comparison of CGI and CORBA implementation.

(10) If an exception happens while monitoring, there should be an pop-up window to show the warning messages.

(11) The popup window can be closed by clicking the "Close" button.

(12) The time for each check is 15 seconds. If there is no result after this time, the system will show a time-out message in the warning popup window.

4.4.2 Class Diagram Design

Based on the above requirements, the class diagram of *NMS2_CGI* was designed and is depicted in Figure 4.16.

The system *NMS2_CGI* is composed of eight classes which represent the type of objects involved in the system design. They are *ClientApplet, PopupWin, WinTitle, GraphCanvas, GraphThread, ClientCGI, RequestHandler* and *SnmpOpr.* Some of them are similar as the ones in *NMS1_CGI.* Their functions are detailed below.

ClientApplet is a Java applet which provides the main window in the Web browser for the Web client to interact with. The window is composed of TextFields (for host name, community name and average poll time), TextAreas (giving text information for queried OIDs), Choices (the choice for a list of OIDs), Buttons and Canvases (for drawing the real time graphs). According to the different actions performed by the client, the object of this class will provide "Help", "Get Text", "Get Graph", "Terminate" and "Choice" (provide a description for an item selected from the choice list) operations. Among them, "Get Graph" is more complicated because it will start a thread to contact the *RequestHandler* separately and draw the figures according to what values it receives. This operation can be stopped by clicking "Terminate". This window is also responsible to open the pop-up windows for giving help information, text information for graph values, OIDs description and system warnings (if an exception happens while processing).

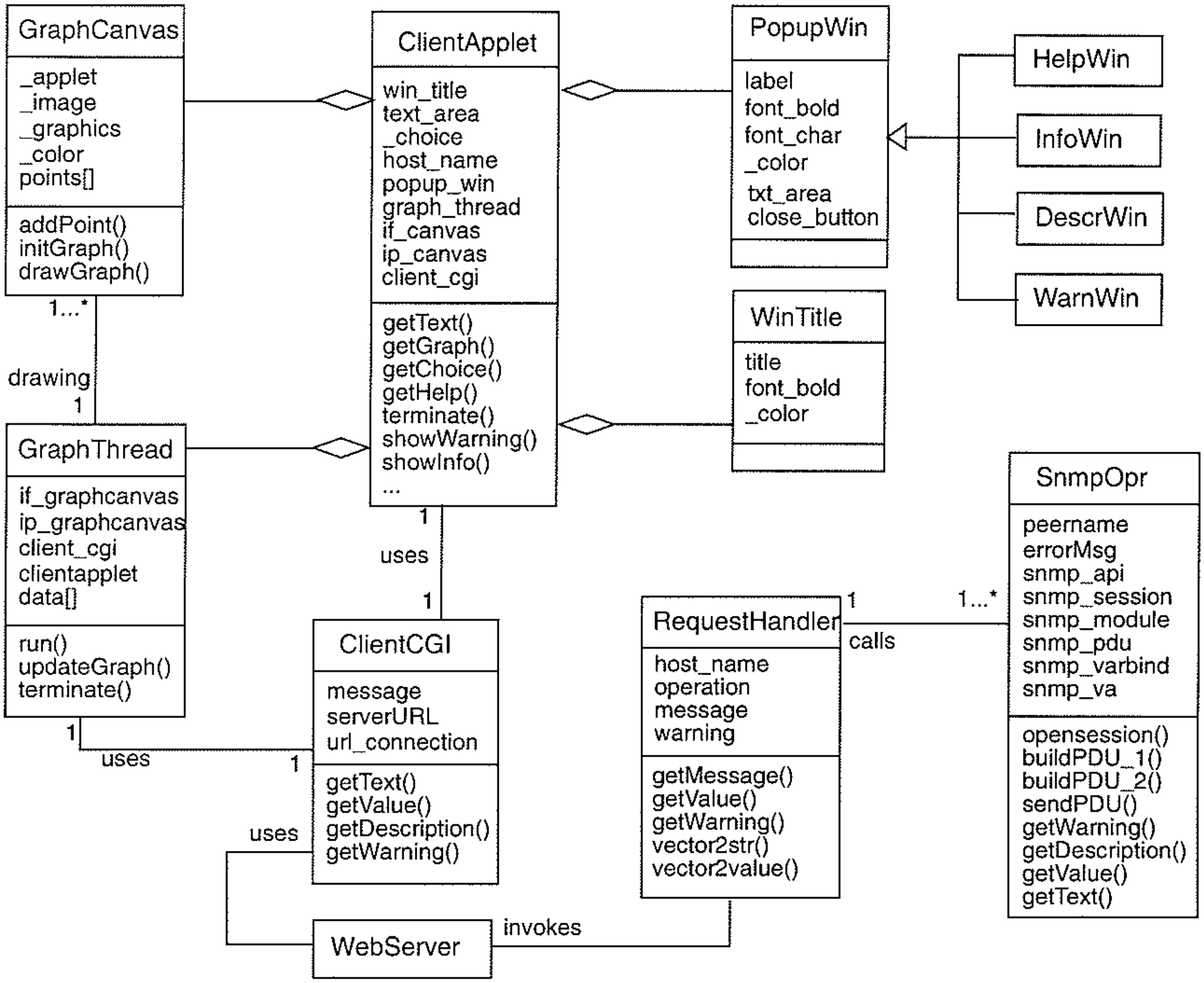


Figure 4.16 Class Diagram of the NMS2_CGI

WinTitle is a class responsible for construction of the window title bar. You can construct a window title by inputting the title name, font, background colour, etc. It is same as the one for *NMS1_CGI.*

PopupWin is a class for constructing a pop-up window to give more information. It is inherited from java.awt.Frame and composed of a label, text-area, close-button, etc. It has four child objects: *HelpWin, InfoWin, DescrWin* and *WarnWin* which are used by *ClientApplet* to provide help, text information about graph values, descriptions about OIDs and warning messages respectively.

GraphCanvas is a class for building the canvas in the applet to draw the graphics. It is invoked by *GraphThread* to add points (the points values) on the canvas in order to get the real time graphs. It is also used by *ClientApplet* to construct the canvas in the applet.

GraphThread is a class for creating a thread to draw the graphics in an object of *GraphCanvas*. It is started by the object of ClientApplet corresponding to the actions taken by the client, and contacts the *RequestHandler* via *ClientCGI* itself to get the data values such as total bytes in and out of an interface or IP packets in and out of the queried machine.

ClientCGI is the class responsible for interfacing the client application to the server side program via the CGI protocol. It creates an URL which points to the CGI script in the Web server side and then constructs the URLConnection to this script. In this way it can output the data to the server and input the data from the server.

RequestHandler is the class which is responsible for handling the query from the Web client, instantiating an object of SnmpOpr to contact the SNMP agent to complete the SNMP operation. It is also the server side CGI interface responsible for interfacing the server to the client

SnmpOpr is the class responsible for SNMP operations such as building the SNMP GetRequest PDU and sending it to the remote managed node. It is the class providing the interface to the SNMP agent.

4.4.3 Sequence Diagram Design

As mentioned before, UML provides sequence diagrams to show the dynamics of a system at a macro-level, which focuses on how the objects in a system send messages to one another. Based on the above development, the sequence diagrams of the system according to different actions taken by an actor (Web client) were designed and they are illustrated in Figure 4.17, 4.18, 4.19.

Figure 4.17 shows the sequence diagram for "Get Text" operation of the *NMS2_CGI*. The step-by-step explanation of it is given below.

(1) The Web user enters the queried machine name and clicks the "Get Text" button.

(2) The applet collects the data which the user enters and forwards it to the object of ClientCGI.

(3) The object of ClientCGI (subsequently just the class name will be used) transfers the queried data to the Web server.

(4) The Web Server parses the request and then invokes the CGI script for it.

(5) The RequestHandler (invoked by the Web server gets the data via the environment variables and Standard Input) parses the request parameters and invokes the SnmpOpr.

(6) The SnmpOpr then contacts the SNMP agent for the local information.

(7) SNMP agent returns the information.

(8) SnmpOpr forwards the result back to the RequestHandler.

(9) RequestHandler then constructs a string of the result and returns it to the Web Server.

(10) Web Server forwards the result back to the ClientCGI.

(11) ClientCGI forwards the result to the ClientApplet.

(12.1)If the result is an exception, ClientApplet invokes WarnWin (Warning Popup Window) and forwards this warning message to it.

(13.1)WarnWin displays the warnings.

(12.2)If the result is not an exception, ClientApplet displays the information in its text area.

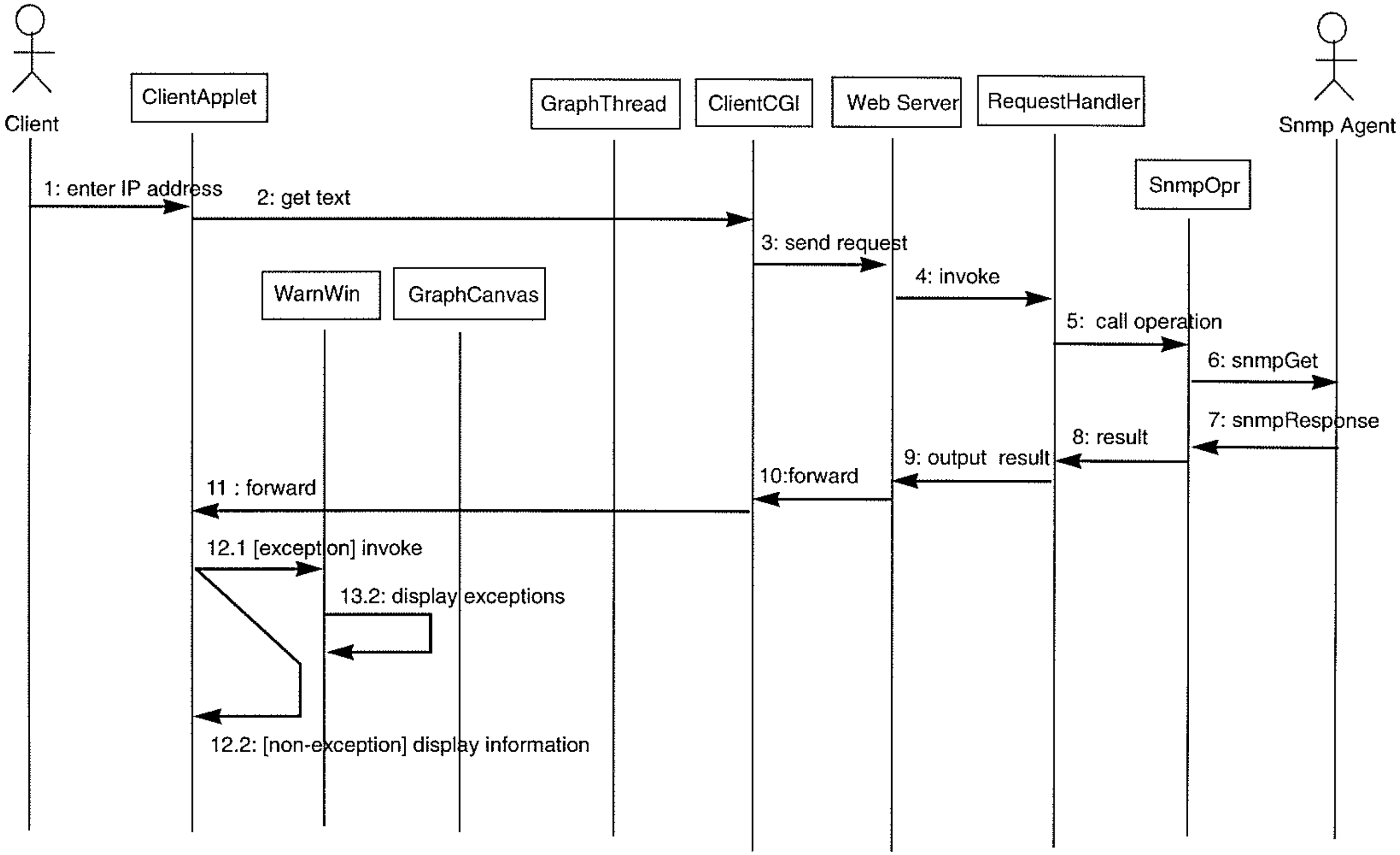


Figure 4.17 The Sequence Diagram for "Get Text" Operation of the NMS2_CGI

Figure 4.18 shows the sequence diagram for "Get Graph" operation of the *NMS2_CGI.* This operation should be done after "Get Text" operation. This is because that the system should provide the text information first and then the graph information. The step-by-step explanation is given below.

(1) Web user clicks the "Get Graph" button.

(2) The applet starts an object of GraphThread.

(3) The object of GraphThread invokes an object of ClientCGI.

(4) - (11) Similar as the steps of (3) - (10) of "Get Text" operation.

(12) ClientCGI forwards the result to the GraphThread.

(13.1)If the result is an exception, GraphThread forwards this exception message to ClientApplet

(14.1)ClientApplet invokes WarnWin and forwards this warning message to it.

(15.1)WarnWin displays the warnings.

(13.2)If the result is not an exception, GraphThread interprets it to the desired data.

(14.2)GraphThread forwards the data in text format to ClientApplet.

(15.2)ClientApplet invokes InfoWin and forwards the data to it.

(16.2)InfoWin displays the information in the text area.

(17.2)GraphThread forwards the data value to GraphCanvas.

(18.2)GraphCanvas draws the graphs according to the data values.

(19.2)GraphThread sleeps for a while (such as 5 seconds).

(20.2)GraphThread then invokes an object of ClientCGI again for "Get Graph" operation.

The later steps are similar to the ones either from (4) - (15.1) or (4) - (20.2).

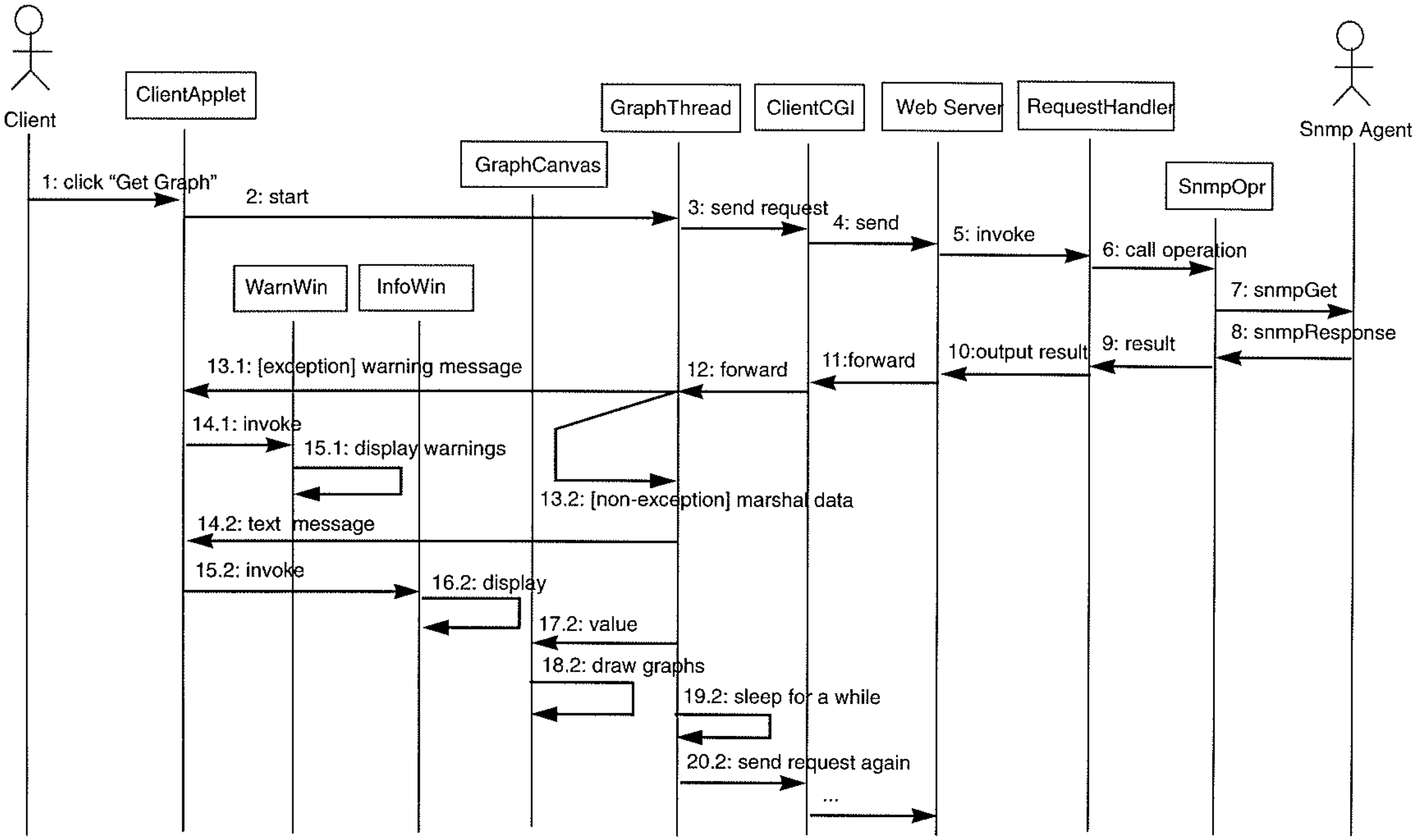


Figure 4.18 The Sequence Diagram for “Get Graph” Operation of the NMS2_CGI

Figure 4.19 shows the sequence diagram for selection operation of the *NMS2_CGI.* This operation can be done either before or after "Get Text" operation, and also it can be done during the "Get Graph" operation. The step-by-step explanation is given below.

(1) The Web user selects an item in the choice list.

(2) - (5) Similar to the steps of (2) - (5) for the "Get Text" operation.

(6) SnmpOpr checks the MIB file (here it is RFC 1213) and gets the descriptions for the queried OIDs. At this time there is no need for SnmpOpr to contact the Snmp Agent.

(7) - (10) Similar to the steps of (8) - (11) for the "Get Text" operation.

(11.1)If the result is not an exception, the ClientApplet invokes DescrWin (Description Popup Window) and forwards the message to it.

(12.1)DescrWin displays the queried OID's description.

(11.2)If the result is an exception, ClientApplet invokes WarnWin (Warning Popup Window) and forwards this warning message to it.

(11.2)WarnWin displays the warnings.

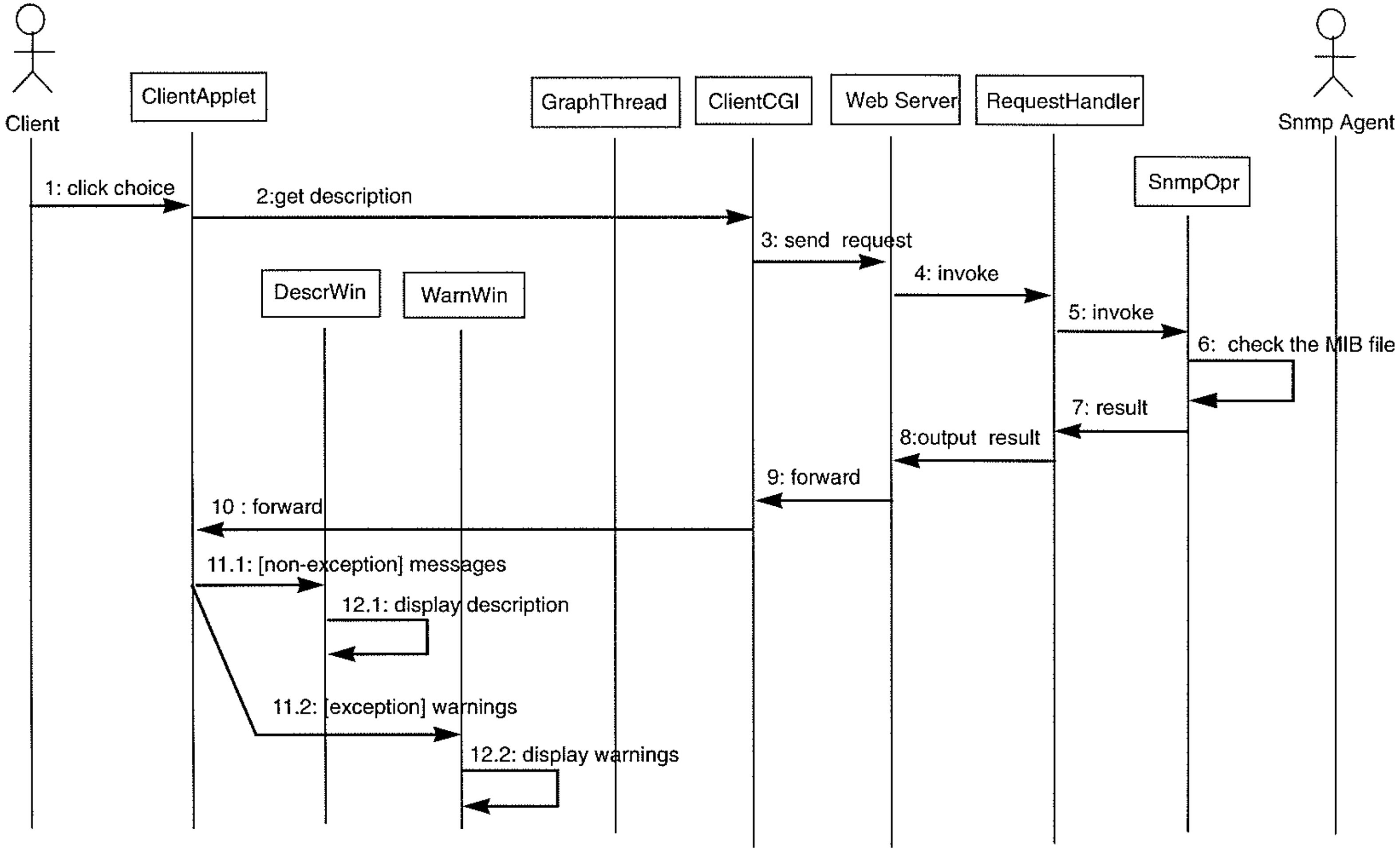


Figure 4.19 The Sequence Diagram for the Selection Operation of the NMS2_CGI

4.4.4 How the system works

Similar to *NMS1_CGI*, after developing and debugging the codes of *NMS2_CGI*, an HTML file and a Jar file are provided for the Web client to download. JAR is used to archive all the class files of the client side to a single Jar file and placed in a folder which can be accessed by the Web client. The HTML file is also placed in a folder which can be accessed by the Web client.

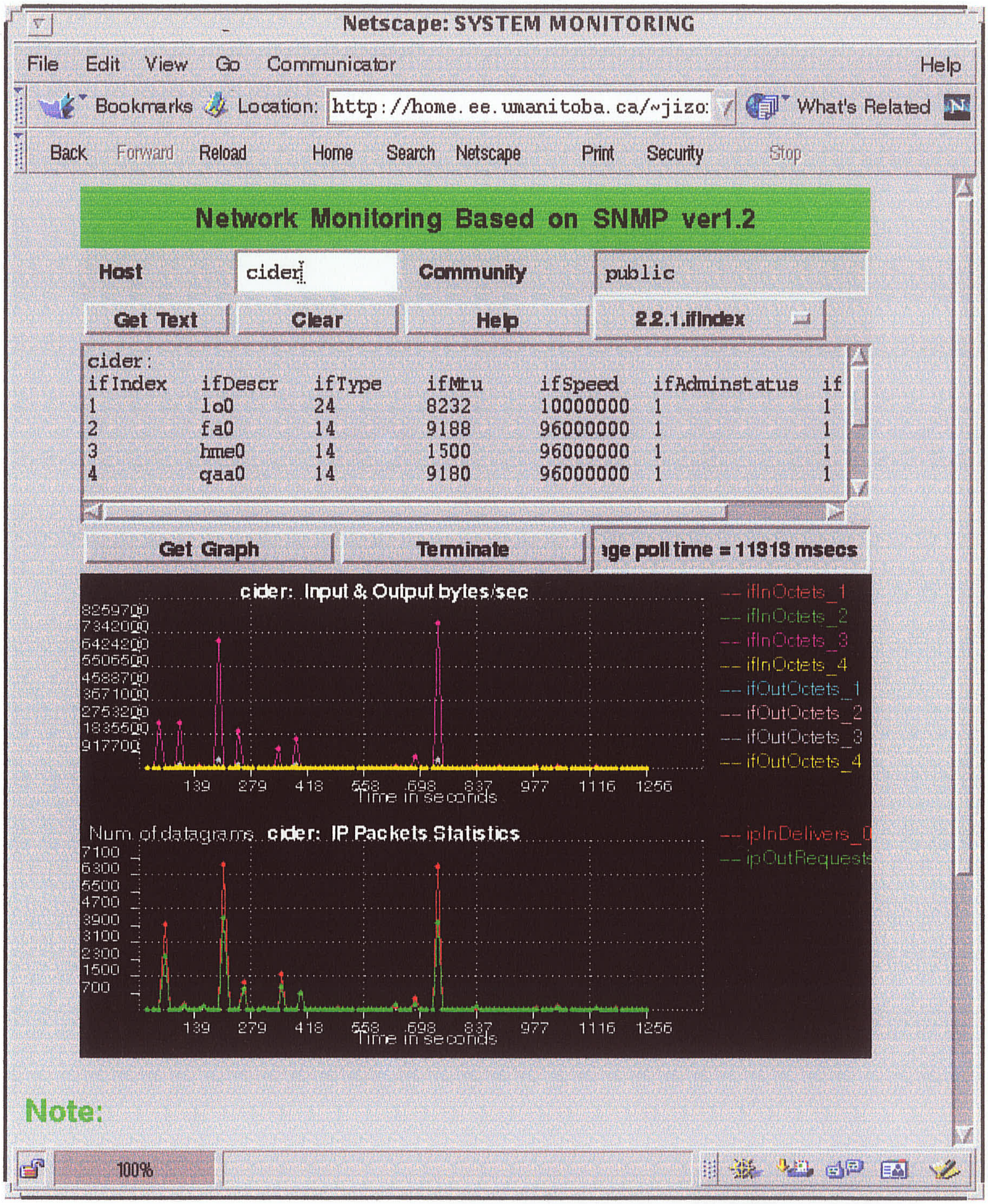
Next the server side class files and a shell script file named *snmpGet2* are placed in a new sub-folder of the cgi-bin folder. The Unix command *ln* is used to make a link between the AdventNet SNMP package and this new folder. In this way there is no need to copy the whole AdventNet SNMP package to this new folder which saves disk space. The contents of *snmpGet2* is only one line of code:

```
/usr/bin/java RequestHandler
```

Now we can run this system via the Web browser. Figure 4.20 illustrates the ClientApplet in the Web browser. Figure 4.21 shows the Information Popup Window during the "Get Graph" operation and Figure 4.22 shows the Description Popup Window for the selection operation (ipIn-Delivers). Figure 4.23 illustrates Help Window.

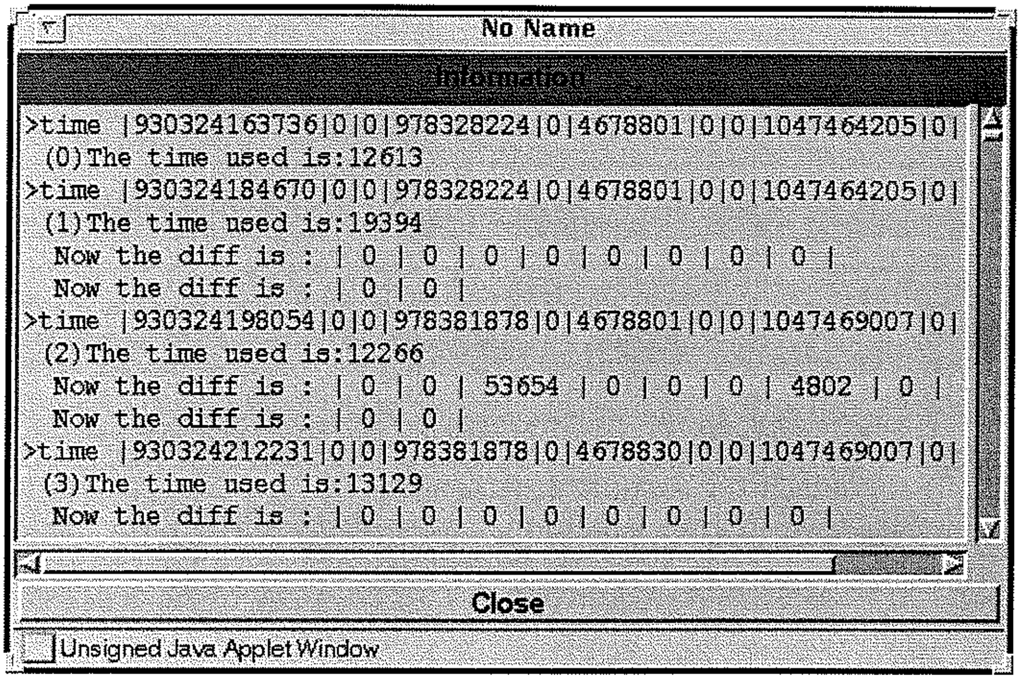
4.5 Summary of Chapter 4

Chapter 4 outlined the design and development process of building network monitoring tools using Web technology, Java, AdventNet's SNMP API and CGI.



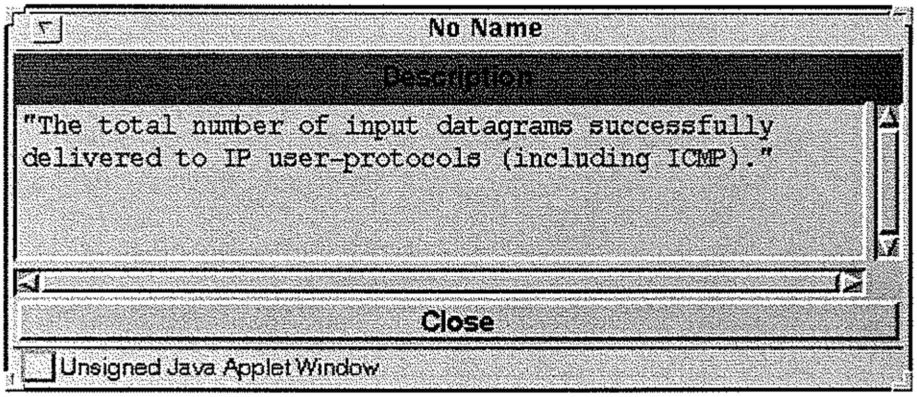
```
cider:
ifIndex      ifDescr     ifType      ifMtu       ifSpeed       ifAdminstatus     if
1            lo0         24          8232        10000000      1                 1
2            fa0         14          9188        96000000      1                 1
3            hme0        14          1500        96000000      1                 1
4            qaa0        14          9180        96000000      1                 1
```

Figure 4.20 The Graphical User Interface of *NMS2_CGI* in Web Browser



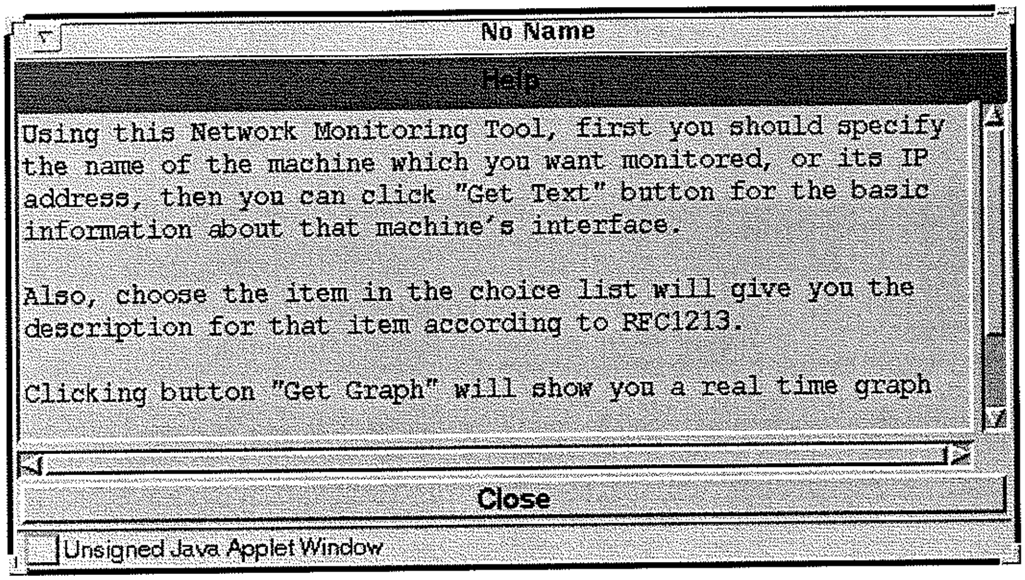
```
>time |930324163736|0|0|978328224|0|4678801|0|0|1047464205|0|
 (0)The time used is:12613
>time |930324184670|0|0|978328224|0|4678801|0|0|1047464205|0|
 (1)The time used is:19394
  Now the diff is : | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  Now the diff is : | 0 | 0 |
>time |930324198054|0|0|978381878|0|4678801|0|0|1047469007|0|
 (2)The time used is:12266
  Now the diff is : | 0 | 0 | 53654 | 0 | 0 | 0 | 4802 | 0 |
  Now the diff is : | 0 | 0 |
>time |930324212231|0|0|978381878|0|4678830|0|0|1047469007|0|
 (3)The time used is:13129
  Now the diff is : | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 4.21 The Information Popup Window of NMS2_CGI



```
"The total number of input datagrams successfully
delivered to IP user-protocols (including ICMP)."
```

Figure 4.22 The Description Popup Window of NMS2_CGI



Using this Network Monitoring Tool, first you should specify the name of the machine which you want monitored, or its IP address, then you can click "Get Text" button for the basic information about that machine's interface.

Also, choose the item in the choice list will give you the description for that item according to RFC1213.

Clicking button "Get Graph" will show you a real time graph

Figure 4.23 The Help Popup Window of NMS2_CGI

CHAPTER 5

Constructing Network Monitoring Tools Using CORBA

In this chapter, I will introduce the construction of a Web-based network monitoring tool using Java, SNMP and CORBA. Also I will introduce VisiBroker from Inprise Corp. (one of the leading CORBA products) and how to use it in this application.

5.1 VisiBroker Overview

As mentioned in Chapter 3, CORBA is one of the most important middleware technologies in our industry. It is a set of specifications defined by the OMG to provide a common architectural framework for object-oriented applications. If we want to use CORBA for our applications, we need a software product which can provide a development and application environment. VisiBroker is one of the leading products which supports the development, deployment, and management of distributed object applications across a variety of hardware platforms and operating systems. It is a complete CORBA2.0 compatible ORB product.

There are two main packages in VisiBroker, one is *VisiBroker for Java* and the other is *VisiBroker for C++*. I chose *VisiBroker for Java (version 3.3)* as my CORBA applications developing tool. The ORB of *VisiBroker for Java* is written totally in Java which makes it a downloadable ORBlet. Netscape includes *VisiBroker for Java* (currently version 2.5) in all its browsers and servers which makes the CORBA IIOP available over the internet. The objects built with *VisiBroker for Java* are easily accessed by Web-based applications which communicate using IIOP.

There are three other components packaged with *VisiBroker for Java*, they are: Naming Service, Event Service and GateKeeper [VBJ33].

Naming Service allows you to associate one or more logical names with an object implementation and store those names in a namespace. Event Service provides a facility that decouples the communication between objects so that multiple objects can send data asynchronously to multiple consumer objects through an event channel. These were not used in this application.

Gatekeeper runs on a web server and enables client programs to locate and use objects that do not reside on the web server and to receive callbacks, even when firewalls are being used. It is a very important component for the Web-based applications as illustrated in Figure 3.6 for a 3-tiered client/server model.

VisiBroker for Java provides several runtime support services, one of them is Smart Agent (osagent) which is used by applications to locate the objects they wish to use. It is a process that must be started on at least one host in the local network for the objects using the VisiBroker ORB. For others, such as Location Service, Object Activation Daemon and Interface Repository, please refer to [VBJ33].

VisiBroker for Java also provides development tools such as: idl2ir, idl2java, java2iiop and java2idl. Among them idl2java is a very basic tool used for developing the Java programs which using VisiBroker ORB. It is a precompiler used to generate Java stubs (for the client side) and skeletons (for the server objects) from an IDL file.

VisiBroker for Java provides very powerful features for developing applications in the CORBA domain, especially for Web-based applications. According to [OCI], VisiBroker outperforms Orbix from Iona (a leading provider of CORBA technology) in many features. So VisiBroker was chosen for my applications.

5.2 Developing applications with VisiBroker

Figure 5.1 shows a picture illustrating the steps for developing a CORBA application using the VisiBroker ORB. Referring to Chapter 3, we need to use IDL first to define the objects services which will be available to the clients. This includes the definitions of types of the objects, their attributes, the methods they export, and the method parameters. After that, running this IDL file through the VisiBroker *idl2java* precompiler will generate a bundle of class files which include stub code for the client program and skeleton code for the object implementation. These class files are contained in the package which we named module in the IDL file.

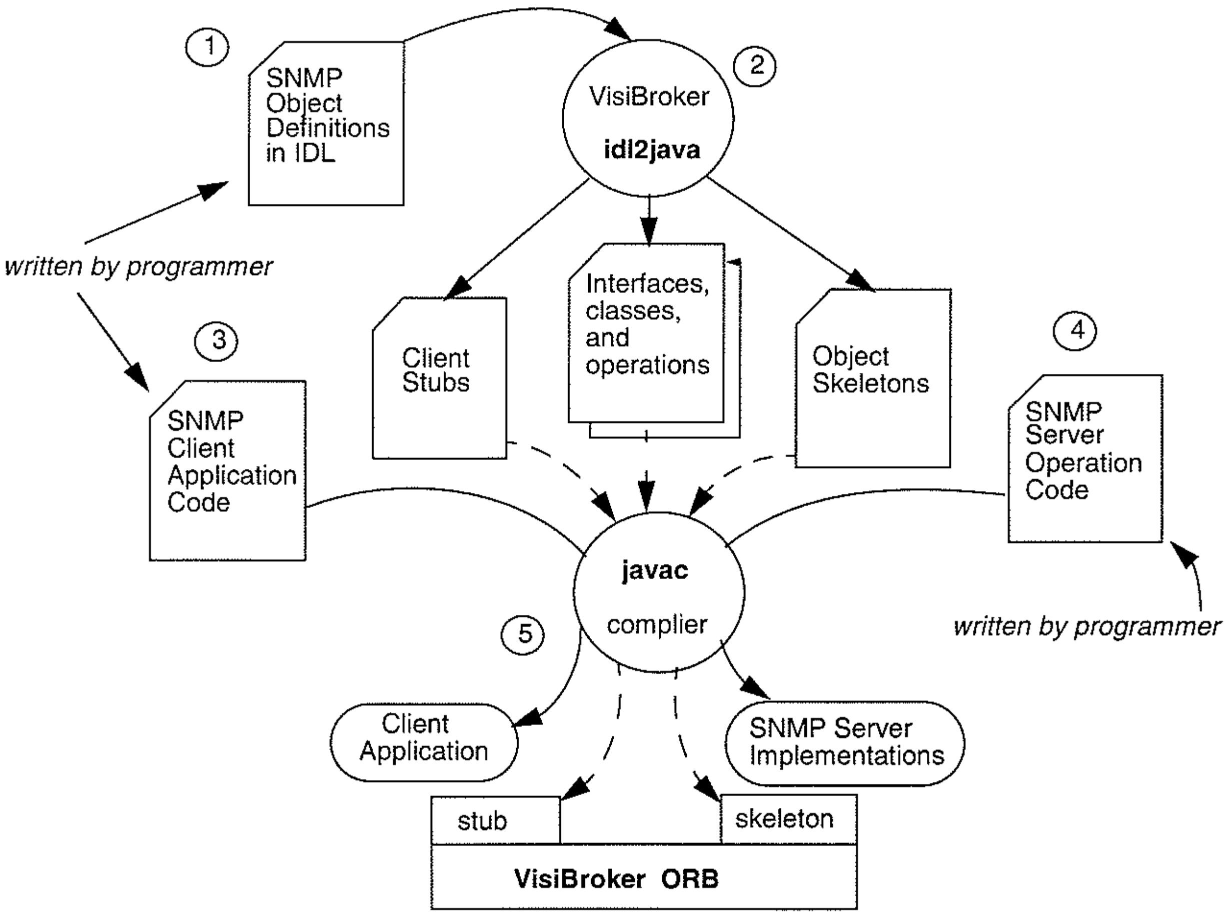


Figure 5.1 Developing SNMP Network Monitoring Tool Using VisiBroker

Next the SNMP client application code and server code are written. For the server code we can develop it according to a file named _example_*.java (* represents the interface name we defined in IDL file) in the package generated by *idl2java.*

After the above steps are completed, we should use the Java compiler to compile them to produce Java class files and run them for the network monitoring application.

5.3 Constructing Network Monitoring System I Using CORBA

The design of the Network Monitoring System I using CORBA (denoted *NMS1_CORBA* for short), which queries the basic information of System group in MIB-II, is described here. Referring to 4.3 of Chapter 4, everything is similar except that we use different middleware: CORBA.

5.3.1 Class Diagram Design of NMS1_CORBA

The class diagram of *NMS1_CORBA* is illustrated in Figure 5.2. Comparing to Figure 4.5, the objects *NMS1_Stub, NMS1_Skeleton, OprInterface* are added and *ClientCGI* is removed because we don't use CGI here. A brief description about them is provided below:

ClientOrbApplet1 is a Java applet which provides the main window in the Web browser for the Web client to interact with. It is similar to ClientApplet in Figure 4.5 except that it uses CORBA now (has the attributes: *orb,* an object of *org.omg.CORBA.ORB* and *op_interface,* an object of *OprInterface*).

SnmpServer is a server implementation which is responsible for handling the query from the Web client, instantiating an object of SnmpOpr to contact the SNMP agent to complete the SNMP operation.

OprInterface is an interface that provides the *snmpGet* operation which is visible to the client side. This operation returns the data of strut type. The interface only provides the operation signature, not the implementation.

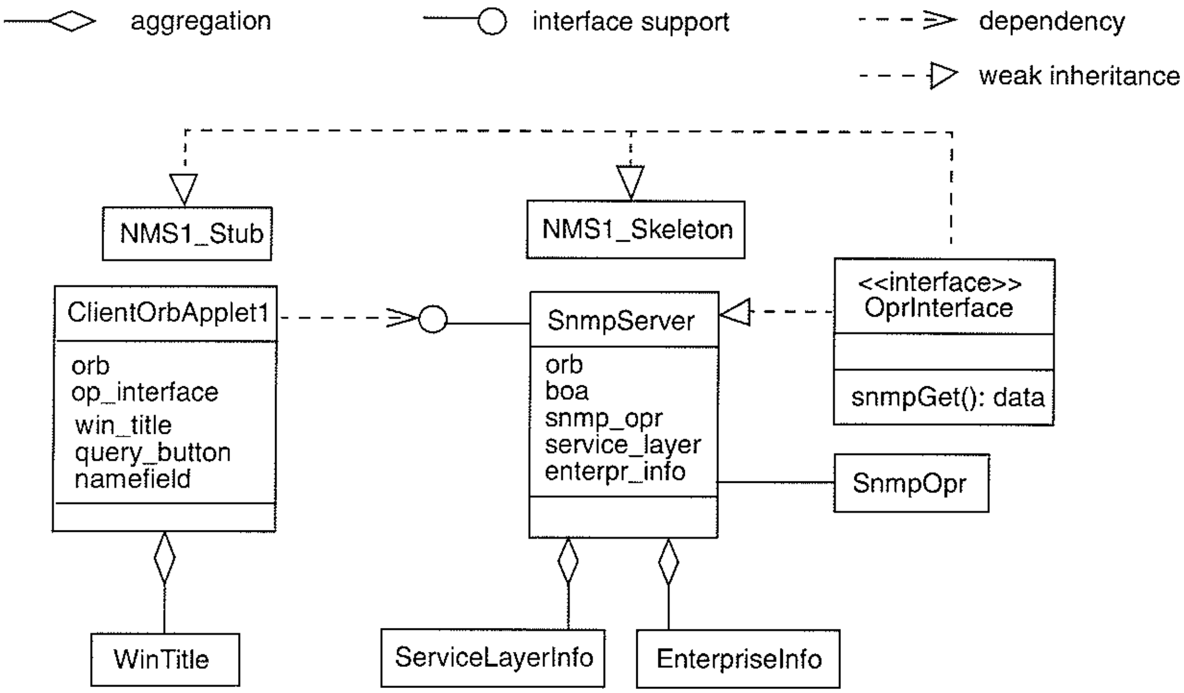


Figure 5.2 The Class Diagram of NMS1_CORBA

NMS1_Stub contains the stub code for the *OprInterface* object on the client side.

NMS1_Skeleton contains the skeleton code for the *OprInterface* object on the server side. The stub code and the skeleton code are automatically generated when compiling the IDL file.

Others, like *WinTitle*, *ServiceLayerInfo*, *EnterpriseInfo* and *SnmpOpr* are same as the objects used in *NMS1_CGI.*

5.3.2 The Code Development — CORBA part

To realize *NMS1_CORBA* in the CORBA domain, we need first to develop the IDL files according to the system requirements. From the class diagram design, we see that we need to define an interface *OprInterface* which embeds a method *snmpGet* that returns the struct data.

Figure 5.3 gives the code for the NMS1.idl.

```
module SnmpSys {
    struct sysData {
       string s_Descr;
       string s_Oid;
       string s_Time;
       string s_Con;
       string s_Name;
       string s_Locat;
       string s_Serv;
       string error;
    };

    exception OprException {
       string reason;
    };

    interface OprInterface {
       sysData snmpGet(in string host) raises (OprException);
    };
};
```

Figure 5.3 The Code of NMS1.idl File

Referring to 3.2.3 of Chapter 3, the keyword module provides a namespace to group a set of interfaces and data types for a common purpose. It is similar to the Java package and introduces an additional level of hierarchy in the IDL namespace.

In NMS1.idl the module name, struct data type, exception name and one interface named *OprInterface* which provides *snmpGet()* operation are designed. According to Figure 5.1, we pass it to *idl2java* of VisiBroker and this generates a number of Java files. These files are stored in a generated sub-directory named *SnmpSys*, which is the module name I specified in NMS1.idl. Some of these files are listed below:

- OprInterface.java — The OprInterface interface declaration.
- OprInterfaceHelper.java — Declares the OprInterfaceHelper class. This class defines helpful utility functions such as bind, read, write, insert, etc.

- OprInterfaceHolder.java — Declares the OprInterfaceHolder class, which provides a holder for passing parameters.
- OprException.java — Declares the OprException class, an object of this class is used to carry the specified information via the ORB.
- SysData.java — The class file used to construct an object of SysData, which is used to transfer the specified data via the ORB.
- _st_OprInterface.java — The stub code for the OprInterface object on the client side.
- _OprInterfaceImplBase — The skeleton code for the OprInterface object on the server side.

Now according to Figure 5.1, we should write the client side code for *NMS1_CORBA*. Referring to Figure 5.2, *WinTitle* is the same as one of the *NMS1_CGI*, hence we only need to rewrite the applet code. Figure 5.4 illustrates the part of code *ClientOrbApplet1.java*. It is similar to the applet code of *NMS1_CGI*, but here we need to initialize the ORB and use *OprInterfaceHelper.bind* to create an object of *OprInterface*, and then invoke the method snmpGet via this object.

The ORB class provides functionalities used by both client and server. To initialize the Visibroker ORB, the method *init()* will be called. In my application *init()* is called with the argument: *this applet*. In this way the ORB in client side will establish a connection to an instance of VisiBroker's Gatekeeper first, which is expected to be running on the machine where the applet has been downloaded from. Gatekeeper helps the client locate and use objects that do not reside on the web server and to receive callbacks which are impossible due to applet sandboxing*.

* Web browsers impose two types of security restrictions on Java applets (also called Java sandbox security):

- They allow applets to only connect back to the host from which the applet was downloaded.
- They allow applets to only accept incoming connections from the host from which the applet was downloaded.

To obtain a reference to the remote object: *OprInterface*, we use the *bind* method in *OprInterfaceHelper*. When the applet calls the *bind* method, the ORB will contact the Smart Agent to locate an appropriate server which is offering the requested interface: *OprInterface*. Then the ORB tries to establish a connection between the applet and this server. If the ORB cannot locate or connect to the server which implements this interface, the *bind* method will raise a CORBA system exception.

```
// ClientOrbApplet1.java

import java.awt.*;
import java.awt.event.*;
...

public class ClientOrbApplet1 extends Applet implements ActionListener {
    private SnmpSys.OprInterface  op_interface;
    ...

    public void init() {
       ...

       // Initialize the ORB.
       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this, null);

       // Locate an Snmp Operation.
       op_interface = SnmpSys.OprInterfaceHelper.bind(orb, "System Operation");
    }

    ...  // later use op_interface.snmpGet to get the queried information
}
```

Figure 5.4 The Part of Code ClientOrbApplet1.java

Next we need to develop the server side code. Referring to Figure 5.2, we have already had *OprInterface.java* which is automatically generated by *idl2java* and stored in the *SnmpSys* package. *ServiceLayerInfo, EnterpriseInfo* and *SnmpOpr* are same as the ones in *NMS1_CGI.* So now we only need to write the code for the server implementation. To realize this, we partition the functions of it into the two class files. One is named *SnmpServer.java* which is responsible for handling the requests from the client and the other is *OprInterfaceImpl.java* which is responsible for implementation of *OprInterface*.

Figure 5.5 gives the code of *SnmpServer.java*. First the ORB and BOA must be initialized before any CORBA objects are created. BOA stands for Basic Object Adapter and is used by object implementations to activate and deactivate the objects they offer to clients. If you use the *BOA_init()* method with no arguments, you accept the default thread policy which is thread pooling. For more information, please refer to [VBJ33].

```
// SnmpServer.java

public class SnmpServer {

    public static void main(String[] args) {
       try {
           // Initialize the ORB.
           org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
           // Initialize the BOA.
           org.omg.CORBA.BOA boa = orb.BOA_init();

           // Create the Snmp Operation object.
           SnmpSys.OprInterface opi =
           new OprInterfaceImpl("System Operation");

           // Export the newly created object.
           boa.obj_is_ready(opi);
           System.out.println(opi + " is ready.");
           // Wait for incoming requests
           boa.impl_is_ready();

       } catch (Exception e) {
           // something failed...
           System.out.println(e);
       }
    }
}
```

Figure 5.5 The Code of SnmpServer.java

Then the *OprInterface* object is created by instantiating the class OprInterfaceImpl. Next it is registered with the BOA through the *obj_is_ready()* method and this makes the object implementation visible to clients on the network.

Finally the *impl_is_ready()* method is called on the BOA which puts the server in an infinite loop waiting for incoming calls and refers them to the appropriate objects.

Figure 5.6 shows the part of code *OprInterfaceImpl.java,* which extends the skeleton (*_OprInterfaceImplBase*) and is responsible for implementation of *OprInterface*. So we should give the definition of the operation *snmpGet()* within this code.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class OprInterfaceImpl extends SnmpSys._OprInterfaceImplBase {

    /** Construct a persistently named object. */
    public OprInterfaceImpl(String name) {
       super(name);
    }

    /** Construct a transient object. */
    public OprInterfaceImpl() {
       super();
    }

    public SnmpSys.sysData snmpGet(String host)
       throws SnmpSys.OptException {

       SnmpSys.sysData re_data;
       String          warning;

       try {
           SnmpOpr snmpopr = new SnmpOpr(host, "public");
           ... // do the implementation

           return re_data;

       } catch (Exception e) {
           System.out.println(" System Exception in snmpGet! \n" +  e);
           return null;
       }
    }
}
```

Figure 5.6 The Part of Code OprInterfaceImpl.java

5.3.3 Putting the NMS1_CORBA system to work

Since this application uses *VisiBroker for Java 3.3*, we need to install it on one of the machines in our local network. For my case I installed it on the one of the Unix workstations. During the installation, we should set or accept the default values for the *OSAGENT_PORT* and *VBROKER_ADM* environment variables. These variables will be used by Smart Agent and other services.

After we finish developing the codes, we send them to *javac* complier to get the bytecodes. Since this is a Web-based application, we need to have a HTML file. Figure 5.7 illustrates the file of NMS1_CORBA.html which embeds the Client.jar and vbjorb.jar files for downloading. The file vbjorb.jar provides the VisiBroker ORB which is used on the client side. As I mentioned, Netscape has already embedded the VisiBroker ORB (currently *VisiBroker for Java 2.5*) in itself so we don't need to download the vbjorb.jar if we use *VisiBroker for Java 2.5*. Since I use *VisiBroker for Java 3.3*, the vbjorb.jar needs to be downloaded. This also makes other Web browsers such as Explorer capable of being used for the client side.

There are two parameters in NMS1_CORBA.html file. The first one specifies use of *VisiBroker for Java 3.3* instead of Communicator embedded *VisiBroker for Java 2.5*. The second one specifies the URL which is associated with the IOR file generated by the Gatekeeper. The ORB on the client side uses this value to locate Gatekeeper's IOR file. In this case the Gatekeeper is running on the Web server (vermouth) at port 15000. We can do this by entering the following command in vermouth:

```
        prompt> gatekeeper &
```

which runs Gatekeeper at the default port 15000. Also we can use Gatekeeper Configuration Manager to set the exterior_port of Gatekeeper.

```
<html>
  <head>
    <title>Local Network Monitoring</title>
  </head>

  <body bgcolor="#C5C5C5">
    <center>
       <applet
           code="ClientOrbApplet1.class" ARCHIVE="Client.jar, vbjorb.jar"
           width=510 height=360>
      <param name=org.omg.CORBA.ORBClass value=com.visigenic.vbroker.orb.ORB>
      <param name=ORBgatekeeperIOR
           value=http://vermouth.ee.umanitoba.ca:15000/gatekeeper.ior>
      <B>Sorry, your web browser should support Java1.1.</B>
       </applet>
    </center>
  </body>
</html>
```

Figure 5.7 The File of NMS1_CORBA.html

Next the Smart Agent is started on one machine of the local network by typing:

```
prompt> osagent &
```

The SNMP server is started by using VisiBroker's *vbj* command. The *vbj* command invokes the Java virtual machine and offers other special features for running a CORBA application. For more information, please refer to [VBJ33]. Now we can start our server by entering:

```
prompt> vbj SnmpServer &
```

And then you will get the following information from the Terminal window:

```
prompt> SnmpOptImpl[Server,oid=PersistentId[repId=IDL:SnmpJava
/SnmpOpt:1.0,objectName=Snmp Operation]] is ready.
```

This server can be started either on the Web server (vermouth in our case) or any other machine in our local network. This is one of the big advantages of using CORBA. Further discussion will be given in next chapter.

Now the system is ready to provide the network monitoring service to the Web clients. The graphic user interface of *NMS1_CORBA* is the same as for *NMS1_CGI*, and is illustrated in Figure 4.15.

5.4 Constructing Network Monitoring System II Using CORBA

The Network Monitoring System II using CORBA (denoted *NMS2_CORBA* for short) is very similar to *NMS2_CGI* except that we use different middleware: CORBA.

5.4.1 Class Diagram Design of NMS2_CORBA

Referring to section 4.4 of Chapter 4, the class diagram of *NMS2_CORBA* is designed and it is illustrated in Figure 5.8. As CGI is not used, *ClientCGI* is removed. But the objects *NMS2_Stub, NMS2_Skeleton* and interface *OprInterface2* are added now for using CORBA. A brief description about them follows:.

OprInterface2 is the interface which defines the methods (service) that the SNMP server provides for clients. Here are some important ones:

- *describe()* returns a string which describes the item in the choice list
- *getEntry()* returns data of struct type which gives the information of the interface group and the ip group for the queried machine according to MIB-2
- *getGraph()* returns data of struct type which gives the value of the total number of bytes in and out of all the interfaces and IP packets in and out of the queried machine
- *getWarns()* returns a string which indicates exceptions happened while processing

ClientOrbApplet2 is similar to *ClientApplet* in Figure 4.16 except that it uses CORBA. So it needs to instantiate an object of *org.omg.CORBA.ORB* and then get an object of *OprInterface2* with the help of the ORB. After that it can invoke the methods: *describe(), getEntry()* and *get-*

Warns() via this interface.

GraphThread is also similar to the one in *NMS2_CGI* except that it does not need to contact *ClientCGI*. Instead it needs to have an object of *OprInterface2*. In this way it can invoke the methods of *getGraph()* and *getWarns()* via this interface.

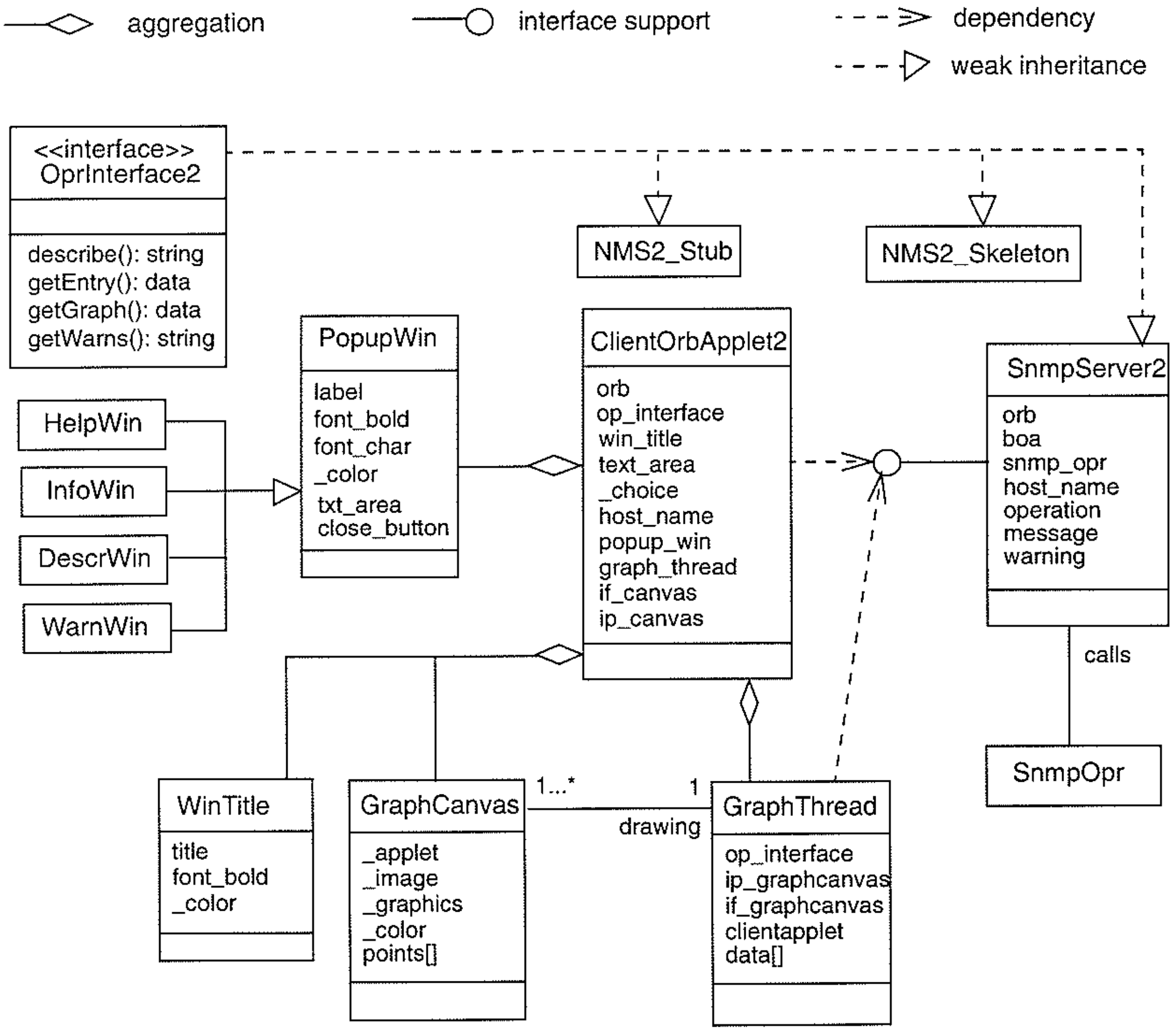


Figure 5.8 The Class Diagram of NMS2_CORBA

SnmpServer2 is similar to RequestHandler in NMS2_CGI which is responsible for handling the queries from the Web client. Here it uses CORBA not CGI, so it should have some CORBA items, such as the object of *org.omg.CORBA.ORB* and the object of *org.omg.CORBA.BOA*.

NMS2 Stub contains the stub code for the *OprInterface2* object on the client side. *NMS2 Skeleton* contains the skeleton code for the *OprInterface2* object on the server side. The stub code and the skeleton code are automatically generated when compiling the IDL file.

Others such as *WinTitle*, *PopupWin*, *GraphCanvas* and *SnmpOpr* are same as the objects used in *NMS2_CGI*, as are *HelpWin*, *InfoWin*, *DescrWin* and *WarnWin*.

5.4.2 The Code Development — CORBA part

Consider the IDL file of *NMS2_CORBA*. From the class diagram design above, we see that we need to define an interface *OprInterface2* which embeds several methods that return the desired data. Figure 5.9 gives the code for the NMS2.idl.

In NMS2.idl the module *SnmpJava* is defined which will be interpreted to a Java package by using *idl2java*. Also there are struct data type and sequence data type. The sequence of struct type of data is similar to the vector data type in Java. Here it is used for the returned data type.

There is one interface named *OprInterface2* in NMS2.idl which provides several operations. Two operations are defined for the SNMP operation initialization, one with arguments for querying the machines in the network and one without arguments for choosing the item description operation. *snmpClose()* is for terminating the "Get Graph" operation. Other operations have already been described in 5.4.1.

```
module SnmpJava {

    struct entStruct {
       string oidInd;
       string oidDes;
       string oidTyp;
       string oidMtu;
       string oidSpd;
       string oidAdm;
       string oidOpr;
    };
    typedef sequence < entStruct > entrySeq;

    typedef sequence < long > oprSeq;

    struct valStruct {
       long long time;
       long long value;
    };
    typedef sequence < valStruct > valSeq;

    interface OprInterface2 {
       void snmpInit_1();
       void snmpInit_2(in string host, in string community);
       void snmpClose();

       string   describe(in string item_oid);
       entrySeq getEntry();
       valSeq   getGraph(in oprSeq upArr);
       string   getWarns();
    };
};
```

Figure 5.9 The Code of NMS2.idl File

According to the process illustrated in Figure 5.1, we pass NMS2.idl to *idl2java* of VisiBroker which generates a number of Java files which are located in a sub-directory named *SnmpJava*. They include:

- OprInterface2.java — The OprInterface2 interface declaration.
- OprInterface2Helper.java — Declares the OprInterface2Helper class which provides the bind method.
- OprInterface2Holder.java — Declares the OprInterface2Holder class for passing parameters.

- _OprInterface2ImplBase.java — The skeleton code for the OprInterface2 object on the server side.
- _st_OprInterface2.java — The stub code for the OprInterface2 object on the client side.

There are also others class files such as *entStruct.java* and *valStruct.java* which are used for constructing the data types in order to transfer the data via the ORB. Also some Helper and Holder class files which are used to assist to achieve this purpose, such as *entStructHelper.java*, *entStructHolder.java*, etc.

Now let's see how to develop the codes for both client side and server side. On client side we have *ClientOrbApplet2, WinTitle, PopupWin, GraphThread* and *GraphCanvas* classes. Among them, *WinTitle.java, PopupWin.java* and *GraphCanvas.java* are same as the ones of *NMS1_CGI.* For *ClientOrbApplet2.java*, similar to *ClientOrbApplet1*, lines of code are added to initialize the Visibroker ORB, to get an object of *OprInterface2* (by invoking the bind method of *OprInterface2Helper).* The methods defined in *OprInterface2* can be invoked for the service.

GraphThread.java here is very similar to the peer one of *NMS2_CGI*, the difference is that it uses CORBA now. As I mentioned in 5.4.1, *GraphThread* should have an object of *OprInterface2* for invoking the methods of *getGraph()* and *getWarns().* This is done by passing that object from *ClientOrbApplet2* to it.

On the server side, the class SnmpServer2 is split to two classes. One is SnmpServer.java which is similar to the one of *NMS1_CORBA* for initializing the ORB, BOA and so on. The other is *OprInterface2Impl.java* which is responsible for implementation of *OprInterface* interface. SnmpOpr is the same as the one in *NMS1_CORBA.*

5.4.3 How the system works

Similar to what we have done to *NMS1_CORBA*, first we finish developing and debugging the codes of *NMS2_CORBA*, and then we need an HTML file and Jar files for the Web client to download. Next the Smart Agent, Gatekeeper and SnmpServer2 are started. Remember that Gatekeeper should be running on Web server and SnmpServer2 can be run on either Web server or other machine in the local network.

And now the system is ready to run. The figures showing this system are same as the ones in *NMS2_CGI* illustrated from Figure 4.20 to Figure 4.23. Please note the values in average polling time field of Figure 4.20 which are much different for *NMS2_CGI* and *NMS2_CORBA* implementations. The discussion of this will be included in the next chapter.

CHAPTER 6

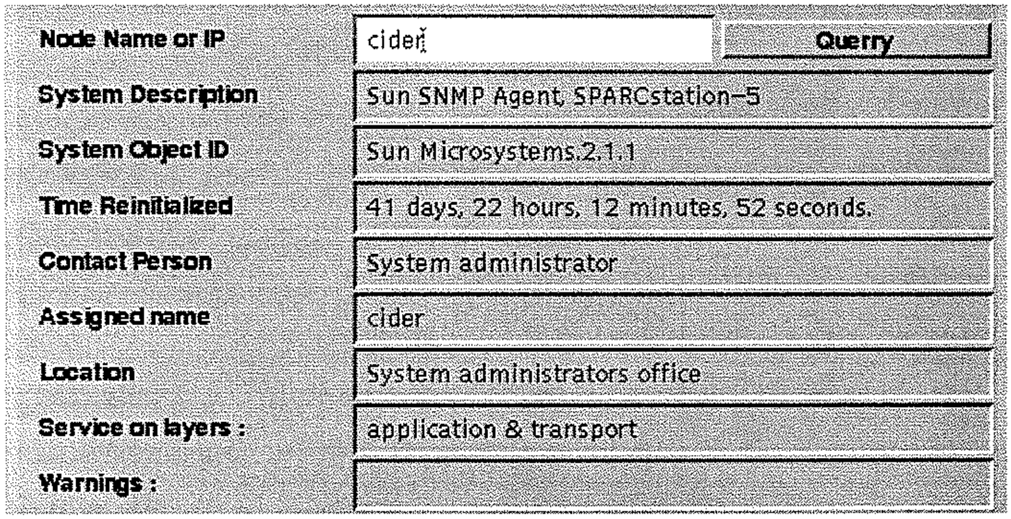
Experimental Results and Discussion

In this chapter, I will give the experimental results for network monitoring via CGI and CORBA and then discuss their performance.

6.1 Experimental Results and Discussion

(1) All the systems (*NMS1_CGI*, *NMS2_CGI*, *NMS1_CORBA* and *NMS2_CORBA*) were tested using Web browsers (Netscape 4.5 or later version, Explore 4.0 or later version) on Unix workstations, PCs and Macintoshes. In all cases the browser behaved as expected.

(2) The network monitoring system 1 (use *NMS1* for short) displays the following basic information as shown in Figure 6.1:



Node Name or IP	cider
System Description	Sun SNMP Agent, SPARCstation-5
System Object ID	Sun Microsystems.2.1.1
Time Reinitialized	41 days, 22 hours, 12 minutes, 52 seconds.
Contact Person	System administrator
Assigned name	cider
Location	System administrators office
Service on layers :	application & transport
Warnings :	

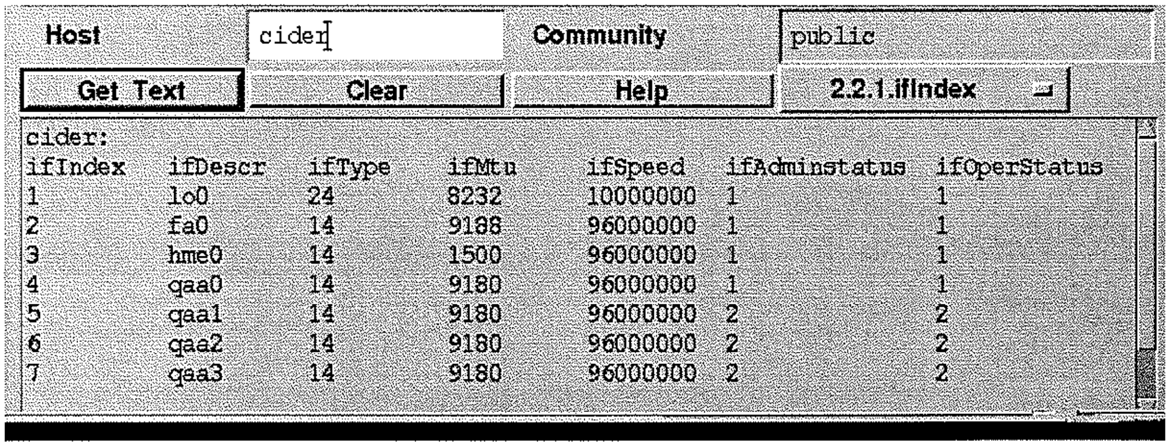
Figure 6.1 Basic Information about cider Machine from NMS1

- The System Description shows that cider is a SPARC 5 workstation, running the Sun operating system and the Sun SNMP agent (*snmpd*) (this is because *snmpd* is embedded in

Solaris 2.6 and later versions)

- System Object ID shows that network management subsystem is the Sun Microsystems.2.1.1
- Time Re-initialized shows how much time (41 days, 22 hours, 12 minutes, 52 seconds) has passed since cider was re-initialized
- Contact Person shows that the system administrator should be contacted for cider's maintenance
- Assigned name gives the administratively-assigned name: cider for this machine
- Location shows that cider is located in system administrators office
- Service on layers shows that cider provides transport and application services according to OSI seven layer architecture

(3) The network monitoring system 2 (use *NMS2* for short) displays both text information and real time graphs of the queried system (cider) (see Figure 6.2, 6.3 & 6.4):



Host: cider

Community: public

```
cider:
ifIndex   ifDescr   ifType   ifMtu   ifSpeed   ifAdminstatus   ifOperStatus
1         lo0       24       8232    10000000  1               1
2         fa0       14       9188    96000000  1               1
3         hme0      14       1500    96000000  1               1
4         qaa0      14       9180    96000000  1               1
5         qaa1      14       9180    96000000  2               2
6         qaa2      14       9180    96000000  2               2
7         qaa3      14       9180    96000000  2               2
```

Figure 6.2 Text Information about the Interface Group of cider

- *ifindex* shows that cider has seven interfaces, numbered from 1 to 7
- *ifDescr* gives information about the interfaces of cider. Here "lo0" represents a software

loopback interface. “fa0” represents a Fore IP emulation interface and “qaa0” a classical IP emulation interface, where “fa0” and “qaa0” both utilize the Fore ATM card. “hme0” stands for the fast ethernet interface. The “qaa1” to “qaa3” interfaces are not used and hence they are defined “down” in *ifAdminStatus* and *ifOperStatus*. All of these parameters are set automatically by the installation of Sun’s operation system (Solaris) or the Fore interface card’s software.

- *ifType* shows the interface types of cider according to its physical/link layer(s). Value 24 stands for “softwareLoopback” which is used to transfer data between processes in the same system. Value 14 stands for “hyperchannel” but it is not known as to why the system set this value to the interfaces here. As there is no corresponding types for those interfaces in interface group of MIB-II, so they should be set to value 1 that denotes none of the values from 2 to 54 can be used to represent these interfaces.
- *ifMtu* gives the size of the largest protocol data unit (in bytes) which can be received or sent on each interface, so here the largest size of PDU can be received and sent on loopback interface is 8232 bytes, and Ethernet interface 9188 bytes, and so on.
- *ifspeed* shows the values of current bandwidth for each interface of cider, such as 10 Mbps for loopback interface, 96 Mbps for Ethernet interface and so on.
- *ifAdminStatus* shows which interface of cider is up (1), down (2) or testing (3). So we can see the first four interfaces of cider are up and the rest three are down.
- *ifOperStatus* also shows that first four interfaces of cider are up and the rest of three down.
- Figure 6.3 shows total number of bytes received and transmitted on all up interfaces dynamically in real time. From Figure 6.2 we see that cider has total seven interfaces, the first four interfaces are up, so *NMS2* verified this and showed the input and output bytes/

sec on those four interfaces.

- Figure 6.4 shows IP packets statistics for cider in a real time graph. The red line represents "ipInDelivers" which is defined as the total number of input datagrams successfully delivered to IP layer. And the green line represents "ipOutRequests" which indicates the total number of IP datagrams supplied to the local IP layer for transmission. From Figure 6.4 we see these two parameters were the same or close within the tested time. This means that there was no traffic congestion in cider machine.

(4) The systems (*NMS1_CORBA* and *NMS2_CORBA*) were tested in the following ways respectively and both of them worked fine.

- Snmp server was started on the Web server (case1)
- Snmp server was started on another machine (Unix workstation) in the local network(case2)

(5) The average poll time data (shown in the applet of NMS2, see Figure 4.20) was collected and is used for performance evaluation of CORBA and CGI implementation. For the CORBA application, this data was collected respectively for two cases as mentioned above. For each case, "Get Graph" was started and an average poll time was calculated after 100 polls. After that, "Get Graph" was started with the CGI approach and another average poll time was calculated. NMS2 was tested 20 times for case1, 20 times for case2 and 40 times for the CGI approach. The data is depicted in Figure 6.5 and 6.6. From them we can see:

- CGI application is much slower than the CORBA peer (both case1 and case2).
- In CORBA approach, case1 is somewhat faster than case2. This is because that Gatekeeper and Snmp Server are running on the same machine (Web server). The trade-off is that it increases the load of the Web server.

- From both Figure 6.5 and 6.6 we can see that the mean of poll time (20 tests for each figure) for CGI approach is not same but they are close (one is 13.4 sec. and the other 14.0 sec.). From the statistical view, these two values will approach to the same value in the long run.
- Figure 6.7 gives much more information about the exact time taken for each poll for the "Get Graph" operation in case1. From the figure we can see that it usually takes around 90 msec for each poll but sometimes it takes much longer than that. It is difficult to say what causes this. The working environment of the client, Web server and managed entities (machines) can cause this to happen.

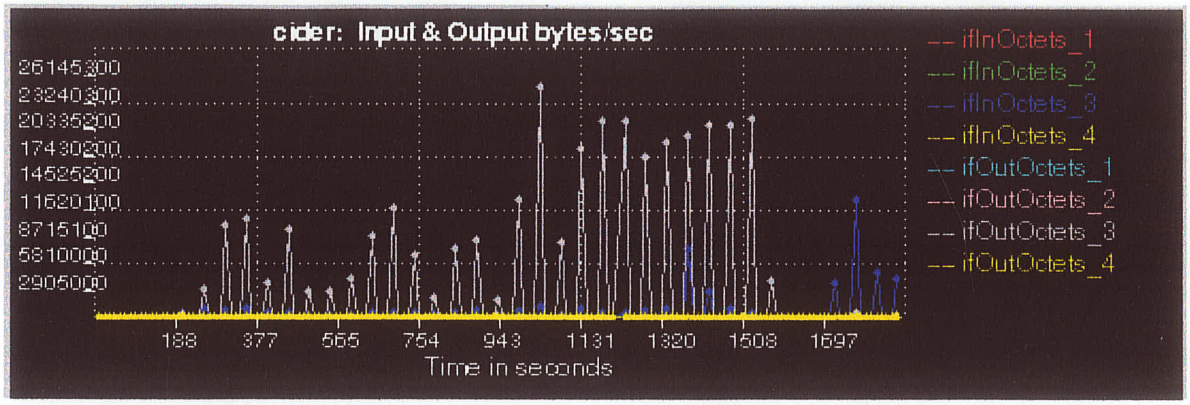


Figure 6.3 The Real Time Graph of Input & Output Bytes/sec of cider

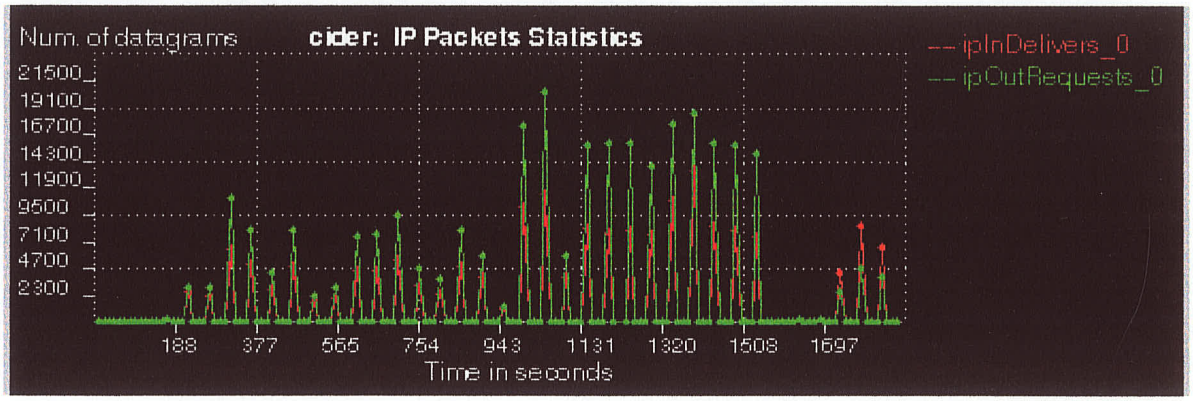
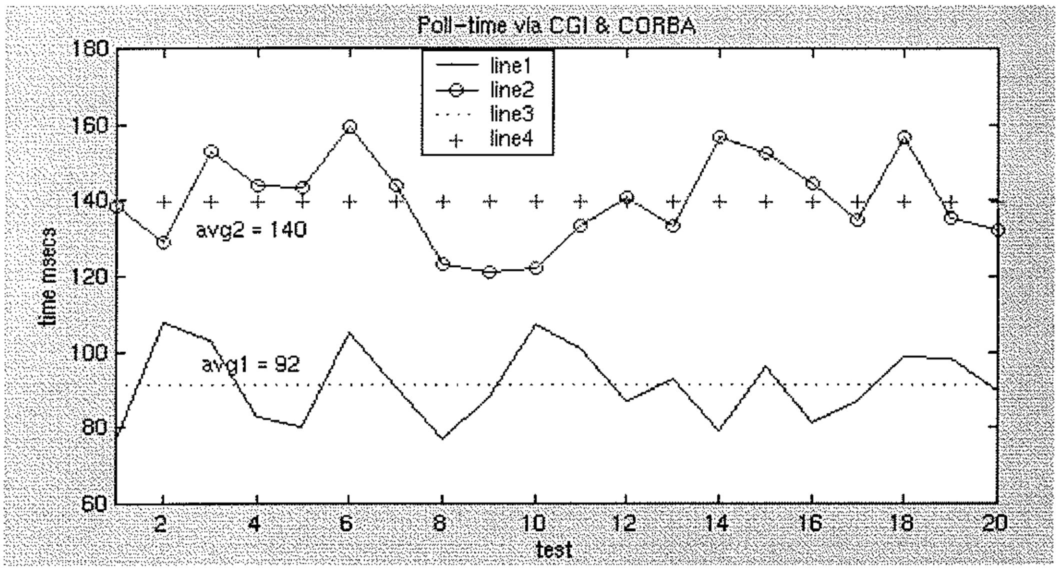


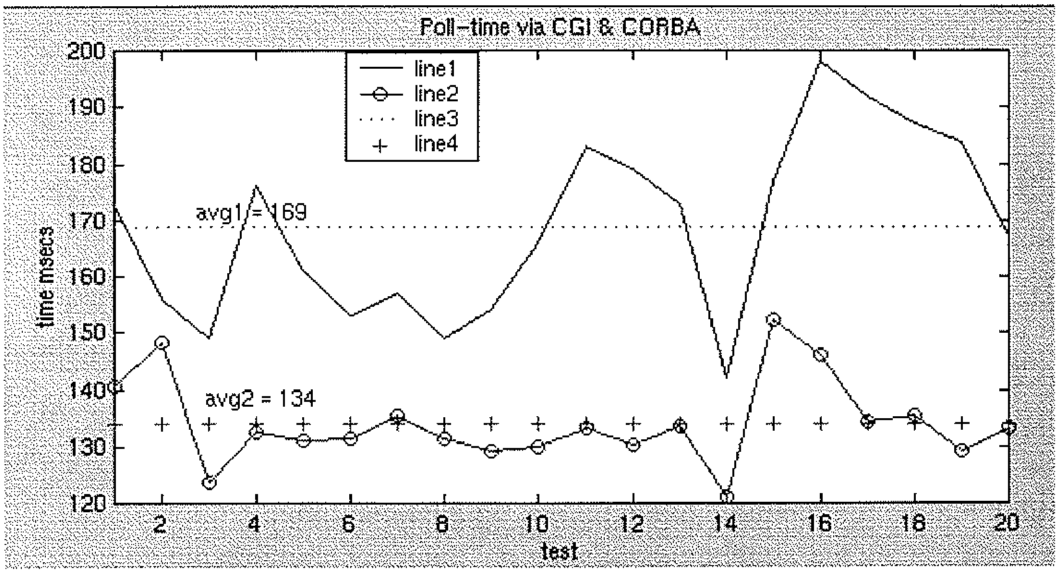
Figure 6.4 The Real Time Graph of IP Packets Statistics of cider



(1) line1 — CORBA case (2) line2 — CGI case, the value in Fig. = **real/100**

(3) line3 — average value for line1 (4) line4 — average value for line2

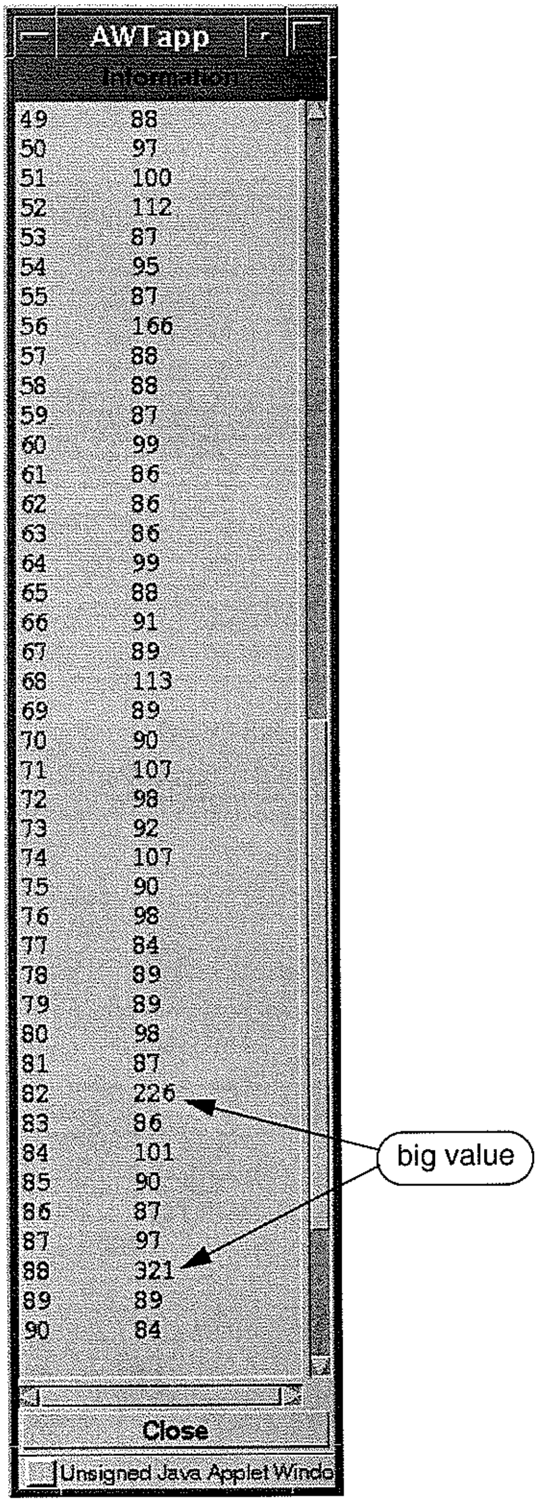
Figure 6.5 Poll Time Graph (SNMP Server Running on the Web Server in the CORBA Case)



(1) line1 — CORBA case (2) line2 — CGI case, the value in Fig. = **real/100**

(3) line3 — average value for line1 (4) line4 — average value for line2

Figure 6.6 Poll Time Graph (SNMP Server Not Running on the Web Server in CORBA Case)



```
49    88
50    97
51    100
52    112
53    87
54    95
55    87
56    166
57    88
58    88
59    87
60    99
61    86
62    86
63    86
64    99
65    88
66    91
67    89
68    113
69    89
70    90
71    107
72    98
73    92
74    107
75    90
76    98
77    84
78    89
79    89
80    98
81    87
82    226
83    86
84    101
85    90
86    87
87    97
88    321
89    89
90    84
```

Figure 6.7 The Information Popup Window in CORBA Application (SNMP Server Running on the Web Server)

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

- This thesis reviewed SNMP technology and realized Web-based network monitoring using this technology. The basic information of System Group, Interface Group and Ip Group under MIB-II can be retrieved and both text and real time graphs can be displayed in the client's Web browser. With my implementations, SNMP is a very good protocol for network monitoring. It is simple and effective. Due to its popularity, we can easily get support by vendors, such as *snmpd* from Sun's Solaris 2.6 and *snmp.exe* from Microsoft Window NT. If we want to monitor or control some specific parameters, we can design our MIB and SNMP agent ourselves based on ASN.1, RFC 1155 and RFC 1157, etc.
- This thesis reviewed some concepts and methodologies of the OO approach and implemented them in UML. UML is now a standard notation adopted by OMG for object-oriented design. In my application, I:
    - Investigated and defined the requirements for the system.
    - Identified the potential classes that compose the system.
    - Mapped the system requirements to class attributes and operations.
    - Created class diagrams and sequence diagrams to capture most of the information about the system in order to help the system design.
- Most of programs developed in my applications are written in Java. The implementation shows that Java is very good for network programming, interface constructing and Web-

embedding (using Java applets). It is simple and object oriented. Also due to its platform independence, the program developed can be running on any machine.

- CGI and CORBA technologies were reviewed and realized in the Web-based network monitoring applications. From the programming view, CORBA's local/remote transparency benefits the programmer considerably. While programming for the client, the programmer does not have to be concerned with transport, server location, object activation, byte ordering, rather just simply invokes the methods of services on the server side as if they are located in the same environment/directory. Using CGI, this becomes clumsy and difficult because we need to specify the location of the server either in an URL or socket, interpret all the message and data either to a string for transmission or from a string for information. From the implementation view, we can see that applications via CGI are much slower than the CORBA peer. Also when the SNMP server is running on the Web server, the CORBA applications are even faster. Furthermore, since the Snmp Server in the CORBA application can be running on any machine, the load on the Web server can be decreased.
- All of the systems (*NMS1_CGI*, *NMS2_CGI*, *NMS1_CORBA* and *NMS2_CORBA*) were tested and they work smoothly. Each of them can be accessed by Web clients and realize the desired functions. They were tested on Unix workstations, PCs and Macintoshes and both Netscape (version 4.5 or later) and Explorer (version 4.0 or later).
- The applications developed in this thesis only show part of the parameters defined in MIB-II. They are easily extended to monitor other parameters defined in MIB-II based on the requirements by adding a graphical interface for the OIDs in the queries.
- Nowadays there are many SNMP packages, one of them is AdventNet SNMP API. Besides this kit, there is another one named AdventNet NetMonitor (also developed by Advent Inc.),

which is a graphical applet builder tool to be used to build Java applets that monitor and control the machines (with SNMP support) in the network. The problem is that it is difficult to add your components when you use this tool to build a graphical user interface. Also there are other limitations, such as you can only monitor the two interfaces for total data in and out of the queried machine. In my application, we can dynamically monitor all up interfaces of the queried machine for total data in and out.

7.2 Future Work

Because of limited time and access rights on our network, there is more work that can be done within this area in the future. For reference, several of them are listed below:

- The applications developed in this project only realize the monitoring part because of access rights to our local network. It is easily extended to the controlling part if we have the write access to the configuration files of the system (set pdu.command = *SET_REQ_MSG*, see [AdventNet]).
- My applications only touched the basic SNMP set: RFC 1155, RFC 1213 and RFC 1157. For further requirements, we can try the Web-based network management by exploring RMON, SNMP v2 or SNMP v3. Furthermore, we can design our MIBs and SNMP agents for some special usages.
- For CORBA applications in my thesis, I have only applied static method invocations. For more advanced usage, we can try dynamic method invocations.
- With CORBA involved, we can get the SNMP trap message in the client's browser by using the CORBA's callback via IIOP. Callback is a call from the server to a client and in this way we can make the client aware what is happening on the server side.
- In my applications I only designed the CORBA objects using Java. Since CORBA provides

high level language bindings, we can design some objects in C++ with the help of *VisiBroker for C++* and then make the Java objects invoke those methods and vice versa.

- In my application, the Snmp Server needs to contact the SNMP agents to get the information according to the client's request. In the future we can try the following: three servers on the server side, one responsible for periodically contacting SNMP agents of the managed nodes for all the information defined in the MIB and write this information to a database with the help of a database server, a third server would be responsible for taking care of the client's requests and reading the queried information from the database.
- In this thesis I explored my applications with both CGI and CORBA implementations. There is another popular kit named servlet which was introduced by JavaSoft to extend the Web server's service for handling client requests [Servlet]. It is widely used now to replace the CGI scripts. We may apply the servlets in our Web-based network management applications and compare the performance of it with CORBA and CGI.

References:

[AdventNet] "Advent SNMP Package, version1.2", "http://www.adventnet.com"

[Ben-Artzi 90] Ben-Artzi A., Chandna A., and Warrier U., "Network Management of TCP/IP Networks: Present and Future", IEEE Network Magazine, pp. 35-43, July 1990

[Berners-Lee 96] Berners-Lee T., Fielding R. & Frystyk H., "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996

[Blight 97] Blight D., Graduate Course Notes: "Telecommunication Networks", University of Manitoba, "http://www.ee.umanitoba.ca/~blight/c24759.html", 1997

[Borenstein 93] Borenstein N., Freed N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993

[Case 90] Case J., Davin J., Fedor M., Schoffstall M., "A Simple Network Management Protocol (SNMP)", RFC 1157, May, 1990

[Cerf 88] Cerf V., "IAB Recommendations for the Development of Internet Management Standards", RFC 1052, April, 1988

[OCI] Comparison of IONA/OTM and Inprise ITS technology, "http://www.objectconsulting.com/reportframe.html"

[CORBA] CORBA, "CORBA/IIOP2.1", Revision 2.1, August 1997

[Corcoran 96] Corcoran C., "Managing Network ills", October 14, 1996, Vol. 18, Issue 42, Enterprise Computing

[Enck 97] Enck J., Blacharski D., "Managing Multivendor Networks", April, 1997

[Harmon 98] Harmon P., Watson M., "Understanding UML: The Developer's Guide -- With a Web-Based Application in Java", 1998

[Jacobson 92] Jacobson I., Christerson M, et al., "Object-Oriented software Engineering: A Use Case Driven Approach", 1992

[Jander 96] Jander M., "Web-Based Management: Welcome to the Revolution", Data Communications, Vol. 25, Iss. 16, pg. 38, Nov., 1996

[JavaSoft] "The Java Language: An Overview", "http://java.sun.com/docs/overviews/java/java-overview-1.html"

[Mazumdar 96] Mazumdar S., "Inter-Domain Management between CORBA and SNMP: WEB-based Management - Corba/Snmp Gateway Approach", "http://www.bell-labs.com/~mazum/", 1996

[McCloghrie 91] McCloghrie K., Rose M., "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, March, 1991

[Mowbray 97] Mowbray T., Ruh W., "Inside CORBA: distributed object standards and applications", 1997

[OMG] "http://www.omg.org/corba/whatiscorba.html"

[Orfali 97] Orfali R. and Harkey D., "Client/Server Programming with Java and CORBA", 1997

[Patchett 97] Patchett C., Wright M., "The CGI/Perl Cookbook", 1997

[Perkins 96] Perkins D., McGinnis E., "Understanding SNMP MIBs", 1996

[Pooley 99] Pooley R. J., "Using UML: Software engineering with objects and components", 1999

[Rose 90] Rose M.T., McCloghrie K., "Structure and identification of management information for TCP/IP-based internets", RFC 1155, May, 1990

[Rosenberger 98] Rosenberger J., "Teach Yourself CORBA in 14 Days", January 1998

[Rumbaugh 98] Rumbaugh J., Jacobson I., Booch G., “The Unified Modelling Language Reference Manual”, Dec., 1998

[Servlet] “http://java.sun.com/products/servlet/2.1”

[Snmpv3] “http://www.ietf.org/html.charters/snmpv3-charter.html”

[Snmp FAQ] “http://www.cis.ohio-state.edu/hypertext/faq/usenet/snmp-faq/top.html”

[Stallings 96] Stallings W., “SNMP,SNMPv2, and RMON”, 2nd edition, 1996

[Stallings 98] Stallings W., “SNMPv3: A Security Enhancement for SNMP”, “http://www.comsoc.org/pubs/surveys/4q98issue/stallings.html”, 1998

[Tittel 96] Tittel E., Gaither M., et. al. “Web Programming Secrets with HTML, CGI, and Perl”, 1996

[VBJ33] “VisiBroker for Java 3.3, Programmer’s Guide”, “http://www.inprise.com/techpubs/books/vbj/vbj33/index.html”, 1998

[VisiBroker] “http://www.inprise.com/visibroker/”

[WBEM FAQ] “http://www.dmtf.org/pres/rele/faq.html”

[WBMP] Web Based Management Page, “http://www.mindspring.com/~jlindsay/web-based.html”

[Weinman 96] Weinman W., “The CGI Book”, 1996