1

**PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT**

PRECISIONSM RESOLUTION TARGETS

Manager-To-Manager Communications Using OSIMIS

by

CARLOS LEI, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

August, 1996

© copyright

1996, Carlos Lei

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file / Votre référence

Our file / Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-19497-3

Canada

Dissertation Abstracts International and Masters Abstracts International are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

SUBJECT TERM

0544

UMI

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Jornalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0446

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0488
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0480
Higher	0745
History of	0520
Home Economics	0278
Industrial	0821
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0823

Psychology	0525
Reading	0535
Religion	0527
Science	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0479
Ancient	0287
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0391
Asian	0305
Canadian (English)	0352
Canadian (French)	0353
English	0393
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0301
Agricultural	0303
Commerce-Business	0305
Finance	0308
History	0309
Labor	0310
Theory	0311
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0378

Ancient	0579
Medieval	0381
Modern	0382
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0385
Law	0398
Political Science	
General	0415
International Law and Relations	0416
Public Administration	0417
Recreation	0814
Social Work	0482
Sociology	
General	0426
Criminology and Penology	0427
Demography	0938
Ethnic and Racial Studies	0431
Individual and Family	
Studies	0438
Industrial and Labor	
Relations	0429
Public and Social Welfare	0430
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0474
Food Science and Technology	0389
Forest and Wildlife	0476
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746

Biology	
General	0306
Anatomy	0287
Botanics	0308
Ecology	0309
Cell	0309
Evolution	0309
Genetics	0309
Immunology	0309
Microbiology	0410
Molecular	0309
Neuroscience	0317
Oncology	0416
Physiology	0309
Plant	0309
Physiology	0309
Veterinary Science	0778
Zoology	0309
Botany	
General	0786
Plant	0786

EARTH SCIENCES

Geography	0485
Geology	0796

Geology	0370
Geography	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Palaeontology	0345
Paleogeography	0426
Paleobotany	0418
Paleozoology	0985
Physiology	0427
Physical Geography	0348
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0344
Anatomy	0300
Chemotherapy	0992
Dentistry	0347
Education	0330
Hospital Management	0769
Human Development	0736
Immunology	0982
Medicine and Surgery	0344
Mental Health	0347
Nursing	0349
Nutrition	0350
Oncology and Gynecology	0380
Occupational Health and Safety	0364
Ophthalmology	0381
Physiology	0351
Pharmacology	0379
Pharmacy	0379
Physical Therapy	0382
Public Health	0374
Radiology	0374
Recreation	0375

Speech Pathology	0460
Toxicology	0383
Home Economics	0384

PHYSICAL SCIENCES

Pure Sciences	
Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0406
Acoustics	0984
Astronomy and Astrophysics	0406
Atmospheric Science	0408
Atomic	0748
Electronics and Electricity	0407
Elementary Particles and High Energy	0798
Fluid and Plasmas	0799
Materials	0409
Nuclear	0310
Optics	0782
Radiation	0784
Solid State	0411
Statistics	0443
Applied Sciences	
Applied Mechanics	0346
Computer Science	0984

Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronic and Electrical	0544
Heat and Thermodynamics	0545
Hydraulic	0546
Industrial	0546
Marine	0547
Materials Science	0548
Mechanical	0548
Metallurgy	0549
Mining	0551
Nuclear	0552
Packaging	0553
Petroleum	0554
Sanitary and Municipal	0554
System Science	0555
Geotechnology	0556
Operations Research	0556
Plastics Technology	0557
Tissue Technology	0558

PSYCHOLOGY

General	0421
Behavioral	0384
Clinical	0382
Developmental	0383
Experimental	0383
Industrial	0384
Personality	0385
Physiological	0386
Psychiatry	0387
Psychometrics	0388
Social	0389

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

**“Manager-To-Manager Communications
Using OSIMIS”**

submitted by **Carlos Lei, B. Eng.**

in partial fulfillment of the requirements for
the degree of Master of Engineering

Department Chair

Thesis Supervisor, **Dr. Bernard Pagurek**

CARLETON UNIVERSITY
August, 1996

Abstract

OSI network management provides powerful management mechanisms and offers effective management solutions. The thesis explores a management platform, namely OSIMIS, for providing OSI network management and presents an implementation of a hierarchical manager-to-manager communications scheme using the OSIMIS platform.

Acknowledgments

I would like to thank my thesis supervisor Dr. Pagurek for his guidance and encouragement during the course of this research. With his knowledge and enthusiasm in this area of research, and through numerous valuable discussion periods, a novice like me has gradually learned to discover the beauty and the complexity of network management, and to appreciate its application.

I would also like to express my gratitude to my family and my beloved girlfriend Maggie. Without their steady love and encouragement, through what was a difficult period of my life, none of this would have been possible.

Table of Contents

Abstract.....	iii
Acknowledgments.....	iv
List of Figures.....	viii
List of Acronyms.....	x
CHAPTER 1: INTRODUCTION	
1.1 Background.....	1
1.2 Overview of the Research.....	3
1.3 Outline of the Thesis.....	4
CHAPTER 2: OSI NETWORK SYSTEM MANAGEMENT	
2.1 Overview of OSI Network Management.....	6
2.2 OSI System Management Concepts.....	7
2.2.1 Management Functional Areas.....	8
2.2.2 Systems Management Functions.....	9
2.3 OSI System Management Model.....	11
2.3.1 Management Information Model.....	12
2.3.1.1 Basic Concepts of the Information Model.....	12
2.3.1.2 Principles of Containment and Naming.....	19
2.4 Communications Model.....	23
2.4.1 Common Management Information Service (CMIS).....	24
2.4.2 Common Management Information Protocol (CMIP).....	29
2.4.3 Performance.....	30
CHAPTER 3: THE OSIMIS MANAGEMENT PLATFORM	
3.1 Overview of OSIMIS.....	33
3.2 The ISO Development Environment.....	35
3.3 Management Protocol and High-Level Abstract Syntax Support.....	36
3.4 Application Coordination Support.....	38
3.4.1 The Mechanism of the Coordinator.....	40
3.4.1.1 The Central Listening Loop.....	41
3.4.1.2 The Coordinator Support.....	42
3.5 The Generic Managed System (GMS).....	44
3.5.1 Managing Real Resources.....	45

3.5.2	Managed Objects.....	46
3.5.3	An Agent Application.....	48
3.5.4	System Management Functions.....	50
3.6	Generic High-Level Manager Support.....	50
3.7	OSIMIS Applications.....	52
3.7.1	Generic Managers.....	53
3.7.2	The Generic CMIS/SNMP Application Gateway.....	53
3.7.3	Generally Useful Agents.....	54

CHAPTER 4: DEMONSTRATIONS OF THE OSIMIS APPLICATIONS

4.1	Overview of the Demonstrations.....	55
4.2	Realizing the Agent-Manager Model of Network Management with an OSI Agent.....	55
4.2.1	Adding a New MO class to the OSIMIS Environment	56
4.2.2	Manipulating the MO Instances.....	62
4.2.3	Modifying an existing MO class.....	66
4.3	Realizing Event Report Forwarding Between Management Entities.....	67
4.3.1	The OSI Event-Report-Management Function.....	67
4.3.2	Introduction to an OSIMIS Generic Manager Application - Evsink.....	69
4.3.3	The Syntax of Evsink.....	70
4.3.4	A Working Example of Using Evsink.....	71
4.3.5	More General Usages of Evsink.....	74
4.4	Realizing the Agent-Manger Model with a SNMP Agent.....	75
4.4.1	Introduction to the OSIMIS Internet Q-Adaptor (IQA).....	75
4.4.2	The Structural Decomposition of IQA.....	78
4.4.3	The General Build Process.....	80
4.4.4	A Working Example of Using the IQA	82
4.5	Providing Thresholding Capabilities to MO Instances.....	93
4.5.1	Modifying the GDMO File.....	94
4.5.2	Using OSI Metric Monitoring Management Function.....	96
4.5.2.1	Providing Thresholds to MOs for an OSI Agent....	98
4.5.2.2	Providing Thresholds to the Attributes for a SNMP Agent.....	100

CHAPTER 5: IMPLEMENTING HIERARCHICAL MANAGER-TO-MANAGER COMMUNICATIONS

5.1	Implementing Manager-To-Manager Communications.....	103
5.1.1	The Manager-To-Manager (M2M) MIB for SNMPv2.....	104
5.1.2	Comparison of the M2M MIB with the OSIMIS Evsink.....	105
5.2	Implementing Hierarchical Manager-To-Manager Communication using OSIMIS.....	107

5.2.1	Approaching a Desirable Implementation.....	107
5.2.2	Building Up the Selected Implementation.....	114
5.2.2.1	Building the thldMonitor MO.....	114
5.2.2.2	Modifying the Evsink.....	115
5.3.3	A Working Example.....	118
5.3.4	Performance of the Selected Implementation.....	123
5.3.5	Recommendations.....	125

CHAPTER 6: CONCLUSIONS

6.1	Summary Of Results.....	130
6.2	Suggestions for Future Work.....	134

REFERENCES	136
------------------	-----

List of Figures

- Figure 2.1 Application Interactions in OSI Network Management
- Figure 2.2 Inheritance Example
- Figure 2.3 Containment Tree, with Relative Distinguished Names.
- Figure 2.4 CMIS Services
- Figure 3.1 The OSIMIS Process Coordination Model
- Figure 3.2 The overall structure of a managed object instance
- Figure 3.3 The GMS Object-Oriented Architecture
- Figure 3.4 The Remote and Shadow MIB Support Models
- Figure 4.1 A Simple Network Management System
- Figure 4.2 Event-Report-Management Model
- Figure 4.3 The Overall Picture of the Demonstration for Evsink.
- Figure 4.4 Manager/Agent Communication Paths
- Figure 4.5 The Three Cases of IQA Mappings
- Figure 4.6 The Structural Decomposition of the IQA
- Figure 4.7 General Build Process of the OSIMIS IQA
- Figure 4.8 OSIMIS Proxy Directory Structure
- Figure 4.9 A Network Management System with a SNMP agent.
- Figure 4.10 The Metric Monitoring Model
- Figure 4.11 Providing thresholds to SNMP attributes
- Figure 5.1 Comparing M2M MIB with Evsink.

- Figure 5.2 The First Approach for Implementing Hierarchical M2M Communications.**
- Figure 5.3 Interactions in the Mid-Level Manager Station of the First Approach.**
- Figure 5.4 Interactions in the Mid-Level Manager Station of the Second Approach.**
- Figure 5.5 The Third Approach of Implementing Hierarchical M2M Communications.**
- Figure 5.6 The Layout of the Demonstration.**
- Figure 5.7 A Hierarchical M2M Communications Network.**

The OSIMIS platform provides the foundation for construction of complex management systems. It is an object-oriented development environment in C++ based on the OSI Management Model. The only thing that user needs when using OSIMIS is to have a thorough understanding of the underlying management protocol, while OSIMIS hides all the implementation complexity of the protocol and harnesses the power as well as the expressiveness of the associated information model through Application Program Interfaces (APIs)[4]. OSIMIS also combines the OSI management power with the large installed base of Internet SNMP capable network elements[2].

In addition, another benefit of using OSIMIS is that, currently, it can be freely obtained from University College London under an educational license. However, since it is not commercially available, there is an enormous lack of information and support for instructing people who are interested in implementing OSI network management using the OSIMIS platform.

1.2 Overview of the Research

The first objective of this research is to explore OSIMIS for providing complex CMIP services. The second is to accomplish a hierarchical manager-to-manager communications scheme by integrating the features of OSIMIS.

In order to achieve these two objectives, this research has the following major goals in using the OSIMIS platform:

- To realize an OSI network management system with an OSI agent as well as with a SNMP agent.

ISO	- International Organization for Standardization
ISODE	- ISO Development Environment
LAN	- Local Area Network
MAN	- Metropolitan Area Network
MIB	- Management Information Base
MIT	- Management Information Tree
MO	- Managed Object
MSAP	- Management Service Access Point
NMF	- Network Management Forum
OMG	- Object Management Group
OSI	- Open System Interconnection
OSIMIS	- OSI Management Information Service
PDU	Protocol Data Unit
PE	- Presentation Element
PSAP	- Presentation Service Access Point
RFC	- Request For Comments
ROSE	- Remote Operations Service Element
SMA	- System Management Agent
SMF	- System Management Function
SMFA	- System Management Functional Area
SMI	- System Management Information
SNMP	- Simple Network Management Protocol

SNMPv2	- Simple Network Management Protocol Version 2
TCP	- Transmission Control Protocol
TMN	- Telecommunications Management Network
UDP	- User Datagram Protocol

CHAPTER 1

INTRODUCTION

1.1 Background

Networks and distributed processing systems are of growing importance and, indeed, have become critical in the business world. Within a given organization, the trend is toward larger, more complex networks supporting more applications and more users. As these networks grow in scale, two facts, as mentioned in [1], become painfully evident. Firstly, the network and its associated resources and distributed applications become indispensable to the organization. Secondly, more things can go wrong, disabling the network or a portion of the network or degrading performance to an unacceptable level.

Also, today's typical organization has a large and growing but amorphous architecture, with a variety of local area networks (LANs) and wide area networks (WANs), supported by bridges and routers, and a variety of distributed computing services and devices. Distributed computing environments require more system administrators, typically one for every 30-50 workstations, as opposed to one per 200 workstations in a proprietary, centralized environment. It is usually too expensive to fill the additional system administrator slots with senior people. Even companies willing to pay the price for senior people have difficulty filling the positions[10].

For these reasons, a large network can no longer be put together and managed by human effort alone. The complexity of such a system dictates the use of automated

network management tools. As networked installations become larger, more complex, and more heterogeneous, the cost of network management rises. In order to control costs, standardized tools are needed that can be used across a broad spectrum of product types and in a mixed vendor environment[1].

In response to this need, two standardization efforts, namely the *SNMP (Simple Network Management Protocol) family* and the *OSI (Open System Interconnection) systems management*, are underway[1]. The adoption of OSI network management protocol continues to be slow in regard to corporate data networks. The widespread availability of network management systems based on SNMP for many Information Technology (I.T.) managers today, may provide a more cost effective solution.

However, the deployment of OSI network management protocol in large networks, especially in the Telecommunications sector (e.g.: the Telecommunications Management Network), is now becoming more evident. It is therefore likely that over the next five years, these OSI based systems will become widely available at a cost effective price for corporate data networks, both large and small[11].

OSI network management provides rich and powerful management mechanisms, which can be used to offer effective, extensible and scaleable management solutions. However, there has been a belief, amplified by the lack of implementations, that OSI management facilities are difficult to implement. For this reason, management platforms have been developed to help researchers or network managers to implement OSI network management. One of these, developed by University College London, UK., is called OSIMIS (OSI Management Information Services)

The OSIMIS platform provides the foundation for construction of complex management systems. It is an object-oriented development environment in C++ based on the OSI Management Model. The only thing that user needs when using OSIMIS is to have a thorough understanding of the underlying management protocol, while OSIMIS hides all the implementation complexity of the protocol and harnesses the power as well as the expressiveness of the associated information model through Application Program Interfaces (APIs)[4]. OSIMIS also combines the OSI management power with the large installed base of Internet SNMP capable network elements[2].

In addition, another benefit of using OSIMIS is that, currently, it can be freely obtained from University College London under an educational license. However, since it is not commercially available, there is an enormous lack of information and support for instructing people who are interested in implementing OSI network management using the OSIMIS platform.

1.2 Overview of the Research

The first objective of this research is to explore OSIMIS for providing complex CMIP services. The second is to accomplish a hierarchical manager-to-manager communications scheme by integrating the features of OSIMIS.

In order to achieve these two objectives, this research has the following major goals in using the OSIMIS platform:

- To realize an OSI network management system with an OSI agent as well as with a SNMP agent.

- To provide event report forwarding capability between management entities.
- To provide thresholding capabilities to both an OSI agent and a SNMP agent.
- For the initial implementation, the hierarchical manager-to-manager communications scheme is to support a hierarchy of three levels, each of which consists of one management entity. However, the implementation must allow future extensions to support a hierarchy with more levels and management entities.

These goals were met and detailed demonstrations of the OSIMIS applications are described in this thesis. The hierarchical manager-to-manager communications scheme meets the primary goal of supporting a management hierarchy of three levels, and can be extended to support more levels with more management entities. Also, with the use of the OSIMIS generic application gateway, not only OSI agents can be integrated into the scheme, but also SNMP agents as well.

1.3 Outline of the Thesis

This thesis is organized as follows: Chapter 2 introduces the reader to OSI network management including both its information model and communications model. Since this thesis is the first one about OSI network management in the department, the information provided in this chapter is more extensive than usual. However, this chapter may be omitted by a reader already familiar with OSI network management concepts. Chapter 3 describes the infrastructure of OSIMIS and the services it offers for implementing OSI network management. Chapter 4 presents detailed demonstrations of the OSIMIS applications. Chapter 5 describes the implementation of a hierarchical

manager-to-manager communications scheme using the OSIMIS platform. Chapter 6 concludes the results obtained in this thesis and describes the directions to which the results can be extended.

CHAPTER 2

OSI NETWORK SYSTEM MANAGEMENT

2.1 Overview of OSI Network Management

The OSI network management model is based on the object oriented paradigm[4]: management of physical or logical real resources is enabled through abstractions of them known as managed objects. These and their properties are formally specified in an abstract form, and this specification together with the access management protocol CMIS/P identify uniquely the interoperable interface for management applications.

The managed object classes are organized in an inheritance hierarchy. The managed objects constituting a Management Information Base (MIB) instance in a managed system are organized in a Management Information Tree (MIT), according to the containment relationships.

Basically, an inheritance relationship specifies that an object is a kind of another object, while a containment one specifies that an object is a part of another object. Figure 2.1 shows the interactions which take place between the management application entities in the OSI network management environment[12].

Managed object classes related to various communications resources are specified by standards groups in a formal language called Guidelines for the Definition of Managed Objects (GDMO). The management information schema, its inheritance and containment

hierarchies together with the management attributes, operations, actions, notifications and behaviour of each managed object classes are formally specified in that language.

Figure 2.1: Application Interactions in OSI Network Management

2.2 OSI System Management Concepts

OSI systems management is defined by a large set of standards issued jointly by the ISO (International Organization for Standardization) and the CCITT (International Consultative Committee on Telegraphy and Telephony). The overall framework for OSI systems management is designed to satisfy network management requirements in five functional areas[1]: fault management, accounting management, configuration management, performance management and security management.

To support these five functional areas, OSI includes a number of general-purpose tools known as systems-management functions. Each functional area can be implemented as an application that relies on some subset of the systems-management functions. These functions, in turn rely on the common management information service (CMIS) for the basic exchange of management information.

2.2.1 Management Functional Areas

The OSI network management divides the task of systems management into five functional areas. These areas provide a useful checklist for assessing any network management offering. A brief description of each of them is given in the following:

1) Fault Management

OSI fault-management facilities allow network managers to detect problems in the communications network and the OSI environment. These facilities include mechanisms for the detection, isolation, and correction of abnormal operation in any network component or in any of the OSI layers.

2) Accounting Management

OSI accounting-management facilities allow a network manager to determine and allocate costs and charges for the use of network resources.

3) Configuration and Name Management

OSI configuration- and name-management facilities allow network managers to exercise control over the configuration of the network components and OSI layer entities.

Configuration may be changed to alleviate congestion, isolate faults, or meet changing user needs.

4) Performance Management

OSI performance-management facilities provide the network manager with the ability to monitor and evaluate the performance of system and layer entities.

5) Security Management

OSI security-management facilities allow a network manager to manage those services that provide access protection of communications resources.

2.2.2 Systems Management Functions

The management functional areas defined in the OSI management framework describe broad areas of network management responsibility. Each of these areas involves the use of specific functions and there is considerable overlap in these support functions. Accordingly, the five functional areas are not standardized as such. Rather, a number of specific functions, referred to as systems management functions (SMFs), have been defined. Each SMF standard defines the functionality to support systems management functional area (SMFA) requirements. A given SMF may support requirements in one or more SMFAs; for example, the event-report management function may be applicable to all SMFAs. Looked at the other way, each SMFAs requires several SMFs.

Each of the SMF standards defines the functionality for the SMF and provides a mapping between the services provided by the SMF and CMISE (Common Management Information Service Element). Each of the five management functional areas makes use of

one or more of the systems management functions. Each systems management function may make use of the services of other SMFs as well as the services of CMISE.

So far, 13 SMFs have been specified. These are:

1. *Object management*: supports the creation and deletion of managed objects and the reading and changing of object attributes. Also specifies notifications to be emitted when the value of an attribute changes.
2. *State management*: specifies a model for how the management state of an object is to be represented. Provides services to support the model.
3. *Relationship management*: specifies a model for representing and managing relationship between managed objects.
4. *Alarm reporting*: supports the definition of fault alarms and the notifications used to report them.
5. *Event report management*: supports the control of event reporting, including the specifications of recipients of reports, the definitions of reports, and the specification of criteria for generating and distributing reports.
6. *Log control*: supports the creation of logs, the creation and storage of log records, and the specification of criteria for logging.
7. *Security alarm reporting*: supports the definition of security alarms and the notifications used to report them.
8. *Security-audit trail*: specifies the kinds of event reports that should be continued in a log used for security evaluation.

9. *Access control*: supports the control of access to management information and operations.
10. *Accounting meter*: provides for accounting for the usage of system resources and a mechanism for enforcing accounting limits.
11. *Workload monitoring*: supports the monitoring of attributes of managed objects that relate to the performance of a resource.
12. *Test management*: supports the management of confidence and diagnostic test procedures.
13. *Summarization*: supports the definition of statistical measures to be applied to attributes and the reporting of summarized information.

2.3 OSI System Management Model

OSI systems management relies heavily on the concept of object-oriented design. Each resource that is monitored and controlled by OSI systems management is represented by a managed object. The MIB is a structured collection of such objects. A managed object can be defined for any resource that an organization wishes to monitor and/or control. Examples of hardware resources are switches, workstations, etc. and that of software resources are queuing programs, and buffer management routines. Several important points about managed objects need to be keep in mind[1]:

- A managed object is an abstraction that is directly available to the systems management function. Some other mechanism, outside the scope of the OSI

management standards, maintains the relationship between the managed object and the actual resource.

- A single managed object may represent a single network resource or many resources.
- The same network resource may be represented by a single managed object or by a number of different managed objects, each of which represents a particular aspect of the resource.
- Not all resources need to be represented by a managed object. This does not mean that such resources do not exist, only that they are not available for OSI systems management.
- Some managed objects are defined solely for the support of management functions and do not represent resources. Examples include event logs and filters.

2.3.1 Management Information Model

2.3.1.1 Basic Concepts of the Information Model

A managed object is defined in terms of attributes it possesses, operations that may be performed upon it, notifications that it may issue, and its relationship with other managed objects. In order to structure the definition of a MIB, each managed object is an instance of a managed object class. A managed-object class is a model or template for managed-object instances that share the same attributes, notifications, and management operations.

The specifications of the OSI SMI (System Management Information) and MIB rely heavily on concepts of object-oriented design. This approach allows for new managed-object classes and functions to be added, as needs are identified, in a modular fashion. The object-oriented approach also provides for extensibility of the related protocols and services. It should be kept in mind that the specifications do not dictate that MIBs be implementing using object-oriented database management systems or object-oriented technology. The only requirement is that the specification of the information conveyed between open systems in systems management protocols, e.g. CMIP (Common Management Information Protocol), use object-oriented design principles.

Encapsulation

Encapsulation is a fundamental characteristic of an object-oriented system. In the network management context, encapsulation has the following significance: Each type of resource to be managed in the system is represented by a managed object class. A specific instance of that resource is represented by a managed object instance. The management data relating to that resource and the management procedures applicable to that resource are packaged together (encapsulated) in the corresponding object. Management applications have access to the resource, for control and monitoring, only by means of the corresponding object. Furthermore, all operations on the managed object are carried out by means of messages sent to the object. The actual data and procedures encapsulated in the object are protected from the outside world. Thus, encapsulation ensures that the integrity of an object is preserved.

Attributes

The actual data elements contained in a managed object are called attributes. Each attribute represents a property of the resource that the object represents, such as the operational characteristics, current state, or conditions of operation. Attributes are most commonly used for control, with the setting of an attribute value causing a change in the behaviour or status of the underlying resource.

The data type of an attribute may be integer, real, Boolean, character string, or some composite type constructed from the basic types. In addition to a data type, each attribute has access rules (read, write, read-write) and rules by which it can be located as the result of a filtered search (matching rules)

An attribute can be a simple scalar variable. Read (get) and write (set, replace) operations are possible on scalar attributes. In addition, an attribute may be set-valued, as defined by the ASN.1 SET-OF construct. Recall that a set-of type is an unordered, variable number (zero or more) of elements, all of one type. In addition to the read and write operations that can be performed on all attributes, operations to add or remove elements from a set-valued attribute are possible.

Within an object class, some of the defined attributes may be grouped together to form an attribute group. The group is merely a convenience that allows the same operation to be performed on all the members of the group with a single operation by applying the operation to the group. The order in which the operation is applied to the group members is not specified.

Object Classes and Inheritance

A managed object class is a template that defines the management operations, attributes, packages, notifications, and behavior included in a particular type of object. All object instances that share these same elements are members of the same class. The individual object instances may differ in the values of their attributes. The class concept is thus a macro-type facility that allows a general type of object to be defined just once and then allows that definition to be reused many times for each actual instance of the object type.

More significantly, the class construct allows for the definition of new object classes in terms of existing classes. This process is referred to as specialization, and a new object class is referred to as a subclass of the class from which it is specialized. The use of the subclass concept has two significant advantages:

1. It allows the development of a class hierarchy, with a subclass, in turn, having its own subclasses. This structure mirrors the actual structure of resources to be modeled in almost every case.
 2. The subclass retains characteristics of its superclass, a concept known as inheritance. This minimizes the need to specify characteristics of individual objects.
- In the OSI systems management context, specialization is achieved by extending the characteristics of an object class in one of more of the following ways:
- The addition of new attributes
 - The extension or restriction of the range of an existing attribute
 - The addition of new operations and notifications

- The addition of arguments to existing operations and notifications
- The extension or restriction of the ranges of arguments to operations and notifications.

Unlike a general purpose object-oriented scheme, OSI systems management does not allow the definition of a subclass by deleting any of the characteristics of its superclass.

As an optional facility, multiple inheritance is allowed. This means that a scheme is specialized from more than one superclass and inherits the operations, attributes, notifications, packages and behaviour from each superclass. Although this allows for the greatest possible reuse of class definitions, it is a difficult design technique to use effectively.

All object classes ultimately derive from a unique object class referred to as *top*. This is the ultimate superclass, and the other object classes form an inheritance hierarchy with *top* as the root. Figure 2.2 is an example of a portion of an inheritance hierarchy.

Operations

Systems management operations apply to the attributes of an object or the managed object as a whole. An operation performed on a managed object can succeed only if the invoking managing system has the access rights necessary to perform the operation, and consistency constraints are not violated.

Behaviour

A managed object exhibits certain behavioural characteristics, including how the object reacts to operations performed on it and the constraints placed on its behaviour.

Figure 2.2: Inheritance Example

The behaviour of a managed object occurs in response to either external or internal stimuli. External stimuli consist of systems management operations delivered in the form of CMIP messages. Internal stimuli are events internal to the managed objects and its associated resource, such as timers.

All managed object instances of the same managed-object class exhibit the same behaviour. The behaviour defines:

- The semantics of the attributes, operations and notifications
- The response to management operations being invoked on the managed object
- The circumstances under which notifications will be emitted
- The dependencies between values of particular attributes
- The effects of relationships on the participating managed objects

Notifications

Managed objects are said to emit notifications when some internal or external occurrence affecting the object is detected. Notifications may be transmitted externally in a protocol or logged. Managing systems may request that some or all the notifications emitted by a managed object be sent to it. Notifications that are sent to a manager or to a log are contained in an event report.

Conditional Packages

A conditional package is a collection of optional attributes, notifications, operations, and behaviour that are either all present or all absent in a managed object. The condition under which a package is present is always a condition reflecting the capability of the underlying resource being modeled by the managed object. An example is the set of options of an X.25 protocol machine.

Allomorphism

Allomorphism, as defined for OSI systems management, is essentially a special case of the object-oriented concept of polymorphism. Polymorphism provides the ability to hide different implementations behind a common interface. For example, defining a unique *print* method for each kind of document in a system would allow any document to be printed by sending the message *print*, without concern for how that method was actually carried out for a given document.

Similarly, allomorphism provides the ability to implement different objects that present the same interface to management stations. Specifically, allomorphism refers to the ability of an instance of a subclass (called an allomorphic subclass) to resemble the

behaviour of, or emulate, its superclass (called an allomorphic superclass) as observed by the systems management protocols. A typical use of allomorphism would be to support the evolution of the MIB. A new object can be defined to emulate the behaviour of an older, obsolete object, so that a management station could still manage the object the same way[1].

2.3.1.2 Principles of Containment and Naming

As shown previously, the object-oriented subclass facility allows for the creation of an inheritance hierarchy (Figure 2.2), which reflects the relationship among various types of objects. It is important to realize that this hierarchy simply represents a convenience for defining a variety of object types with a minimum of text. It is also a useful structuring tool in designing objects for an MIB. However, the inheritance hierarchy does not reflect the structure of an actual MIB. This structure is defined using the object-oriented containment facility.

The Containment Structure

The containment facility allows one object to “contain” one or more other objects. Containment is achieved by including a reference to the subordinate (contained) object in the superior (containing) object. The reference is in the form of the object identifier of the subordinate object and is stored as the value of an attribute in the superior object. A subordinate managed object may be contained in only one superior managed object, enforcing the condition that the MIB structure be a tree structure.

A containing object may itself be contained in another object, allowing the construction of a tree of arbitrary depth. Thus, the MIB structure can directly model real-world hierarchical structure, such as assembly, subassemblies, components and directory, files, fields.

Naming

Just as explained that there is a distinction between the inheritance hierarchy, which defines the relationship among object classes, and the containment hierarchy, which defines the relationship among object instances in the MIB, there is also a distinction between the naming scheme for object classes and that for object instances.

First, consider *object classes*. Each object class registered in the registration tree is identified by a unique object identifier. Each object identifier is a sequence of integers that navigates through the registration tree of assigned identifiers to the managed object class. This is the same scheme as is used in the SNMP MIB, except, in the case of SNMP, there is no structural difference between object classes and object instance: the same structure is exhibited.

The naming scheme for *object instances* is completely distinct from that for object classes and is dictated by the containment relationship. The naming scheme works as follows:

1. Each managed object class includes an attribute that is used in naming instances of that object.
2. The *relative distinguished name* of an object instance corresponds to a specific value of the naming attribute. This value must be unique among all objects that are

subordinate to the same superior. The actual form of a relative distinguished name is an assertion that an attribute has a particular value, e.g. MS-Id = "BDC", where "MS-Id" is the name of the attribute and "BDC" is the desired value.

3. The *distinguished name* of an object instance is formed as the sequence of relative distinguished names from the root of the containment tree to this object.

Figure 2.3 shows an example of a containment tree. For each object instance, its object class and its relative distinguished name are shown. Also, the distinguished name of each object instance are shown in the same figure as well.

It is important to note that a managed object instance name (the value of the naming attribute) is created when the instance is created. These names do not have to be registered or made public. They do have to be exchanged between interoperating managed systems to permit access to the object. Also, although the naming scheme is based on containment, not all forms of containment are necessarily used for naming. Containment can be used to create pointers between object instances that reflect the structure of the MIB and that go beyond a simple tree structure.

The Three Trees of OSI Systems Management

In brief, there are three distinct and independent tree structures used in OSI system management:

1. *ISO Registration Tree*: This is a naming tree where definition of the following are registered: managed object classes, attribute definitions, actions, notifications and packages. The registration tree may be thought of as a dictionary or library of "stubs" that can be stuck into new managed object class definitions. Since they are registered they have

Figure 2.3: Containment Tree, with Relative Distinguished Names.

well-known names and agreed upon semantics. This is an example of the benefit of re-use available when using object-oriented principles.

2. *Inheritance Tree*: This tree shows how the definition of object classes is derived from other object classes using object-oriented principles. Inheritance allows for re-use of an object class structure, with refinements to define a related but distinct object class.

3. *Containment Tree*: This is the MIB structure. It shows the objects an agent contains and the hierarchy/containment of those objects. This tree is used not only to define the MIB structure but as a means of unambiguously referencing object instances.

It is instructive to compare these structures with the MIB structure used in SNMP. For SNMP, there is a single tree structure, which is the ISO registration tree. This tree serves to define names for objects, as it does for OSI systems management. However, in the case of SNMP, all MIB objects are scalar or tables, and do not include attributes, notifications, and so forth. Therefore, the benefits of re-use that are available for OSI systems management are not available for SNMP.

In addition to providing a naming specification, the SNMP tree structure serves to define the SNMP MIB, much as the containment tree does for OSI systems management. The use of groups and tables provide rudimentary structuring tools. However, the structuring capability with OSI containment trees is richer and more flexible.

2.4 Communications Model

The fundamental function within OSI systems management is the exchange of management information between two entities (manager, agent) by means of a protocol. The functionality within OSI systems management is referred to as the common management information service element (CMISE). As with most areas of functionality within OSI, CMISE is specified in two parts[1]:

1. The interface with a user, specifying the services provided. This is the common management information service (CMIS).
2. The protocol, specifying the protocol data unit (PDU) format and associated procedures. This is the common management information protocol (CMIP).

CMISE users need to be able to establish associations in order to perform management operations. This service is provided by the association control service element (ACSE). For invoking an operation on a remote open system, the CMISE relies on the services of the remote operation service element (ROSE).

2.4.1 Common Management Information Service (CMIS)

The common management information service (CMIS) defines the services provided for OSI systems management. These services are invokable by management processes in order to communicate remotely.

The CMIS services are specified in terms of primitives that can be viewed as commands or procedure calls with parameters. These are listed in Figure 2.4. The services are of two types: confirmed services require that a remote management process send a response to indicate receipt and success or failure of the requested operation; nonconfirmed services do not provide responses.

Three categories of service are relevant to CMIS:

1. *Association service:* CMIS users need to establish an application association to communicate. They rely on the ACSE for the control of application associations.
2. *Management notification service:* This service is used to convey management information applicable to a notification. The definition of the notification and the consequent behaviour of the communicating entities is dependent on the specification of the managed object that generated the notification and is outside the scope of CMIS.

3. Management operation services: These six services are used to convey management information applicable to systems management operations. The definition of the operation and the consequent behaviour of the communicating entities is dependent on the specification of the managed object at which the operation is directed and is outside the scope of CMIS.

CMIS provides two structuring facilities:

1. Multiple responses to a confirmed operation can be linked to the operation by the use of a linked-identification parameter.
2. Operations can be performed on multiple managed objects, selected to satisfy some criteria and subject to a synchronizing condition.

Linkage

The M-GET, M-SET, M-ACTION, and M-DELETE service primitives can specify operation on multiple objects. When multiple objects are specified in a management request, one response is returned for each object. Some linkage technique is needed to match the multiple responses to the initial request that generated the responses. This linkage is necessary since there may be a number of outstanding requests and incoming responses must be matched to the correct preceding outgoing request.

Linkage is provided by means of the linked-identifier parameter, which appears in each of the response and confirm primitives. The value of the parameter is the same as the invoke identifier that appears in the request and indication primitives. The invoke identifier is a unique identifier assigned to each operation.

(a) Management - Notification Service		
Service	Type	Definition
M-EVENT-REPORT	Confirmed/nonconfirmed	Reports an event about a managed object
(b) Management - Operation Services		
Service	Type	Definition
M-GET	Confirmed	Requests the retrieval of management information
M-SET	Confirmed/nonconfirmed	Requests the modification of management information
M-ACTION	Confirmed/nonconfirmed	Requests that a user perform an action
M-CREATE	Confirmed	Requests that a user create an instance of a managed object
M-DELETE	Confirmed	Requests that a user delete an instance of a managed object
M-CANCEL-GET	Confirmed	Requests that a user cancel a previously requested and currently outstanding invocation of the M-GET service

Figure 2.4: CMIS Services

Managed Object Selection

CMIS provides a powerful set of tools that allow a user to select one object or a number of objects to be subject of a management operation. These facilities are provided as parameters in the M-GET, M-SET, M-ACTION, and M-DELETE service primitives.

Scoping

Scoping refers to the identification of an object or objects to which a filter is to be applied. Scoping is defined with reference to a specific managed object instance, referred to as the *base managed object*. The base managed object is the starting point for the selection of one or more objects to which a filter is to be applied. Recalled from earlier

sections, by the principles of containment and naming, managed objects form a hierarchy, or tree structure. Using the base object as the root, a subtree of the overall object tree is obtained.

Filtering

A filter is a Boolean expression, consisting of one or more assertions about the presence or values of attributes in a scoped managed object. Each assertion may be a test for equality, ordering, presence, or set comparison. Attribute-value assertions can require that the following matching rules be met:

- *Equality:* Attribute value is equal to that asserted.
- *Greater or equal:* Attribute value supplied is greater than or equal to the value of the attribute.
- *Less or equal:* Attribute value supplied is less than or equal to the value of the attribute.
- *Present:* Attribute is present.
- *Substrings:* Attribute value includes the specified substrings in the given order
- *Subset of:* All asserted members are present in the attribute.
- *Superset of:* All members of the attribute are present in the asserted attribute.
- *Non-null-set intersection:* At least one of the asserted members is present in the attribute.

The filter can be a single test, called a filter term, applied to each attribute value to be tested. More complex tests can be made by combining individual filter terms, using AND, OR and NOT.

As example of this notation, consider the following condition to be encoded:

(objectClass **equal to** protocolEntity)
and (entityId **starts with** "123")

and (**severity not Equal to minor**)
or (**badPduCount greater than or equal to 20**))

where **boldface** words are operators and plain-type words are variables.

The following value notation represents this condition:

```
test-filter    CMISFilter ::=
                and { item equality { objectClass, Object-Class protocolEntity },
                    item substrings (initialString { entityID, PrintableString "123" } ),
                    or { not item equality { severity, Severity minor },
                        item lessOrEqual { badPduCount, INTEGER 20 } } }
```

The filter test is applied in the following manner:

1. The filter is applied to every managed object selected by the scope parameter.
2. For each selected object, the filter involves a test of one or more attributes.
3. Assertions about the value of an attribute are evaluated according to the matching rules associated with the attribute syntax.
4. If an attribute value assertion is present in the filter and that attribute is not present in the object, then the result of that test is FALSE.
5. The object or objects for which the filter test evaluates to TRUE are selected for the application of the operation.

Synchronization

The scoping parameter may result in the selection of more than one managed object to be subject to filtering. In turn, if more than one object is scoped, the filtering parameter may result in the selection of more than one object for which the operation is to be performed. The question then arises as to the order in which objects will be processed.

Since the order in which object instances are selected by the filter is not specified but is left as a local implementation matter, this order cannot be used.

However, CMIS does include the concept of synchronization. The CMISE-service user may request one of two types of synchronization:

1. *Atomic*: All managed objects selected for the operation are checked to ascertain whether they are able to successfully perform the operation. If one or more managed objects are not able to do so, then none performs it.
2. *Best effort*: All managed objects selected for the operation are requested to perform it.

2.4.2 Common Management Information Protocol (CMIP)

While the CMIS defines the services for management operations, the common management information protocol (CMIP) defined procedures for the transmission of management information and defines the syntax for the management service of CMIS. CMIP is defined in terms of CMIP protocol data units (PDUs) that are exchanged between peer common management information service elements (CMISEs) to carry out the CMIS service.

As suggested earlier, CMIS provides seven services for performing management operations, in the form of service primitives. For the management operation services, CMISE employs the CMIP to exchange PDUs. The CMIP, in turn relies on the services of ROSE.

2.4.3 Performance

Three competing requirements for a network management system can be stated[1]:

- It should not seriously degrade the network operation it is intended to maintain.
- Its decisions must be made and its control actions taken quickly before network conditions significantly change. Otherwise, instability may arise in the measurement/control-action feedback loop.
- It should provide a rich set of services to handle a wide range of network management function and to provide detailed monitoring and control.

The first requirement implies that network bandwidth and computing resources be conserved. A useful rule of thumb that has been suggested by a number of designers is that the absolute maximum allowable bandwidth consumption by management operations should be 5 percent. On the other hand, the need for timely action implies that a significant amount of management information be exchanged on a regular basis. Furthermore, the need for a feature rich network management service also implies that a lot of management information will be generated.

These issues must be dealt with in any network management system but become particularly acute in the case of OSI systems management. Three interrelated reasons can be cited:

1. OSI systems management offers a broad range of functions and services. If the user takes advantage of these services, the results are sizable implementation at each node and a lot of network traffic.

2. The OSI MIB is large and complex. The individual unit of management is the object, which may include attributes, notifications, and actions. There may be many such objects in a large structure, including objects closely tied to resources, as well as those related to management functions, such as logs, summarization objects, and so forth.
3. CMIP PDUs tend to be large because of the use of BER (basic encoding rules) encoding. Although data encoded using BER are machine-independent, they are usually not as compact as machine-dependent representations. Furthermore, CMIP uses the seven-layer OSI architecture, resulting in substantial blocks of data for individual exchanges.

One technique to reduce network traffic is to streamline the protocol architecture supporting CMIP. However, this approach involves some very complex issues about altering the layer management in the OSI 7-layers model.

Another way to limit CMIP traffic is to be efficient in accessing objects. For example, to access a single object instance, object-class and object instance identifiers must be supplied, each of which may be a long string of integers. Thus, a rather large PDU may be required to access a single small counter (e.g., a 32-bit counter). If many counters must be interrogated, considerable bandwidth would be consumed. However, if the MIB is properly organized, the scoping and filtering facility of CMIS can be employed so that a reference to a single base object results in access to a number of counters simultaneously. This approach works well with newly deployed network because it is always easy to

properly organized the MIB at the very beginning when its size is still relatively small. But for an existing network with a MIB of enormous size, this approach is just not feasible.

A third way for reducing CMIP traffic is to employ a hierarchy of managers. Local managers can be used to manage small LAN segments and convey summary information to more global managers. This reduces overall traffic, since most of the management messages are confined to those network segments where they are needed. A global manager could control local systems by setting parameters in local managers, which then forward these settings to one or more local systems. This method is relatively simpler and it is feasible for both newly deployed or existing networks. Therefore, this is the approach taken in this research project and an implementation of this approach using the OSIMIS platform is presented in chapter 5.

CHAPTER 3

THE OSIMIS MANAGEMENT PLATFORM

3.1 Overview of OSIMIS

OSIMIS is an object-oriented management platform based on the OSI model, implemented mainly in C++[2]. OSIMIS provides an environment for the development of management applications. This environment hides the details of the underlying management service through object-oriented Application Program Interfaces (APIs). Also, it allows designers and implementors to concentrate on the intelligence to be built in management applications rather than the mechanics of management service/protocol access.

The manager-agent model and the notion of managed objects as abstractions of real resources are used, but the separation between managing and managed systems is not strong in engineering terms. A management application can be in both roles; this is particularly true in situations where a management system is decomposed according to a hierarchical logical layered approach.

In fact, OSIMIS was designed from the beginning to support the integration of existing systems with either proprietary management facilities or different management models. Different methods for the interaction with real managed resources are supported, encompassing loosely coupled resources, as is the case with subordinate agents and management hierarchies. The fact that the OSI model was chosen as the basic management model facilitates the integration of other models, the latter usually being less

powerful, as is the case with the Internet SNMP and the emerging OMG CORBA based technologies[13].

OSIMIS already provides a generic application gateway between CMIS and SNMP. Using OSI management as an end-to-end integrating model is also in line with the NMF OMNIPoint[14] approach, which suggests a multiple technology architecture for future heterogeneous environments.

OSIMIS uses the ISODE (ISO Development Environment) [15] as the underlying OSI communications mechanism, but may also be decoupled from it when the current ongoing work on supporting the X/Open XOM/XMP/XDS[16] APIs is finalized. The advantage of the ISODE environment, however, is the provision of services like FTAM and a full implementation of the OSI Directory Service (X.500). These services are essential in complex management environments. Further, a number of underlying network technologies are supported: X.25; CLNP; and also TCP/IP, through the RFC1006 method. These constitute the majority of currently deployed networks, while interoperation of applications across any of these is possible through Transport Service Bridging.

OSIMIS has been, and is still being developed in a number of European research projects, namely the ESPRIT INCA, PROOF, MIDAS, and the RACE NEMESYS and ICM. It has been used extensively in both research and commercial environments. It has served as the management platform for a number of other ESPRIT and RACE projects in the TMN (Telecommunications Management Network) and distributed systems and service management areas.

3.2 The ISO Development Environment

The ISO Development Environment (ISODE) is a platform for the development of OSI services and distributed systems. It provides an upper layer OSI stack that conforms fully to the relevant ISO/CCITT recommendations, including tools for ASN.1 manipulation and remote operations stub generation. Two fundamental applications also come with it: an extensible full Directory Service (X.500); and File Transfer (FTAM) implementations.

Also, ISODE is implemented in the C programming language and runs on most versions of the UNIX operating system. ISODE does not provide any network and lower layer protocols - for example: X.25, CLNP - but relies on implementations for UNIX-based workstations, which are accessible through the kernel interface. The upper layer protocols realized are the transport, session and presentation protocols of the OSI 7-layer model. Application layer Service Elements (ASEs) are also provided as building blocks for higher level services. These services are the Association Control, Remote Operations and Reliable Transfer Service Elements (ACSE, ROSE and RTSE).

The above mentioned services, in conjunction with the ASN.1 support, are used to implement higher level services. A special lightweight presentation layer that runs directly on top of TCP is also provided: this is used for the CMOT (CMIP over TCP) stack. In engineering terms, the ISODE stack is a set of libraries linked with applications using it.

ASN.1 manipulation is very important to OSI distributed applications. The ISODE approach for a programmatic interface (API) relies in a fundamental abstraction known as Presentation Element (PE). This is a generic C structure capable of describing in a

recursive manner any ASN.1 data type. An ASN.1 compiler known as **pepsy** is provided with C language bindings. The compiler produces concrete representations - that is, C structures corresponding to the ASN.1 types - and also encode/decode routines, that convert PEs to a data stream and vice versa according to the encoding rules (e.g. BER). It should be noted that X/Open has defined an API for ASN.1 manipulation known as XOM[16], which though similar in principle to that of ISODE, is syntactically very different.

One of the most important concepts pioneered in ISODE is that of interworking over different lower layer protocol stacks. This interworking is realized through Transport Service bridging (TS-bridge). ISODE provides an implementation of the ISO Transport Protocol (TP) class 0 over X.25 or over the Internet TCP/IP, using the RFD1006 method, in which TCP is treated as a reliable network service. The ISODE session protocol may also run over the ISO TP class 4 and the Connectionless Network Protocol (CLNP). Transport service bridges, which are simple relaying applications similar to Interworking Units, may be used to link subnetworks of all these different communities. Transport service bridges also provide end-to-end interoperability hiding the heterogeneity of the underlying network technology. The combinations mentioned constitute the vast majority of currently deployed networks.

3.3 Management Protocol and High-Level Abstract Syntax Support

OSIMIS is based on the OSI management model as the means for end-to-end management. As such, it implements the OSI Common Management Information

Service/Protocol (CMIS/P). This is implemented as a C library and uses the **ISODE ACSE**, and **ROSE** and its **ASN.1** support. For every request and response, **CMIS** primitive is realized through a procedure call. Indications and confirmations are realized through a single “wait” procedure call. Associations are represented as communication endpoints (file descriptors) and operating system calls. For example, the **Berkeley UNIX select(2)** can be used for multiplexing them to realize event driven policies.

Regarding **ASN.1** manipulation, it is up to an application to encode and decode the above values, as this adds to its dynamic nature by allowing late bindings of types to values and graceful handling of error conditions. **OSIMIS** provides mechanism to support high-level objected-oriented **ASN.1** manipulation, shielding the programmer from details, and enabling distributed programming using simply **C++** objects as data types. This mechanism is achieved by using polymorphism, which encapsulates behaviour in the data types, determining how encoding and decoding should be performed through an **ASN.1** meta-compiler which produces **C++** classes for each type. Encode, decode, parse, print and compare methods are produced together with a get-next-element one for multi-valued types (**ASN.1 SET OF** or **SEQUENCE OF**). Finally, the very important **ANY DEFINED BY** construct is automatically supported through a table driven approach, which maps types to syntaxes. This high-level O-O **ASN.1** approach is used by higher level **OSIMIS** services, such as **GMS** (Generic Managed System), **RMIB** (Remote MIB) and **SMIB** (Shadow MIB) , all of which are described in the subsequent sections.

3.4 Application Coordination Support

Management and, more generally, distributed applications have complex needs in terms of handling external input. Management applications have additional needs for internal alarm mechanisms for arranging periodic tasks in real time, (polling, etc.) Furthermore, some applications may need to be integrated with Graphical User Interface (GUI) technologies, which have their own mechanisms for handling data from the keyboard and mouse. In this context, the term "application" implies one process in operating systems terms.

There are in general different techniques to organise an application for handling both external and internal events. The organization needs to be event driven so that no resources are used when the system is idle. The two major techniques are[2]:

- i) the use of a single-threaded execution paradigm
- ii) the use of a multi-threaded one

In the first technique, external communications should follow an asynchronous model, as waiting for a result of a remote operation in a synchronous fashion will block the whole system. Of course, a common mechanism is needed for all the external listening and demultiplexing of the incoming data; this is part of what the OSIMIS Application Coordination Support provides. In the second, many threads of control can be executing simultaneously (in a pseudo-parallel fashion) within the same process. This means that blocking on an external result is allowed. This is the style of organization distributed systems platforms as they are based on RPC (Remote Procedure Call), which is inherently synchronous.

The advantage of the first mechanism is as follows: it is supported by most operating systems and, as such, is lightweight and efficient. Its drawbacks: it introduces state for handling asynchronous remote operation results. The second mechanism allows more natural programming in a stateless fashion with respect to remote operations, but it requires internal locking mechanisms and re-entrant code. In addition, such mechanisms are not yet commonly supported by operating systems and, as such, are not very efficient. An additional problem in organising a complex application concerns the handling of internal timer alarms: most operating systems do not “stack” them; that is, there can only be one alarm pending for each process. This means that a common mechanism is needed to ensure the correct usage of the underlying one.

OSIMIS provides an object-oriented infrastructure in C++ which allows one to organize an application in a fully event-driven fashion and a single-threaded execution paradigm, where every external or internal event is serialised and taken to completion on a “first-come-first-serve” basis. This mechanism allows the easy integration of additional external sources of input or timer alarms: it is realised by two C++ classes, the Coordinator, and the Knowledge Source (KS).

There should always be one instance of the Coordinator, or any derived class in the application; the existence of it is totally transparent to application implementors, except at the initialisation time. Its supports can be accessed through KS-derived classes. The KS is an abstraction of a general application object with the potential of a) receiving data asynchronously in external communication endpoints, and b) being woken up periodically in various real time tasks. The term has its roots in AI blackboard systems and, generally,

KSs are responsible for implementing the intelligence of an application with respect to external communications and periodic real-time activities. This model is depicted in Figure 3.1.

The coordination mechanism is designed in such a way as to allow integration with those of other systems. This integration is accomplished through special coordinator derived classes which will interwork with a particular mechanism. This interworking is achieved by still controlling the sources of input and timer alarms of the OSIMIS KSs but, instead of performing the central listening, the input sources and the alarms are passed to the other system's coordination mechanism. This interworking is needed for OSIMIS agents which wish to receive association requests, since ISODE imposes its own listening mechanism. This listening mechanism hides the Presentation Service Access Point (PSAP) on which new ACSE associations are accepted. A similar mechanism is needed for Graphical User Interface technologies, which have their own coordination mechanisms. In this case, simply a new special coordinator class is needed for each of them. At the moment, the X-Windows Motif and the InterViews graphical object library are integrated.

3.4.1 The Mechanism of the Coordinator

Since the Coordinator is an essential part of the OSIMIS infrastructure providing coordination for every internal and external event, the actual program codes of the Coordinator class have been analyzed, in order to understand what mechanism the Coordinator uses and how it works. The results are presented in the following sections.

Figure 3.1: The OSIMIS Process Coordination Model

3.4.1.1 The Central Listening Loop

The Coordinator uses a fully event-driven scheme with respect to all external communications and scheduled real-time alarms through a first-come-first-served policy. A central listening loop is adopted in order to execute this policy.

```

void    Coordinator::listen ( )
{
    // set signal handlers
    setAlarmSignalHandler( );
    setTerminateSignalHandler( );

    // start the listening loop
    for (;;)
        readCommEndpoints( );
}

```

First, `setAlarmSignalHandler( )` is used for defining which method to call when the signal `SIGALRM` occurs. This is a signal sent off by the UNIX system alarm clock, which

OSIMIS uses to schedule the wake-ups of the managed objects (MOs). In this case, the method `serviceWakeUp( )` of the Coordinator class is called to handle the awakened MO.

Similarly, `setTerminateSignalHandler( )` is used for defining which method to call when one of the terminating signals (SIGINT, SIGQUIT, SIGTERM) occurs. These signals may be a result of program error or user request. The method `terminate( )` of the Coordinator class is called for handling the termination.

At that point, the listening loop starts. The method `readCommEndpoints( )` is responsible for listening to all the sockets registered by MOs exercising an event-driven scheme. When an event occurs (or data is available) at a particular socket, the method will notify the corresponding MO to deal with the event.

However, if the signal SIGALRM occurs, the loop is interrupted to give way for calling the method `serviceWakeUp( )`, as mentioned earlier. When the method finishes its service, the loop will resume its process at the point at which it was interrupted.

In this way, the Coordinator can simultaneously monitor the activities of the two types of services (i.e.: real-time alarms and external communications) and is able to serve them on a first-come-first-served basis.

3.4.1.2 The Coordinator Support

Basically, the Coordinator supports two types of MOs, and each of them is discussed with their built-in supporting methods in this section.

i) for MOs exercising a polling scheme:

```
int    scheduleWakeUps( );
int    cancelWakeUps( );
int    serviceWakeUp( );
```

The Coordinator maintains a linked list of all the scheduled alarms. When a MO wants to be woken up at every certain period, all it has to do is to call `scheduleWakeUp( )`. This method then inserts its request into the linked list. When it is time to wake up this MO, a signal `SIGALRM` is emitted by the UNIX system alarm clock and `serviceWakeUp( )` is called.

Since every MO is derived from a KS in OSIMIS, the method `serviceWakeUp( )` notifies the KS of the awaking MO. It can then start to update all of its attributes. After that, the MO can either wait for the next wake up or it can stop the process by calling `cancelWakeUp( )`. This method will delete its request from the linked list and the MO will not be woken up again.

ii) for MOs exercising a event-driven scheme:

```
int    registerCommEndpoint( );
int    deregisterCommEndpoint( );
int    readCommEndpoints( );
```

The Coordinator uses `registerCommEndpoint( )` to register a port that the corresponding MO wants to get data from. Then using `readCommEndpoints( )`, the Coordinator listens on it and notifies the MO when data is available. This process is called "call-backs" in OSIMIS terms.

In Berkeley-based UNIX systems, listening on a port implies listening on a socket. Thus, what the `registerCommEndpoint( )` actually does, is add the file descriptor (fd) of the socket into the `FD_SET`, which is the set of fds to which `readCommEndpoints( )` listens. Conversely, `deregisterCommEndpoint( )` is used for deleting the fd of a socket in the `FD_SET`.

3.5 The Generic Managed System (GMS)

The Generic Managed System (GMS) provides support for building agents that offer the full functionality of the OSI management model[2], including scoping, filtering, linked replies and cancel-get. OSIMIS fully supports the Object Management, Event Reporting and Log Control Systems Management Functions (SMFs). It also supports the QualityOfServiceAlarm notification of the Alarm Reporting Function and, in part, the Access Control, Metric and Summarization object. In conjunction with the GDMO compiler, it offers a very high level API for the integration of new managed object classes, where only semantic aspects need to be implemented. It also offers different methods of access to the associated real resources, including proxy mechanisms based on the Coordination mechanism.

The GMS is built using the coordination and high level ASN.1 support infrastructure. Most of its facilities are provided by the following three C++ classes, and more details of them are presented in the next section:

- i) the MO, which is the abstract class providing generic managed object support;
- ii) the MOClassInfo, which is a meta-class for a managed object class; and
- iii) the CMIS Agent, which provides OSI agent facilities.

The GMS library also contains generic attribute types such as counter, gauge, counterThreshold, gaugeThreshold and tideMark, as well as specific attributes and objects as in the Definition of Management Information (DMI) relating the SMFs[2].

3.5.1 Managing Real Resources

The managed real resources may be either physical or logical. They may reside in an operating systems kernel, in communications boards, or even in remote systems, in a proxy fashion. As mentioned previously, management of real resources is achieved through abstractions of them known as Managed Objects (MOs).

A two-way communications is needed between MOs and the resources they encapsulate for monitoring, controlling and asynchronous reporting of events. The MO should always provide an up-to-date and consistent view of the corresponding resource through the management interface[8]. In fact, there are three types of methods for accessing the real resources:

- i) access upon external request;
- ii) "cache-ahead" through periodic polling; and
- iii) update through asynchronous reports.

The first method signifies that the values are fetched only if requested through the management protocol. In the second, requests are responded to quickly, especially with respect to loosely coupled resources, but timeliness of information may be slightly affected. Finally, the third method is good only if it can be tailored so that there is no unnecessary overhead when the agent is idle.

The GMS offers support for all of these methods through the coordination mechanism. When asynchronous reports from a resource or asynchronous results to requests are expected, it is likely that a separate object will be needed to demultiplex the incoming information and deliver it to the appropriate MO. These objects are usually

referred to as Internal Communications Controllers (ICC) which are essentially specialised knowledge sources[2].

3.5.2 Managed Objects

Managed objects are represented internally as language objects that contain data (attributes) and offer a well-defined method interface. In every managed object, there is both instance and class information. Instance information is its data, that is, its attributes and any other instance specific information. The class information is common to all instances of a class; therefore, it is internally represented as a separate object instance[8].

The class information is related to the attributes, actions and notifications for the class, as well as the initial attribute values, "template" ASN.1 objects for manipulating notifications and integer tags associated to the object identifiers. This leads to the introduction of a common meta-class for all the managed object classes - the `MOClassInfo`. The inheritance tree is internally represented by instances of this class linked in a tree-like fashion, as shown in Figure 3.3.

Managed object classes are simply realized by equivalent C++ classes produced by the GDMO compiler, and augmented manually with behaviour. Through access to meta-class information, requests are first checked for correctness and authorization before the behaviour code that interacts with the real resources is invoked.

Also, managed object instances are linked internally in a tree mirroring the containment relationships. Scoping is relegated to a tree search while special care is taken to make sure the tree reflects reality when accessed. Filtering is provided by comparing

methods of the attributes - which are simply the C++ syntax objects or derived classes when behaviour is coded at the attribute level.

Basically, the codes of a managed object consist of three parts:

- i) the generic codes that enable object addressing, attribute addressing, scoping and filtering;
- ii) the codes specific to the managed object classes that constitute the instance; these codes may be produced by a GDMO compiler; and
- iii) the codes specific to the real functionality of the object. i.e.: the codes that interact with the corresponding real resource or the rest of the MIB for control objects.

Furthermore, an object instance will contain codes for at least two separate object classes. These are the Top and the specific class realized. These codes have access to the class object and instantiate the attributes of the basic and any conditional packages.

Every managed object instance has some codes specifically for accessing the real resource. There is an internal Application Program Interface (API), through which these hand-written specific codes are invoked. The GMS provides a library of procedures implementing all the requests and responses for performing management service API, such as Get, Set and etc. An example of API realizing a CMIS Get request is given in the following:

```
int      M_Get (msd, invoke, objClass, objInst, scope, filter, access, sync, nattrs, attrs, mi )
```

```
int      msd, invoke;
MIdentifier*  objClass;
MName*      objInst;
CMISScope*  scope;
CMISFilter* filter;
External*   access;
CMISSync    sync;
int         nattrs;
MIdentifier attrs[];
MSAPIndication* mi;
```

Moreover, these are simple APIs, since all the possible checks have already been done by the generic code. In particular, all the attributes, actions and notifications are referred to through integer tags, instead of encoded ASN.1 values. This can make the amount of codes, that need to be written by a MO implementor, to be absolutely minimized. The overall structure of an object instance is shown in Figure 3.2.

Figure 3.2: The overall structure of a managed object instance

3.5.3 An Agent Application

Every agent application needs an object that understands the management protocol, exercises access control, passes requests to the managed objects and returns replies[8]. In GMS, this object is implemented by a specialized knowledge source called the CMIS Agent. It handles association establishment and release, receives CMIS

3.5.4 System Management Functions

As already stated, OSIMIS supports the most important of the systems management functions. As far as GMS is concerned, these functions are realised as special managed objects, generic attribute and notification types which can simply be instantiated or invoked. This is the case, for example, with the alarm reporting, metric and summarization objects. In other cases, the GMS knows the semantics of these classes and uses them accordingly (as in event and log control). Notifications can be emitted through a special method call: all the subsequent notification processing is carried out by the GMS in a transparent fashion to application codes.

3.6 Generic High-Level Manager Support

Programming manager applications using the CMIS API can be tedious. Higher object-oriented abstractions can be built on top of the CMIS services. OSIMIS provides two types of service for these abstractions. They are the Remote MIB and the Shadow MIB support services[2].

The Remote MIB

The Remote MIB (RMIB) support service offers a higher API, which provides the abstraction of an association object. This handles association establishment and release, hides object identifiers through friendly names, and hides ASN.1 manipulation using the high-level ASN.1 support. It also hides the complexity of CMIS distinguished names and filters through a string-based notation. Further, it assembles linked replies and provides a high-level interface to event reporting which hides the manipulation of event

discriminators. Finally, it provides error handling at different levels. There is also a lower level interface for applications that do not want this friendliness and the performance cost it entails, but these applications still need the high-level mechanisms for event reporting and linked replies.

In the RMIB API, there are two basic C++ classes involved; the **RMIBAgent**, which is essentially the association object (a specialized KS in OSIMIS terms); and the **RMIBManager** abstract class, which provides call-backs (described in section 3.4.1.2), for asynchronous services offered by the **RMIBAgent**.

While the RMIB infrastructure offers a much higher level facility than a raw CMIS API such as the OSIMIS MSAP (Management Service Access Point), its nature is closely linked to that of CMIS (apart from the fact that it hides the manipulation of event forwarding discriminators to effect event reporting). Although this facility is perfectly adequate for even complex managing applications, as it offers the full CMIS power (scoping, filtering, etc.), simpler higher-level approaches could be very useful for rapid prototyping. One such facility is provided by the Shadow MIB support service.

The Shadow MIB

The Shadow MIB (SMIB) support service offers the abstraction of objects in local address space, "shadowing" the real managed object handled by remote agents. The real advantages of such an approach are twofold. First, the API can be less CMIS-like for accessing the local objects, since parameters such as distinguished names can simply be replaced by pointers in local address space. Second, the existence of images of MOs as local shadow objects can be used to cache information and optimize access to the remote

agents. The caching mechanism could be controlled by local application objects, tailoring it according to the nature of the application in hand, in conjunction with shared management knowledge regarding the nature of the remote MIBs. The model and supporting C++ classes are very similar to the RMIB's. The two models are illustrated in Figure 3.4.

3.7 OSIMIS Applications

OSIMIS offers a set of generic manager applications, graphical or command-line based, for providing the full power of CMIS and a generic application gateway between CMIS/P and the Internet SNMP.

Figure 3.4: The Remote and Shadow MIB Support Models

3.7.1 Generic Managers

There is a class of applications which are usually referred to as MIB browsers. The reason is that they allow one to move around in a management information tree, retrieve and alter attribute values, as well as perform actions, create and delete managed objects. OSIMIS provides a MIB browser with a Graphical User Interface based on the InterViews X-Window's C++ graphical object library. OSIMIS also provides a set of programs (e.g. *mibdump*, *mcreate*, and *mset*) that operate from the command line and realise the full set of CMIS operations. There is also an event sink application that can be used to receive event reports according to specified criteria.

3.7.2 The Generic CMIS/SNMP Application Gateway

The current industry standard for network element management is the Internet SNMP, which is a simplified version of the OSI CMIP. Generic application gateways between them are possible without any semantic loss, because the operations and the information model of SNMP is a pure subset of those of OSI. The OSIMIS generic application gateway acts as a proxy agent for converting management operation services between an OSI manager and a SNMP agent.

This work involves a translator between Internet MIBs to equivalent GDMO's, and a special back-end for the GDMO compiler which produces run-time support for the generic gateway. That way, the handling of any current or future MIBs will be possible without the need to change a single line of code. It should be added that the generic

gateway of OSIMIS, called IQA (Internet Q-Adapter), only works with SNMP version 1 at the present time.

3.7.3 Generally Useful Agents

There are two useful agents implementing specific MIBs which come with OSIMIS. The first agent application, which is named SMA (System Management Agent), contains a non-standard implementation of the MIB for OSI Transport Protocol (TP). Also, it contains another non-standard UNIX MIB, with just one managed object showing the number of users in a UNIX system with an associated threshold and tide-mark, whose only purpose is to serve as an example of the interface to the generic parts of the system.

The second, which is named OIM-SMA (OSI Internet MIB agent), implements a native OSI version of TCP/IP SNMP MIB-II, as in the RFC1214 (without the EGP and SNMP groups). This can be used to find out the utilisation of interfaces, manage addresses and routes in workstations, etc.

CHAPTER 4

DEMONSTRATIONS OF THE OSIMIS APPLICATIONS

4.1 Overview of the Demonstrations

This chapter consists of four application demonstrations which illustrate the capabilities of OSIMIS for realizing OSI network management. One should note that all the demonstrations presented in this thesis are implemented using the OSIMIS version 4.0. Other versions may not have all the required applications, or they may have different command syntaxes for the applications.

Section 4.2 shows an OSIMIS user or a MO implementor, how to realize an Agent-Manager model with an OSI agent using the OSIMIS GMS. Section 4.3 describes the method of implementing the OSI event-reports-forwarding management function, using another OSIMIS facility, the *evsink*. Also, section 4.4 shows the usage of the OSIMIS Internet Q-Adapter (IQA) for realizing the Agent-Manager Model with a SNMP agent. Finally, section 4.5 presents two methods for providing thresholding capabilities to the existing MOs in OSIMIS.

4.2 Realizing the Agent-Manager Model of Network Management with an OSI Agent

The key task for an MO implementor is to derive a C++ class to represent the new MO which he wants to put in the MIB[3]. The starting point will be a definition of the MO

class - typically in GDMO format in some standard. This will include other things like the identity of its parent MO classes in the inheritance hierarchy, and a list of additional attributes which the new MO class is to contain.

This pattern will be reflected in the C++ class, a new class derived from a parent class, with the additional attributes being added. In many cases, these additional attributes will be drawn from the standard types provided by the GMS (gauge, counter, etc.). If this is not the case, a new C++ class must be implemented to represent the new attribute type.

In practice, most of the code written by a MO implementor will only concern interfacing the MO to the real resource. The other MO interface - the one to the agent itself - is already provided by the GMS through its generic MO class. Moreover, an OSI agent named SMA (System Management Agent) is also provided in OSIMIS. If the MO implementor wants the agent to manage a particular real resource of his own, all he needs to do is to define a MO class, by either deriving it from an existing class or building it from the ground up. Then he adds it to the OSIMIS MIT. The next section shows the method of adding a new MO class in OSIMIS by going through an example that I have implemented.

4.2.1 Adding a New MO class to the OSIMIS Environment

Before showing the steps for building a new MO class, there are some directories on the UNIX system that are going to be referred to in the subsequent sections:

<code>LIB_DIR</code>	<code>=</code>	<code>/usr/local/lib/osimis</code>	
<code>INCLUDE_DIR</code>	<code>=</code>	<code>/usr/local/include/osimis</code>	
<code>OSIMIS_HOME_DIR</code>	<code>=</code>	<code>/home/carlos/osimis</code>	(in my case)

A list of all the steps needed to build a new MO class is shown below.

1. Define the required attributes of the new MO class.
2. Define the new MO class with all the required information in GDMO format.
3. Compile the GDMO file of the MO class.
4. Write the codes for interfacing the MO to the real resources.
5. Compile everything into a library file.
6. Update the OSIMIS oidtable files.
7. Update the initialization file of the agent.
8. Update the meta-class tree of the agent.
9. Update the “Makefile” of the agent to include the library file of the new MO class.
10. Recompile the agent to include the new MO class.

By using these steps, I have implemented the following example for creating and adding a new MO class named “mailObj” for counting the number of emails in my mailbox. Details of each of the steps for implementing this example are presented below:

Step 1) Define the required attributes:

In this case, the mailObj MO class must have at least the following two attributes:

- *mailObjId* - the string-type attribute used to distinguish one instance from another of this MO class.
- *nMails* - the gauge-type attribute for reflecting the current number of emails.

Step 2) Define the MO class in GDMO format:

```
-- ##### Class Templates #####

mailObj MANAGED OBJECT class
  DERIVED FROM    top;
  CHARACTERIZED BY mailObjPackage;
  REGISTERED AS    { uclManagedObjectClass 53 };

-- ##### Name Binding Templates #####

mailObj-system          NAME BINDING
  SUBORDINATE OBJECT class mailObj AND SUBclassES,
  NAMED BY
  SUPERIOR OBJECT class  system AND SUBclassES,
  WITH ATTRIBUTE          mailObjId,
  CREATE                  ;
  DELETE                  ONLY-IF-NO-CONTAINED-OBJECTS;
  REGISTERED AS           { uclNameBinding 53 },

-- ##### Package Templates #####

mailObjPackage PACKAGE
  ATTRIBUTES
    mailObjId      GET
    nMails          GET,

  NOTIFICATIONS
    objectCreation,
    objectDeletion,
    attributeValueChange,

  REGISTERED AS    { uclPackage 53 },

-- ##### Attribute Templates #####

mailObjId ATTRIBUTE
  WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module SimpleNameType,
  MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING,
  REGISTERED AS    { uclAttributeID 531 },

nMails ATTRIBUTE
  DERIVED FROM gauge,
  REGISTERED AS    { uclAttributeID 532 },

-- ##### Notification Templates #####

objectCreation NOTIFICATION
  BEHAVIOUR      objectCreationBehaviour,
  WITH INFORMATION SYNTAX Notification-ASN1Module ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attributeList         attributeList;
  REGISTERED AS          { smi2Notification 6 },

objectDeletion NOTIFICATION
  BEHAVIOUR      objectDeletionBehaviour;
  WITH INFORMATION SYNTAX Notification-ASN1Module ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attributeList         attributeList;
  REGISTERED AS          { smi2Notification 7 },

attributeValueChange NOTIFICATION
  BEHAVIOUR      attributeValueChangeBehaviour,
  WITH INFORMATION SYNTAX Notification-ASN1Module AttrValChangeInfo
```

```

AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attrValChangeDefinition attrValChangeDefinition,
REGISTERED AS          ( smi2Notification 1 ),

qualityofServiceAlarm NOTIFICATION
BEHAVIOUR              qualityofServiceAlarmBehaviour,
WITH INFORMATION SYNTAX Notification-ASN1Module.AlarmInfo
AND ATTRIBUTE IDS
    probableCause      probableCause,
    perceivedSeverity   perceivedSeverity,
    thresholdInfo      thresholdInfo,
REGISTERED AS          ( smi2Notification 11 );

```

Step 3) Compile the GDMO file of the MO class just by typing in:

```
gdmo mailMIB.gdmo
```

As a result of the compilation, four files are obtained:

mailObj.cc, mailObj.h	---	the C++ files for interfacing the MO to the agent.
Makefile.mo	---	the Makefile for compiling the C++ files of the MO class to a library file
oidtable.tmp	---	a file used to append to the existing OSIMIS oidtable files to let the agent know about the existence of the new MO class

Step 4) Create two C++ files, "mailObj.inc.h" and "mailObj.inc.cc" for interfacing the MO

to the real resources. In this case, the contents of the two files are as follows:

For "mailObj.inc.h",

```

protected:
    char* updateNMAils ();
    int   get (int, int, AVA*&, Bool = False, int = -1),
public
    int   createRR (AVA*&, void* = NULLVD, int = -1).

```

For "mailObj.inc.cc,"

```

/* For creating a MO instance of the MO class mailObj */
int   mailObj::createRR (AVA*& err, void*, int) { ..... }

/* For getting the meta-class information of the MO class mailObj */
int   mailObj::get (int attrId, int classLevel, AVA*& err, Bool checkOnly, int) { ... }

/* For updating the number of emails in the mailbox using the UNIX command "messages" */
char   *mailObj::updateNMAils { ... ..... }

```

the first two procedures can be accomplished by modifying the same ones from other existing MO classes. The third one is the only procedure that needs to be hand-written from scratch for interfacing the mailObj MO to the real resource - in this case, the mailbox.

After finishing the codes in these two files, copy them to the INCLUDE_DIR.

Step 5) Compile the mailObj C++ files into a library file:

- i) Modify the LIB argument inside the file "Makefile.mo". In this case, the argument is changed as "libmailmib.a".
- ii) Rename the file "Makefile.mo" to "Makefile".
- iii) Compile the mailObj files by typing in:

```
make lib
```

Then, a library file "libmailmib.a", for the new MO class mailObj, is obtained.

Step 6) Update the OSIMIS oidtable files:

- i) Go to the LIB_DIR.
- ii) Update the files "oidtable.at" and "oidtable.gen" using the "oidtable.tmp" file obtained in step 4. The file "oidtable.at" contains all the information about the attributes and notifications of the MOs, while the "oidtable.gen" file concerns itself with the name bindings of the MOs.

Step 7) Update the initialization file of the agent:

- i) Go to the LIB_DIR.
- ii) Add a new entry to the file "mib.init.sma", such as

```
mailObj      mailObjId=carlos
```

Thus, every time the agent SMA runs, a mailObj MO instance with an ID “carlos” is automatically created.

Step 8) Update the meta-class tree of the agent:

- i) Go to the directory OSIMIS_HOME_DIR/agent/sma/.
- ii) Add new entries to the file “Sma.h”.

```

        include "Create.h"
        .....
        include "uxObj1.h"
ADD      include "mailObj.h"
        .....
        MOClass moclasses[] =
        .....
        {"uxObj1",
         uxObj1::getClassInfo,
         uxObj1::initialiseClass },
ADD    {"mailObj",
         mailObj::getClassInfo,
         mailObj::initialiseClass },
        .....

```

Step 9) Modify the “Makefile” of SMA to include the library file (“libmailmib.a”) of the new mailObj MO class:

- i) Suppose the new library file is located at OSIMIS_HOME_DIR/agent/mail_mib/.
- ii) Add a new entry to the MIBLIBS argument in the “Makefile” as follows:

```

MIBLIBS      =      ../isode_mib/libisode.a ..... \
                    ...../ ...../ ../mail_mib/libmailmib.a

```

Step 10) Recompile the file “sma” (the OSI agent) so that the new MO class will be included:

Just type in: make sma

Then, a new executable file “sma” is obtained.

4.2.2 Manipulating the MO Instances

Before running the OSI agent SMA on a station and starting to manipulate the MO instances, an entry for the desired station on which the agent is going to execute has to be added in the file "isoentities", in the directory OSIMIS_HOME_DIR/etc/. The entry in my case is as follows:

```
artie    SMA    1.17.5.12.0    #603/Internet=134.117.4.63+11010
```

Also, an environmental variable named OSIMISETCPATH has to be set to the proper directory before the SMA can execute successfully. My entry for this variable is:

```
OSIMISETCPATH = /home/osimis/carlos/etc
```

Now, the OSI agent SMA can be executed by just typing "sma" at the system prompt, and the screen display should look like:

```
taylor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
application entity title: SMA-artie

initialising meta-class information...
top
system
discriminator
.
.
mailObj
.
.
entity
monPort
connection
transportEntity
transportConnection
subsystem

setting up notification to event record mappings...
objectCreation          eventLogRecord
objectDeletion          eventLogRecord
attributeValueChange    eventLogRecord
stateChange             eventLogRecord
qualityofServiceAlarm   eventLogRecord
initialising MIB...
restarting logs
system {systemId=artie}
```

```

subsystem {subsystemId=4}
transportEntity {subsystemId=4@entityId=isode}
mailObj::update nMails 53
mailObj {mailObjId=carlos}
listening...

```

After the SMA is running at the agent station, the human manager can manipulate the MO instances, using the OSIMIS generic manager applications. In this section, all the examples are done in a simple network management system, as shown in Figure 4.1.

Figure 4.1: A Simple Network Management System

Here are some of the examples:

- Showing the syntax of *mibdump*, one of the generic manager applications used for retrieving information of MO instances from an agent.

```

> mibdump
usage: mibdump { <agent> <host> | -S <agent> }
        [-c <class>] [-i <instance>]
        [-s <scope> [<sync>]] [-f filter] [-a attr ...] ]

```

- Retrieving the information of the mailObj MO created in the previous section:

```

> mibdump SMA artie -c mailObj -i systemId=artie@mailObjId=carlos

```

```

tailor : /home/pegasus/carlos/osimis/proxy/ica/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/ica/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52

```

Get Result - id 0x000001, class mailObj, instance {mailObjId=carlos}, time
 19960703223603.000140--400
 objectClass: mailObj
 nameBinding: mailObj-system
 mailObjId: carlos
 nMails: 53

1 replies received

- Retrieving the information of all the MOs in the first level of the OSIMIS MIT using the scoping capability:

> mibdump SMA artie -s 1stlevel

taylor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Linked Reply Get - id 0x000001, class subsystem, instance {subsystemId=4}, time
 19960703223649.501121--400
 objectClass: subsystem
 nameBinding: subsystem-system
 subsystemId: 4

Linked Reply Get - id 0x000001, class uxObj1, instance {uxObjId=test}, time
 19960703223649.895270--400
 objectClass: uxObj1
 nameBinding: uxObj1-system
 uxObjId: test
 sysTime: 960703223649Z
 wiseSaying: Hello world
 nUsers: 1

Linked Reply Get - id 0x000001, class mailObj, instance {mailObjId=carlos}, time
 19960703223649.698967--400
 objectClass: mailObj
 nameBinding: mailObj-system
 mailObjId: carlos
 nMails: 53

3 replies received

- Retrieving only the MO instance with its objectClass as "mailObj" on the first level using the scoping and the filtering capabilities:

> mibdump SMA artie -s 1stlevel -f "(objectClass= mailObj)"

taylor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Get Result - id 0x000001, class mailObj, instance {mailObjId=carlos}, time
 19960703223755.905111--400
objectClass: mailObj
 nameBinding: mailObj-system
 mailObjId: carlos
 nMails: 53

1 replies received

- Retrieving only the attribute "nMails" of the mailObj MO instance:

> mibdump SMA artie -s 1stlevel -f "(objectClass=mailObj)" -a nMails

tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Get Result - id 0x000001, class mailObj, instance {mailObjId=carlos}, time
 19960703223916.453019--400
nMails: 53

1 replies received

- Creating a new MO instance of the mailObj MO class with the ID "John":

> mcreate SMA artie -c mailObj -i systemId=artie@mailObjId="John"

tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Create Result - id 0x000001, class mailObj instance {mailObjId=John} time
 19960703224032.746580--400
objectClass: mailObj
 nameBinding: mailObj-system
mailObjId: John
 nMails: 53

Then, both mailObj MOs can be shown by retrieving all the MOs again in
 the first level:

> mibdump SMA artie -s 1stlevel

tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Linked Reply Get - id 0x000001, class subsystem, instance {subsystemId=4}, time
 19960703224052.922886--400
objectClass: subsystem
 nameBinding: subsystem-system
 subsystemId: 4

Linked Reply Get - id 0x000001, class uxObj1, instance {uxObjId=test}, time
 19960703224053.121749--400
 objectClass: uxObj1
 nameBinding: uxObj1-system
 uxObjId: test
 sysTime: 960703224053Z
 wiseSaying: Hello world
 nUsers: 1

Linked Reply Get - id 0x000001, class mailObj, instance {mailObjId=carlos}, time
 19960703224053.017874--400
 objectClass: mailObj
 nameBinding: mailObj-system
mailObjId: carlos
 nMails: 53

Linked Reply Get - id 0x000001, class mailObj, instance {mailObjId=John}, time
 19960703224053.255527--400
 objectClass: mailObj
 nameBinding: mailObj-system
mailObjId: John
 nMails: 53

4 replies received

- Deleting the mailObj MO with the ID "John" :

> mdelete SMA artie -c mailObj -i systemId=artie@mailObjId= John

tailor : /home/pegasus/carlos/osimis/proxy/qa/etc/test/osimistailor
 oidtable : /home/pegasus/carlos/osimis/proxy/qa/etc/test/oidtable
 OIDs 374 : attributes 490 : classes 52

Delete Result - id 0x000001, class mailObj instance {**mailObjId=John**} time
 19960703224204.365732--400

1 managed objects deleted

4.2.3 Modifying an existing MO class

The purposes for modifying an existing MO class can be divided into two categories:

1. To alter the information in the GDMO file of a MO class, such as varying the data type of an attribute or adding a new type of notification, etc.

2. To alter the interfaces of a MO class, either the one between the MO class and the agent, or the one between the MO class and the real resources. The latter interface is the usual alternation.

For modifications in the first category, the MO implementor will have to repeat all ten steps described in section 4.2.1. However, for the second category, only the four steps shown below are needed:

- i) Modify the C++ (.inc.h and .inc.cc) files of the MO class.
- ii) Recompile the library file of the MO class.
- iii) Copy the modified h files to the INCLUDE_DIR.
- iv) Recompile the OSI Agent SMA..

4.3 Realizing Event Report Forwarding Between Management Entities

4.3.1 The OSI Event-Report-Management Function

The OSI event-report-management function is responsible for providing the capability of event report forwarding between management entities. The event-report-management function enables the user to control the selection of events and their distribution to manager-specifiable destinations[1]. It enables the user to create an EFD (Event Forwarding Discriminator) control object. This object defines manager-creatable/selectable criteria by which managed-object notification may be conveyed remotely as event reports, as well as time periods, during which such event-forwarding discrimination can occur. Finally, the function enables the user to suspend and resume event reporting on a per-destination basis.

Figure 4.2 illustrates the event-report-management model, which describes the conceptual components that provide for remote event reporting and local processing of potential events. It also describes the message flows for the control messages, event-reporting messages, and retrieval messages.

Figure 4.2: Event-Report-Management Model

The behaviour of the model[1] is as follows:

1. At least one managed object in the managed system is capable of generating notifications that may be forwarded within an event report. Notifications are issued by MO when events occur.
2. An event-detection and processing module receives the locally generated notifications and forms potential event reports. Then the reports are distributed to all EFDs within the local system. The EFDs determine which event reports are to be forwarded to a particular destination as real event reports. The EFDs also

contain a discriminator construct (or a filter) that specifies the characteristics which a potential event report must satisfy in order to be eligible for forwarding.

3. The EFDs are MOs, and like other MOs, have states and attributes. Thus, they can be initiated, terminated, suspended and resumed through the use of management operations.
4. Also, like other MOs, EFD objects may emit notifications. These notifications are processed as potential event reports by all EFDs, including the EFD that generated the notification.

4.3.2 Introduction to an OSIMIS Generic Manager Application - Evsink

The *evsink* is a generic manager application provided by OSIMIS to realize the event-report-management model. The *evsink* enables one to connect to a remote agent application, and to request or receive event reports. It first establishes a management association, creating an EFD management control object at the agent, and requests the event reports as specified through the command line arguments.

An agent can simultaneously have several EFD objects which are created by *evsinks* from different management entities. Each EFD control object selects the desirable event reports requested; these are sent to the corresponding *evsink*. Then the desirable event reports for different management entities are obtained and are displayed on the screen.

When the *evsink* receives a termination signal (due to user request or program error), it deletes the EFD object it has created at the agent and releases the management association before it exits.

4.3.3 The Syntax of Evsink

The synopsis of *evsink* is as follows[3]:

```
evsink { <agent> <host> | -S <agent> } -f [ <filter> | <eventType> ... ]
```

The Command Line Arguments

The <agent> argument is mandatory and expresses the remote logical management application "qualifier". In OSIMIS, there are two such application qualifiers: SMA for the ISO Transport and the example UNIX MIB agent; and OIM-SMA for the OSI internet MIB agent (OSI view of the SNMP MIB-II for TCP/IP). The <host> argument specifies the name of the host where the agent runs.

The option -f represents the use of filter expressions, and its detail is described in the following section. If this option is not given in the command line, all the event reports from the <agent> are sent to the management entity running the *evsink*, without any filtering.

The Filter Expressions

A filter expression is used to construct a CMISFilter value. The CMISFilter contains attribute value assertions (AVA) that are grouped with the logical operators AND, OR and NOT. The CMISFilter is what the EFD object uses to select the desirable event reports for its corresponding *evsink*.

In the filter expressions, there are characters used to represent each AVA and logical operators such as “>=“ for “greater or equal” and “!” for “NOT”.

The <filter> argument specifies the attribute names used in the filter expression.

For example,

```
evsink SMA artie -f “((systemId=artie) & (objectClass=mailObj))”
```

tells the EFD to select only the event reports related to a mailObj MO instance with a systemId of “artie”.

Similarly, the <eventType> argument specifies the names of the event types that are of interest. For instance,

```
evsink SMA artie -f “((eventType=objectCreation) | (eventType=objectDeletion))”
```

tells the EFD to select only reports about the creation or the deletion of any object instance.

However, the use of the argument <filter> is not possible yet at this point. The manual has mentioned that before using an attribute name in the filter expression, a function for it has to be defined and it has to be registered in some sort of syntax table. But up to the present time, the exact locations of those syntax tables are still unknown. So only the argument <eventType> can be used in a filter expression at this moment.

4.3.4 A Working Example of Using Evsink

This section shows an example that I have implemented for showing the capability of *evsink*. Suppose an agent is running at station “artie”, and the stations “pegasus” and “merylin” are designated as managers to monitor the activities of the agent by receiving all of its event reports. Also, for simplicity, the station “pegasus” is the only one that can

perform management operations on the agent in this example and the demonstration is divided into three phases:

Phase one: Setting up

At artie: Start the agent by just typing "sma" at the system prompt.

At merylin: According to the syntax mentioned previously, start an *evsink* by typing the following at the prompt.

evsink SMA artie (assuming all the events reports are desirable)

At pegasus: Activate the Xwindow. Then open a new window and run an *evsink* by entering the same command line as the one above in merylin.

Open another window for performing management operation on the agent.

Phase two: Performing management operations

At pegasus: Perform a *M_Create* at the window for management operations by entering.

mcreate SMA artie -c mailObj2 -i systemId=artie@mailObjId=carlos

This tells the agent to create an object instance of the object class "mailObj2" with an object ID "carlos".

At artie: A message about the creation of this object instance is displayed on the screen, and an event report is sent off to the connecting *evsinks* created by the other two stations.

Phase three: Getting the event reports

At pegasus: At the window running the *evsink*, an event report about the creation of the above object instance is received.

At merylin: The same event report is also received.

The overall picture of the three phases described above is shown in Figure 4.3.

Figure 4.3: The Overall Picture of the Demonstration for evsink.

As long as a managing station (like pegasus) has an *evsink* running in one of its windows, it can execute any other applications in other windows, and the *evsink* will automatically gather all the event reports sent out by the designated managed station (like artie).

In the example just described, the object instance of MO class "mailObj2" consists of three thresholds for counting the number of emails in the user's mailbox. The user at pegasus can work on other applications, while *evsink* is monitoring the mailbox for the user at the same time. Once the number of the user's emails crosses one of the three thresholds in the appropriate direction, an event report is sent off from artie to the *evsink*

attributes are supported by the remote SNMPv1 agents is also kept, so that the IQA does not request attributes that have been determined to be non-existent already.

Asynchronous Trap to Event-Report translation utilises the *proxyKS* object, which is an instance of an OSIMIS Knowledge Source. It listens on the incoming SNMPv1 Trap UDP ports (e.g. 162) that have been configured for each remote SNMPv1 agent.

4.4.3 The General Build Process

In order for the IQA to manage a SNMP device, a management interface between the two entities is needed. Providing a management interface for a SNMP manageable device to a OSI management platform is achieved by translating the SNMP MIB into an GDMO MIB and using a GDMO compiler to produce a .sif (Simple Input Format) file and an oidtable file which are used at run-time together with other files. The general build process is shown in Figure 4.7[6].

Figure 4.7: General Build Process of the OSIMIS IQA

The translation process for producing the .sif and oidtable files is largely automated, and involves converting the SNMP SMI into its equivalent GDMO codes in

4.4 Realizing the Agent-Manager Model with a SNMP Agent

In order to realize an Agent-Manager model with a SNMP agent in an OSI network management system, the SNMP agent has to be integrated into the OSI environment so that the OSI communications protocol - namely CMIP - can be able to manage the SNMP agent like any other OSI's. The issues in accomplishing this management coexistence of CMIP and SNMP, and the approach taken by the OSIMIS generic application gateway, namely the IQA (Internet Q-Adapter), are discussed in [7]. The following part of this section presents the structure and the usage of the IQA.

4.4.1 Introduction to the OSIMIS Internet Q-Adapter (IQA)

The Internet Q-Adapter is a generic gateway or a proxy agent, provided by OSIMIS, which offers efficient mapping capabilities between CMIP and SNMP. As illustrated in Figure 4.4[7], this gateway is acting as an intermediate hop in the manager/agent communication.

Figure 4.4: Manager/Agent Communication Paths

The IQA allows an OSI management platform to manage one or more SNMP manageable devices. In operation, the IQA takes CMIP requests and maps these into SNMP requests. SNMP responses and traps are translated into CMIP replies and events. There are three different cases in which the IQA can carry out the mappings:

- 1) The IQA resides on the agent station, and it receives CMIP requests of the OSI manager from the management network. It translates them to SNMP ones and forwards them to the local SNMP agent which replies by sending out SNMP responses or traps. Then, the IQA again translates them back to CMIP ones and sends them to the OSI manager through the network. Therefore, in this case, only CMIP PDUs are transmitted between the manager and the agent stations.
- 2) Conversely, the IQA can also be residing on the manager station, so that here, only SNMP PDUs are being transmitted between the manager and the agent stations.
- 3) Furthermore, the IQA can be residing on a stand-alone station translating the request and responses between the manager and the agent stations. In this case, both CMIP and SNMP PDUs are transmitted in the management network.

These three events are depicted in Figure 4.5.

Also, the OSIMIS IQA tends to be state-based as opposed to stateless, which means that a MIB representation or image of the SNMP MIB is held inside IQA in contrast to a stateless system where no information of the MIB is stored[6].

Figure 4.5: The Three Cases of IQA Mappings

In the stateless approach, a proxy does not maintain any MIB data of the SNMP agents. Instead, for each received CMIP request, the proxy generates one or more SNMP requests to the SNMP agent, in order to achieve the same intent as the CMIP request. This is why the state-based approach usually provides better response time, but it has the drawback that the data retrieved might not be current. In this approach, the polling frequency used to update the locally replicated MIB has a significant effect on the accuracy of the response.

In fact, the IQA is not totally state-based in that it uses cached data up to a short timeout. If the currently held data fails some test of timeliness, a remote SNMP request is generated to refresh the data before a reply is sent to the managing process.

With the use of the IQA, a CMIS manager can manage several SNMP agents, each with a different proprietary SNMP MIB. The IQA depending on the CMIP manager could reside in the same platform or could be established as a stand-alone device. The gateway approach of the IQA adds manageability to a whole world of resources with SNMP management capabilities, but the usage of the OSI model is restricted to the semantics of the SNMP model. Because it is a subset of the OSI model, there are certain commands on the OSI side of the IQA that do not have their equivalents on the SNMP side.

4.4.2 The Structural Decomposition of IQA

The structural decomposition of the Internet Q-Adapter gateway is shown in Figure 4.6[7]. At start-up, an instance of each of the IQA system, *proxySystem*, *cmipsnmpProxy* and *remoteSystem* classes is instantiated. The *proxySystem* object represents the gateway's local resources, whilst the *remoteSystem* object represents the remote SNMPv1 systems.

The *cmipsnmpProxy* object reads the initial configuration requirements and creates a *cmipsnmpProxyAgent* object, a *remoteSystem* object and an *SnmprMIBAgent* object for each remote SNMPv1 system. The *remoteSystem* objects can only be created successfully if a poll of the remote SNMPv1 agent receives a response. The *SnmprMIBAgent* objects encapsulate an SNMPv1 protocol interface.

A tree of *SnmplmageMO* MOs, corresponding to objects held at the remote SNMPv1 agent, will be built up below the respective *remoteSystem* objects in response to incoming CMIS requests. MO class descriptions are held within the

SnmplImageMOClassInfo meta-class instances, which are themselves constructed into an MIT during the initialisation phase.

Figure 4.6: The Structural Decomposition of the IQA

The *SnmplImageMOs* utilise the meta-class information to determine whether the corresponding SNMP SMI objects are single or multiply instanced. If multiply instanced, then the INDEX attributes are indicated so that received object values can be converted into Relative Distinguished Names in CMIS responses. Meta class information on which

attributes are supported by the remote SNMPv1 agents is also kept, so that the IQA does not request attributes that have been determined to be non-existent already.

Asynchronous Trap to Event-Report translation utilises the *proxyKS* object, which is an instance of an OSIMIS Knowledge Source. It listens on the incoming SNMPv1 Trap UDP ports (e.g. 162) that have been configured for each remote SNMPv1 agent.

4.4.3 The General Build Process

In order for the IQA to manage a SNMP device, a management interface between the two entities is needed. Providing a management interface for a SNMP manageable device to a OSI management platform is achieved by translating the SNMP MIB into an GDMO MIB and using a GDMO compiler to produce a .sif (Simple Input Format) file and an oidtable file which are used at run-time together with other files. The general build process is shown in Figure 4.7[6].

Figure 4.7: General Build Process of the OSIMIS IQA

The translation process for producing the .sif and oidtable files is largely automated, and involves converting the SNMP SMI into its equivalent GDMO codes in

the OSI environment. The tool used to carry out this conversion is *imibtool*. It currently only supports SNMPv1 MIBs and does not perform a 100% conversion, so some manual or hand post-processing needs to be carried out to get the GDMO codes into the right shape for the GDMO compiler. The compiler takes the post-processed GDMO codes as its input and produces a .sif file and an oidtable file. However, *imibtool* does not deal with the conversion of traps, as these do not appear in the SNMP SMI. Traps are dealt with separately after the GDMO compilation stage and are not part of the conversion process.

The .sif file produced is specific to the structure of the SNMP MIB being proxied for. If more than one SNMP MIB is to be proxied for, then only one .sif file needs to be produced for identically structured MIBs. Where SNMP MIBs differ in their structure, a separate .sif file is produced for each SNMP MIB.

The actual steps of the general build process are as follows:

- 1) Set up the environmental variables GDMODIR, OSIMISETCPATH and SMICINCL.
- 2) Set up a MIB_include_file which specifies the names of all the SNMP MIBs to be translated into GDMO.
- 3) Add SNMPv1 MIB group information to smdump_gdmo.ic
- 4) Call imibtool to create GDMO code.
- 5) Perform manual post-processing.
- 6) Call GDMO compiler to create a .sif file and two oidtable files.
- 7) Set up IQA communications information.

More details of each step are shown in [6]. The structure of the OSIMIS directories, where the IQA package is kept, is shown in Figure 4.8.

Figure 4.8: OSIMIS Proxy Directory Structure

4.4.4 A Working Example of Using the IQA

This section shows an example that has been implemented for illustrating a method of setting up and using the IQA to manage a SNMPv1 agent on a workstation. In this case, the SNMPv1 agent only supports MIB-II (RFC1213). Therefore, it needs an GDMO representation of the RFC1213, which is a .sif file for the RFC. This .sif file is supposed to be obtained using the steps from 2 to 6 (mentioned in the previous section). However, the .sif file for RFC1213 is already provided by the IQA as one of the sample .sif files named as "rfc1213.sif". Also, a modified version of the oidtables is provided as well. Therefore, the steps from 2 to 6 can be skipped in this example.

Configuring the IQA

a) Setting up the environmental variables:

In my case, they are set up as follows.

```
setenv GDMODIR          = /home/carlos/osimis/proxy/iqa/etc/gdmodir
```

```
setenv OSIMISETCPATH    = /home/carlos/osimis/proxy/iqa/etc/test
```

```
setenv SMICINCL         = /home/carlos/osimis/proxy/iqa/imibtool/mibs
```

The first environmental variable is concerned with the related files of the *gdmo* compiler, while the third one is concerned with the MIBs files that are going to be translated by the *imibtool*.

b) Setting up the iqastartupagents.icm file

The first file responsible for proper IQA communications is the "iqastartupagents.icm" in the directory OSIMISETCPATH/iqadir/. My version of the file is as follows:

```
"artie", { snmpUDPDomain:0x86:75.04.3f 00.a1 }, snmpV1, { IIMCrfc1213}, 3, 2000,  
"private", 0, 162, "SNMP-trap"
```

Parameter 1 to 8 is mandatory for proper communications between the IQA and the SNMPv1 agent it is proxied for. Parameter 9 and 10 is optional in the sense that it is only concerned with SNMP traps. Here is the description of each parameter:

- i) the remote agent's Id - it can be a character or anything else.
- ii) the ipaddress/port number of the SNMP agent being proxied for - this is actually the hexadecimal representation of the ipaddress/port number of the machine in which the agent is located. In this case, the station *artie* has an ipaddress of

2

OF/DE

2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

134.117.4.34.63 and a port number of 161, and so its ipaddress/port number in hex is 0x86:75:04:3f:00:a1.

- iii) the remote management protocol which must be snmpV1.
- iv) one of more MIB OIDs that are remotely supported - this entry corresponds to what MIB(s) the IQA is going to proxied for.
- v) the maximum number of retransmissions upon a timeout of request.
- vi) the retransmission timeout in milliseconds.
- vii) the required community string for the remote agent - in my case, the SNMP agent that the IQA is to proxy for has two different community strings. One is for "reading" attributes, while the other is for "writing" attributes. Since this example is trying to show not only how to retrieve attributes through IQA but also to modify the SNMP attributes through it, the writing community string "private" is used instead of the reading community string "public".
- viii) a "magic" number - a value of "1" permits the IQA to recover from failed communications with agents that do not return "TooBig" SNMP request responses.
- ix) the port that the IQA listens for SNMP traps - this port is usually 162 which is a privileged port. This means that permission is required to listen to it. Normally, it is achieved by running the IQA as root, and this can easily be done with the help of the network administrator. My permission listing for the executable "iqa" looks like this:

```
--rwsrwx--- 1 root  2138112 Feb 14 09:53 iqa*
```

recompiling using the OSIMIS generic manager application *mset* for adapting any arising situations.

ii) The following two procedures need to be added into the file "mailObj.inc.cc":

```
int    wakeUp (char*);
int    buildReport (int, int, void*&, Bool&);
```

The first procedure is for triggering the alarms when the attribute crosses the designated thresholds. The second one is for constructing the reports that will be sent with the event report when the alarms occur.

Finally, repeat the 10 steps described in section 4.2.1 to get a new library file and link it to the agent. Then, run the "sma" again. The SMA will start polling the attribute after it starts. Here is the screen display when the SMA is doing the polling:

```
tailor : /home/pegasus/carlos/osimis/proxy/iga/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iga/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
application entity title: SMA-artie
```

```
initialising meta-class information...
top
system
.
```

```
setting up notification to event record mappings...
objectCreation          eventLogRecord
objectDeletion          eventLogRecord
```

```
.
initialising MIB...
restarting logs
system {systemId=artie}
subsystem {subsystemId=4}
.
```

```
.
listening. .
mailObj::update nMails 2
      GaugeThreshold::check
mailObj::update nMails 3
```

```
GaugeThreshold::check
mailObj::update nMails 4
```

representation for the RFC1213 objects. The MIB requires an OID, "iimcRFC1213", which must also be listed in "oidtable.gen".

d) Setting up the isoentities file

An entry needs to be made in the file "isoentities" to let the IQA be able to run on the desired station. In my case, the entry was:

```
artie   IQA   1.17.5.14.0 #605/Internet=134.117.4.63+11010
```

"artie" is the name of the desired station, and the string "134.117.4.63" is its IP address.

e) The oidtable.at and oidtable.gen files

Although a modified version of the files "oidtable.at" and "oidtable.gen files" has already been given in OSIMIS for running IQA, it is worthwhile to mention how they differ from their counterparts in running the SMA. In the IQA version of the file "oidtable.at", not only has it got all the entries for the attributes and notifications of the MOs used in the SMA, but it also has all the entries for that of the SnmpImageMOs used in the IQA. Some of the entries of the SnmpImageMOs are shown below:

internetSystemId:	iimcAutoName.1.3.6.1.2.1.1	:SnmpGenericRDN
interfacesId:	iimcAutoName.1.3.6.1.2.1.2	:SnmpGenericRDN
atId:	iimcAutoName.1.3.6.1.2.1.3	:SnmpGenericRDN
ipId:	iimcAutoName.1.3.6.1.2.1.4	:SnmpGenericRDN
icmpld:	iimcAutoName.1.3.6.1.2.1.5	:SnmpGenericRDN
.		
.		
sysDescr:	iimcAutoObjAndAttr.1.3.6.1.2.1.1.1	:SnmpDisplayString
sysObjectID:	iimcAutoObjAndAttr.1.3.6.1.2.1.1.2	:SnmpOID
.		
.		
ifNumber:	iimcAutoObjAndAttr.1.3.6.1.2.1.2.1	:SnmpInteger
ifEntryId:	iimcAutoName.1.3.6.1.2.1.2.2.1	:SnmpGenericRDN
ifIndex:	iimcAutoObjAndAttr.1.3.6.1.2.1.2.2.1.1	:SnmpInteger

Similarly, the difference in the file "oidtable.gen" is that the IQA version has all the entries concerning the object class information and the name bindings for the SnmpImageMOs.

Some of these entries are listed below:

```
internetSystem:      iimcAutoObjAndAttr.1.3.6.1.2.1.1
interfaces:          iimcAutoObjAndAttr.1.3.6.1.2.1.2
at:                  iimcAutoObjAndAttr.1.3.6.1.2.1.3
ip:                  iimcAutoObjAndAttr.1.3.6.1.2.1.4
.
.
ifEntry:             iimcAutoObjAndAttr.1.3.6.1.2.1.2.2.1
ipForwardEntry:      iimcAutoObjAndAttr.1.3.6.1.2.1.4.24.2.1
tcpConnEntry:        iimcAutoObjAndAttr.1.3.6.1.2.1.6.13.1
udpEntry:            iimcAutoObjAndAttr.1.3.6.1.2.1.7.5.1
egpNeighEntry:       iimcAutoObjAndAttr.1.3.6.1.2.1.8.5.1
.
.
internetSystem-remoteSystemNB: iimcAutoNameBinding.1.3.6.1.2.1.1
interfaces-remoteSystemNB:     iimcAutoNameBinding.1.3.6.1.2.1.2
at-remoteSystemNB:             iimcAutoNameBinding.1.3.6.1.2.1.3
```

Running the IQA

After having everything set up, the IQA will start up by just typing "iqa" at the system prompt. One should note that the IQA needs to be run as root for it to have the permission to listen to port 162, and accordingly, receive SNMP traps. Once the IQA runs up, it creates the necessary OIDs and connects to the running SNMP agent on the station specified in the file "iqastartupagent.icm". As a result, the screen display should look like the following:

```
taylor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistaylor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
application entity title: IQA-artie
Buffering class name [internetSystem]
Buffering class name [interfaces]
.
.
Buffering class name [udpEntry]
Buffering class name [egpNeighEntry]

initialising meta-class information...
top
system
```

discriminator
 eventForwardingDiscriminator
 log
 logRecord

cmipsnmpProxyAgent
 snmpSecurityParameter

setting up notification to event record mappings...

qualityofServiceAlarm eventLogRecord
 objectCreation eventLogRecord

internetAlarmColdStart eventLogRecord
 internetAlarmWarmStart eventLogRecord
 internetAlarmLinkDown eventLogRecord
 internetAlarmLinkUp eventLogRecord
 internetAlarmAuthenticationFailure eventLogRecord
 internetAlarmEgpNeighbourLoss eventLogRecord
 internetAlarmTestEnterpriseSpecific eventLogRecord

initialising MIB...

restarting logs

system {systemId=artie}

proxySystem {systemId=NULL}

Processing: "artie", {snmpUDPDomain:0x86:75:40:02:00:a1}, snmpV1, {IIMCrdc1213}, 3, 2000.
 "private", 0, 162, "SNMP-trap"

Party connect PS = 2240552 SD = 8

remoteSystem::remoteSystem() for systemId=artie

attributes initialised ...

new -> Internet oid 1.3.6.1.2.1.1.1 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.2 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.3 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.4 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.5 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.6 newOIDlen 8

new -> Internet oid 1.3.6.1.2.1.1.7 newOIDlen 8

Completion of requestId 1103527591 TXed 835324463 S

cmipsnmpProxyAgent Object created

cmipsnmpProxy {systemId=NULL@cmipsnmpProxyId=NULL}

snmpSecurityParameter

{systemId=NULL@cmipsnmpProxyId=NULL@snmpSecurityParameterId=NULL}

addClass internetSystem

addClass interfaces

addClass ifEntry

addClass egpNeighEntry

addClass snmp

listening...

Managing the SNMP agent through IQA

After getting the IQA running, the OSIMIS user can test the IQA using the OSIMIS generic manager applications. This section demonstrates some of the ways of managing a SNMP agent through the IQA. All the examples in this section are done in a network management system shown in Figure 4.9. In this case, which management (manager or agent) system the IQA is residing on, does not make a big difference in terms of the impact on the bandwidth usage of the management network. However, when a Monitor Metric MO is used to monitor the SNMP agent through the IQA, the choice of IQA's location will have a much greater effect on the bandwidth usage. More details of this issue have been discussed in section 4.5.2.2.

Figure 4.9: A Network Management System with a SNMP agent.

The following are some of the examples of managing a SNMP agent through the IQA:

- Retrieving all the SNMP ImageMOs, which are the MOs for the SNMP groups, in the second level of the OSIMIS MIT using the OSI scoping capability:

```
> mibdump IQA artie -s 2ndlevel
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
```

```
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
```

OIDs 374 : attributes 490 : classes 52

Linked Reply Get - id 0x000001, class cmipsnmpProxy, instance
 {systemId=NULL@cmipsnmpProxyId=NULL}, time 19960710225046.190397--400
objectClass: cmipsnmpProxy
 nameBinding: cmipsnmpProxy-systemNB
 cmipsnmpProxyId: NULL
 snmpMsgRetryLimit: 0
 snmpMsgTimeout: 0

Linked Reply Get - id 0x000001, class internetSystem, instance
 {systemId=artie@internetSystemId=NULL}, time 19960710225046.216622--400
objectClass: internetSystem
 nameBinding: internetSystem-remoteSystemNB
 internetSystemId: NULL
 sysDescr: "ALLDESC"
 sysObjectId: 1.3.6.1.4.1.42.2.1.1
 sysUpTime: 228724047
 sysContact: "ALLCONT"
 sysName: "artie.sce.carleton.ca"
 sysLocation: "ALLOC"
 sysServices: 72

Linked Reply Get - id 0x000001, class interfaces, instance
 {systemId=artie@interfacesId=NULL}, time 19960710225046.225010--400
objectClass: interfaces
 nameBinding: interfaces-remoteSystemNB
 interfacesId: NULL
 ifNumber: 2

Linked Reply Get - id 0x000001, class at, instance {systemId=artie@atId=NULL}, time
 19960710225046.231429--400
objectClass: at
 nameBinding: at-remoteSystemNB
 atId: NULL

Linked Reply Get - id 0x000001, class ip, instance {systemId=artie@ipId=NULL}, time
 19960710225046.238441--400
objectClass: ip
 nameBinding: ip-remoteSystemNB
 ipId: NULL
 ipForwarding: 2
 ipDefaultTTL: 255
 ipInReceives: 636318

Linked Reply Get - id 0x000001, class snmp, instance {systemId=artie@snmpId=NULL},
 time 19960710225046.553167--400
objectClass: snmp
 nameBinding: snmp-remoteSystemNB
 snmpId: NULL
 snmpInPkts: 9
 snmpOutPkts: 8

snmpOutGetResponses: 8

```
snmpOutTraps: 0
snmpEnableAuthenTraps: 1
```

10 replies received

- Retrieving the SNMP ImageMO with the objectClass "snmp" using the OSI filtering capability:

```
> mibdump IQA artie -s 2ndlevel -f "(objectClass=snmp)"
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Get Result - id 0x000001, class snmp, instance {systemId=artie@snmpId=NULL}, time
19960710225141.099654--400
```

```
objectClass: snmp
nameBinding: snmp-remoteSystemNB
snmpId: NULL
snmpInPkts: 16
snmpOutPkts: 15
```

```
snmpOutGetResponses: 15
snmpOutTraps: 0
snmpEnableAuthenTraps: 1
```

1 replies received

- Retrieving only the SNMP attribute "snmpInPkts" of the objectClass "snmp":

```
> mibdump IQA artie -s 2ndlevel -f "(objectClass=snmp)" -a snmpInPkts
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Get Result - id 0x000001, class snmp, instance {systemId=artie@snmpId=NULL}, time
19960710225305.793071--400
```

```
snmpInPkts: 23
```

1 replies received

- Setting the attribute "sysContact" of the SNMP system group through the IQA:

i) Showing the attribute before setting:

```
> mibdump IQA artie -c internetSystem -i
systemId=artie@systemId=artie@internetSystemId=NULL
```

Note: the first systemId corresponds to the machine at which the SNMP agent is located, while the second one corresponds to the machine on which the IQA is running.

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Get Result - id 0x000001, class internetSystem, instance
{systemId=artie@internetSystemId=NULL}, time 19930625114549.266266--400
```

```
objectClass: internetSystem
nameBinding: internetSystem-remoteSystemNB
internetSystemId: NULL
sysDescr: "ALLDESC"
sysObjectID: 1.3.6.1.4.1.42.2.1.1
sysUpTime: 95134421
sysContact: "before"
sysName: "artie.sce.carleton.ca"
sysLocation: "ME4242"
sysServices: 72
```

1 replies received

ii) Setting the attribute:

```
> mset IQA artie -c internetSystem -i
systemId=artie@systemId=artie@internetSystemId=NULL -w sysContact="after"
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Set Result - id 0x000001, class internetSystem, instance
{systemId=artie@internetSystemId=NULL}, time 19960625114641.654078--400
sysContact: "after"
```

1 replies received

iii) Showing the attribute again after the setting:

```
> mibdump IQA artie -c internetSystem -i
systemId=artie@systemId=artie@internetSystemId=NULL
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Get Result - id 0x000001, class internetSystem, instance
{systemId=artie@internetSystemId=NULL}, time 19960625114735.568869--400
```

```
objectClass: internetSystem
nameBinding: internetSystem-remoteSystemNB
internetSystemId: NULL
```

```

sysDescr: "ALLDESC"
sysObjectID: 1.3.6.1.4.1.42.2.1.1
sysUpTime: 95145047
sysContact: "after"
sysName: "artie.sce.carleton.ca"
sysLocation: "ME4242"
sysServices: 72
1 replies received

```

- Receiving an event report or a trap of "authenticationFailure" from the SNMP agent through the IQA:
 - i) In a manner similar to what has been illustrated in section 4.3.4, type in the following at the system prompt on the manager station.


```

evsink IQA artie      ( artie is the station on which the IQA is running )
          
```
 - ii) When the trap arrives at the IQA from the SNMP agent being proxied for, the IQA translates the trap to an event report and sends it out. Then, the manager station gets an event report for this trap through the *evsink* as follows:

EventReport - internetAlarmAuthenticationFailure

```

class: cmipsnmpProxyAgent
inst: {systemId=NULL@cmipsnmpProxyId=NULL@
      cmipsnmpProxyAgentId=artie}
time: 19960613145334.100344--4
report: {probableCause: 1.3.6.1.6.3.1.1.5.5}
        {attributeIdList: NULL}{objectInstanceList: NULL}
        {UnknownVarBindList: NULL}{internetTrapInfo: NULL}
        {perceivedSeverity: Indeterminate}{notificationId: 834692014}
        {CorrelatedNotifications: NO PRINT}{transportDomain: 1.3.6.1.6.1.1}
        {transportAddress: 867504c9}{accessControlInfo: commStr:SNMP-trap}
        {additionalInformation: NULL}}

```

4.5 Providing Thresholding Capabilities to MO Instances

In many situations of a network management system, it is useful to have the agent monitoring a particular attribute of a MO (e.g. bridgeForwardDelay), so that the manager can be notified right away from the agent when a particular attribute experiences certain

changes. In OSIMIS, there are two methods that can provide a polling scheme to a MO and each of them is described in the following sections

4.5.1 Modifying the GDMO File

The first method is to modify the original GDMO file of its MO class. Using the mailObj example presented previously, the following lines will have to be added to the GDMO file ("mailMIB.gdmo"):

```
-- ##### Package Templates #####
mailObjPackage PACKAGE
  ATTRIBUTES
    nMailsThld          GET-REPLACE,
  NOTIFICATIONS
    qualityofServiceAlarm. -- for nMailsThld

-- ##### Attribute Templates #####
nMailsThld ATTRIBUTE
  DERIVED FROM gauge-Threshold.
  REGISTERED AS { uclAttributeID 533 };
```

Also, there are a few changes that are needed in the mailObj C++ files:

- i) The main changes of the file "mailObj.inc.h" are as follows:

```
private:
#define MIInterval      5      /* the period between each polls */
#define NMailsThldLow   3      /* default low value */
#define NMailsThldHigh  5      /* default high value */
```

The last two values are the gauge thresholds since they are always specified in pairs of values: a triggering and a canceling threshold. The former will generate a notification when crossed only if the latter has been previously crossed in the opposite direction. This prevents the continuous generation of notifications when the measured value oscillates the triggering threshold.

The MO implementor can alter these three values in the .h file before recompiling to suit a particular known requirement. These values can also be changed after

recompiling using the OSIMIS generic manager application *mset* for adapting any arising situations.

n) The following two procedures need to be added into the file “mailObj.inc.cc”:

```
int    wakeUp (char*);
int    buildReport (int, int, void*&, Bool&);
```

The first procedure is for triggering the alarms when the attribute crosses the designated thresholds. The second one is for constructing the reports that will be sent with the event report when the alarms occur.

Finally, repeat the 10 steps described in section 4.2.1 to get a new library file and link it to the agent. Then, run the “sma” again. The SMA will start polling the attribute after it starts. Here is the screen display when the SMA is doing the polling:

```
taylor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
application entity title: SMA-artie
```

```
initialising meta-class information...
top
system
```

```
.
setting up notification to event record mappings...
objectCreation          eventLogRecord
objectDeletion          eventLogRecord
.
```

```
.
initialising MIB...
n .starting logs
system {systemId=artie}
subsystem {subsystemId=4}
.
```

```
.
listening. .
mailObj::update nMails 2
      GaugeThreshold::check
mailObj::update nMails 3
```

```
GaugeThreshold::check
mailObj::update nMails 4
```

GaugeThreshold::check
 mailObj::update nMails 5

When the attribute "nMails" crosses the designated thresholds in the proper direction, the manager station will be able to get an event report from the agent (e.g. SMA) using the *evsink*:

EventReport - qualityofServiceAlarm

```
class: mailObj
inst: {mailObjId=carlos}
time: 19960703231505.411351--4
report: AlarmInfo - probableCause: thresholdCrossed perceivedSeverity:
indeterminate thresholdInfo: nMailsThld 5
```

4.5.2 Using OSI Metric Monitoring Management Function

The second method is to use the OSI metric monitoring management function[5]. The whole idea behind monitor metric objects is to provide thresholding facilities in a generic fashion. Monitor metric objects may be instantiated within an application in an agent role, and be configured to monitor, at periodic intervals, an attribute of another real resource managed object. The observed attribute should be a counter or gauge; the metric object either observes it as is, or converts the observed values to a rate (derived gauge) over time. Statistical smoothing of the observed values is also possible if desired.

The significance of this facility is the attachment of gauge thresholds and tidemarks to the resulting derived gauge, which may generate quality of service alarms and indicate the high and/or low "water mark", as desired by systems using this function. In fact, the metric objects essentially enhance the "raw" information model of the observed object. The metric monitor functionality can be summarized as:

- *data capture*: through observation or "scanning" of a managed object attribute.

- *data conversion*: potential conversion of a counter or gauge to a derived gauge;
- *data enhancement*: potential statistical smoothing of the derived result; and
- *notification generation*: QoS alarm and attribute value change notifications.

The metric monitoring model is shown in Figure 4.10[5].

Figure 4.10: The Metric Monitoring Model

The metric objects offer the OSI management power through event reporting and logging, even if the "raw" observed management information model does not support such notifications. More importantly, these objects prevent the use of rate, thresholds and tidemarks in a way tied to specific managed objects but they allow the same flexibility and power dynamically, whenever a managing system needs it. Such a monitoring facility network reduces the management traffic between applications and the impact on the managed network by supporting an event-based operation paradigm. It should be

emphasised that these facilities are of "managing" nature, but are offered within a managed object cluster across a management interface.

4.5.2.1 Providing Thresholds to MOs for an OSI Agent

As already mentioned, the Monitor Metric MOs are the ones responsible for providing thresholding capabilities to MOs in a generic fashion. Also, it can provide or suspend a threshold whenever a MO needs this. For the mailObj MO developed previously, a Monitor Metric MO can easily be created in OSIMIS as in the following:

```
> mcreate SMA artie
  -c monitorMetric
  -a observedObjectInstance="systemId=artie@mailObjId=carlos"
  -a observedAttributeId="nMails "
  -a granularityPeriod="secs:10"
  -a severityIndicatingGaugeThreshold="{ Low:1 Switch: On High:3 Switch: On }"
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
Create Result - id 0x000001, class monitorMetric instance {scannerId=1} time
19960703230847.718391--400
objectClass: monitorMetric
nameBinding: scanner-system
scannerId: 1
administrativeState: unlocked
granularityPeriod: secs:10
operationalState: enabled
observedObjectInstance: systemId=artie@mailObjId=carlos
observedAttributeId: nMails
derivedGauge: 1
severityIndicatingGaugeThreshold: {Low:1 Switch:On High:3 Switch:On}
severityIndicatingTideMarkMax: maximum: cur 1 prev 0.00 reset 960703230847Z
severityIndicatingTideMarkMin: minimum: cur 1 prev 0.00 reset 960703230847Z
previousScanCounterValue: 0
previousScanGaugeValue: 0
counterOrGaugeDifference: False
```

If the pair of the threshold values or the time in between each poll needs to be changed, they can be modified any time using *mset*. However, before this modification, the state of the attribute "administrativeState" needs to be set as "locked". Thus, first:

```
> mset SMA artie -c monitorMetric
-i systemId=artie@scannerId=1
-w administrativeState=locked
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Set Result - id 0x000001, class monitorMetric, instance {scannerId=1}, time
19960703230932.197017--400
```

```
administrativeState: locked
```

```
1 replies received
```

Then the pair of threshold values can be modified to the desired values as follows:

```
> mset SMA artie -c monitorMetric -i systemId=artie@scannerId=1 -w
severityIndicatingGaugeThreshold= "{Low:4 Switch:On High:6 Switch:On}"
```

```
tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52
```

```
Set Result - id 0x000001, class monitorMetric, instance {scannerId=1}, time
19960703231048.299396--400
```

```
severityIndicatingGaugeThreshold: {Low:4 Switch:On High:6 Switch:On}
```

```
1 replies received
```

From this time, the Monitor Metric MO scans the attribute “nMails” in every time period specified in the attribute “granularityPeriod”. When the attribute crosses the thresholds specified in the attribute “severityIndicatingGaugeThreshold”, the Monitor Metric MO will send off an event report on behalf of the mailObj MO; the manager station will then get the event report through *evsink* in the following manner:

EventReport - qualityofServiceAlarm

```
class: monitorMetric
inst: {scannerId=1}
time: 19960703230057.459451--4
report: AlarmInfo - probableCause: thresholdCrossed specificProblems:
(resourceRequestRate) perceivedSeverity:
indeterminate thresholdInfo severityIndicatingGaugeThreshold 4
monitoredAttributes: {observedObjectInstance:
systemId=artie@mailObjId=carlos observedAttributeId: nMails}
```

4.5.2.2 Providing Thresholds to the Attributes for a SNMP Agent

Providing thresholding capabilities to the SNMP ImageMOs (or the SNMP attributes) is very important in integrating a SNMP agent into an OSI network management environment. The reason is that, unlike the OSI GDMO, there is no SNMP specialised way to define thresholds for SNMP attributes. Therefore, the easiest way for an OSI manager to monitor a SNMP attribute is to use a Monitor Metric MO.

In OSIMIS, a manager application can monitor a SNMP attribute by simply using a Monitor Metric MO through the IQA. Figure 4.11 shows how Monitor Metric MOs provide thresholding capabilities to SNMP attributes in the OSIMIS environment.

Figure 4.11: Providing thresholds to SNMP attributes

The Monitor Metric MOs scans the SNMPImageMOs, which are representations of the SNMP attributes, for checking whether or not the designated attributes have crossed their thresholds. When a SNMP attribute has crossed the thresholds defined in the

corresponding Monitor Metric MO, it will send off an event report on behalf of the SNMP attribute to the EFD MO. If the event report passes the restrictions designated at the EFD MO, then it will be sent off to the managing station.

As mentioned earlier, the location of IQA in this case has a great impact on the network bandwidth usage. The reason is that, when a manager application has specified a Monitor Metric MO at the IQA for monitoring a particular SNMP attribute, the IQA as a proxy agent will poll the SNMP attribute on behalf of the manager application, at a periodic interval specified in the Monitor Metric MO. Therefore, putting the IQA on the agent station can localise all the polling between the IQA and the SNMP agent. This is particularly important in a bandwidth-constrained environment or in the case that a SNMP attribute needs to be monitored very closely.

The following is a simple example showing the method of monitoring a SNMP attribute using the Monitor Metric MO through the IQA. The attribute "snmpOutTraps" is the one that needs to be monitored. Then, similar to what has been illustrated in the previous section, a Monitor Metric MO can be created as follows:

```
> mcreate IQA artie
    -c monitorMetric
    -a observedObjectInstance=
        "systemId=artie@systemId=artie@snmpId=NULL"
    -a observedAttributeId="snmpOutTraps"
    -a granularityPeriod="secs:5"
    -a severityIndicatingGaugeThreshold="{Low:42 Switch:On High:45 Switch:On}"
```

Likewise, the pair of threshold values or the time period between each poll can be modified using *mset*, as suggested earlier. When the attribute "snmpOutTraps" of the SNMP agent crosses the designated thresholds, the Metric Monitor MO will send off an

event report on behalf of the SNMP attribute. Using *evsink*, the manager application will receive the report, as in the following manner:

EventReport - qualityofServiceAlarm

class: monitorMetric
inst: {scannerId=1}
time: 19960623172253.081259--4
report: AlarmInfo - probableCause: thresholdCrossed specificProblems:
{resourceRequestRate} perceivedSeverity:
indeterminate thresholdInfo: severityIndicatingGaugeThreshold 45
monitoredAttributes: {observedObjectInstance:
systemId=artie@systemId=artie@snmpId=NULL
observedAttributeId: snmpOutTraps}

CHAPTER 5

IMPLEMENTING HIERARCHICAL MANAGER-TO-MANAGER COMMUNICATIONS

5.1 Implementing Manager-to-Manager Communications

As explained previously in section 2.4.3, employing a hierarchy of managers is the most efficient way to reduce CMIP traffic; in fact, this applies to any types of network traffic. For this reason, a manager-to-manager communications scheme is needed in order for different managers to communicate with each other in the hierarchy. In the OSI context, it can be accomplished using the OSI event-report-management function, which translates to the use of *evsink* in the OSIMIS environment. In fact, not only OSI has the capability to implement a manager-to-manager communications scheme, but SNMPv2 also has a so-called manager-to-manager MIB for providing communications between managers. An overview of the manager-to-manager MIB of SNMPv2 is presented in section 5.1.1. Further, in order to show what services a good manager-to-manager communications scheme has to offer, a comparison of the manager-to-manager MIB of SNMPv2 with the OSIMIS *evsink* is discussed in the section 5.1.2. Finally, an implementation of a hierarchical manager-to-manager communications scheme using the OSIMIS platform is described in the section 5.2.

5.1.1 The Manager-To-Manager (M2M) MIB for SNMPv2

The M2M MIB for SNMPv2 consists of a set of objects that describe the behaviour of an SNMPv2 entity that acts in a manager role. The MIB consists of two groups: the `snmpAlarm` group; and the `snmpEvent` group[1].

The `snmpAlarm` group is used to define a set of thresholds for network performance. If a threshold is crossed in the appropriate direction, an alarm is generated and sent to the central console. The alarm group consists of a single table, `snmpAlarmTable`, and a single scalar variable. Each entry in the table specifies a particular variable to be monitored, a sampling interval, and threshold parameters.

The `snmpEvent` group is used to support the definition of events and the configuration of notifications. The `snmpEvent` group consists of two tables, `snmpEventTable` and `snmpEventNotifyTable`, plus associated scalar objects.

Each entry in the `snmpEventTable` defines an event. Some events are triggered by an associated condition in the `snmpAlarmTable`; others are triggered on behalf of conditions defined in a NOTIFICATION-TYPE macro.

The `snmpEventNotifyTable` defines notifications that should occur when an associated event is fired. An `InformRequest` PDU is used to send off the event, by one SNMPv2 manager entity to another, to provide management information. After the receiving entity has successfully obtained the `InformRequest` PDU, it sends back a `Response` PDU to the sending entity. However, if the sending entity cannot obtain the `Response` PDU in a certain period of time, the `InformRequest` PDU is retransmitted until the number of retransmissions has reached a certain value.

5.1.2 Comparison of the M2M MIB with the OSIMIS Evsink

In comparing the two different mechanisms, both of them are put into the same situation, in which only two managers are trying to communicate with one another. Figure 5.1 shows how each of them copes with the given situation.

In the case of using M2M MIB for SNMPv2, a MIB is defined for the two managers; all the thresholds and event notifications of the variables are defined in the MIB. When an event notification is sent out, an InformRequest PDU is used with a Response PDU. This is done in order to ensure that the receiving end has obtained the notification successfully. However, the M2M MIB is not capable of selecting desirable event reports. Every event report, generated by the alarms defined in the MIB, is forwarded from one end to the other without any filtering.

Figure 5.1: Comparing M2M MIB with Evsink.

In the case of adopting the OSIMIS *evsink*, each manager uses an *evsink* to monitor the other entity. All the thresholds and event notifications of each MO are defined in the MO itself. An EFD control object is created to select the desirable event reports for the managers, at the corresponding agents by each *evsink*. In terms of reducing the flow of unnecessary management information, the ability to select desirable event reports for management entities is indispensable. It is particularly important in real-time or other bandwidth-constrained environments.

Also, there is a conditional package in the EFD object class which can specify the mode for reporting events (confirmed or nonconfirmed), in order to ensure that the event reports arrive at the receiving end. If one or more managing stations are added into the situation, the EFD control object at the agents may include a `backupDestinationListPackage`. This is a list, in priority order, of application entities that may be used as an event destination if the EFD's primary destination entity fails.

Therefore, compared with the M2M MIB for SNMPv2, the OSIMIS *evsink* is preferable for implementing a M2M communications scheme. Consequently, this is the direction taken when approaching a hierarchical M2M communications scheme in the next section.

5.2 Implementing Hierarchical Manager-To-Manager Communications using OSIMIS

5.2.1 Approaching a Desirable Implementation

By using the infrastructure of OSIMIS and the generic manager applications it provides, there are several ways to implement a hierarchical manager-to-manager communications scheme. This section describes the process of deriving the desirable implementation for this research project.

First of all, there are several criteria that have to be considered when implementing a hierarchical manager-to-manager communications scheme. These criteria are stated as follows:

- The implementation must be able to reduce the overall traffic in the network.
- The mid-level Manager is a "dual-role entity", which means that it acts in both an agent role and a manager role. To managers requesting information from the mid-level manager, it is an agent, while to those agents it queries, it is a manager.
- The mid-level manager must be able to manipulate or to summarize the data received from the agents or managers in the lower level into useful information, so that the high-level manager can have a global view of the sub-network under a mid-level manager.

For instance, the mid-level manager should be able to keep track of the number of alarms for attributes (e.g.: snmpGenErrors), and then notify the high-level manager, after the number of alarms for any attribute has crossed some thresholds.

- The high-level manager must be able to alter the conditions under which it should be notified by the mid-level manager.

First Approach

Considering the criteria just mentioned, the layout of the first approach using the OSIMIS platform is determined, and is shown in Figure 5.2.

Figure 5.2: The First Approach for Implementing Hierarchical M2M Communications.

The procedures of how the first approach works are as follows:

1. On the low-level agent station, the pollings of MOs are done locally and the agent application emits all event reports triggered by any MOs to the mid-level manager station.

- 2a. On the mid-level manager station, an *evsink* is used for receiving all the event reports emitted from the low-level agent. In addition, all the information of the event reports are stored in a text file, which is treated as the logical real resource of an *thldMonitor* MO residing with the OSI Agent SMA. The updating procedure of this MO is responsible for interacting with the real resource (an example being the procedure “updateMails” of the *mailObj*). This procedure extracts data from the text file containing the event reports information. After some processing has been done on the data, the results are reflected in the *thldMonitor* MO by each of its attributes containing: a) the name of a certain attribute of a MO; and b) the number of times this attribute has caused an alarm at the lower-level agent station.

Conceptually, the *thldMonitor* MO stores the event information as follows:

```

objectClass: thldMonitor
nameBinding: thldMonitor-system
thldMonitorId: QualityOfServiceAlarm
thld1Value: nMails 5
thld2Value: snmpOutTraps 3
.
.
```

- 2b. A Monitor Metric MO for monitoring the *thldMonitor* MO also resides with the OSI Agent. If any attributes of the latter MO cross the thresholds designated by the high-level manager, the Metric Monitor MO will send off an event report about the given situation to the high-level manager station. The interactions between the entities in the mid-level manager station are shown in Figure 5.3.
3. On the high-level manager station, the event report is received using *evsink*. Then, from the information given by the *thldMonitor* MO in the event report, the high-level manager is notified about the situation of the low-level agent.

Figure 5.3: Interactions in the Mid-Level Manager Station of the First Approach.

This implementation looks relatively simple because only the updating procedure of the thldMonitor MO needs to be hand-written from "scratch". Any other requirement is already provided in the OSIMIS environment. However, this implementation has two major shortcomings:

- The text file containing the event reports information can only be accessed after the *evsink* quits. Therefore, the thldMonitor MO can only update its attributes after the high-level manager suspends the event monitoring of the lower level sub-network. Thus, the whole implementation acts like an event logs analyzing facility, rather than a manager-to-manager communications scheme which needs to have real-time event monitoring capability.
- When trying to reflect the event reports information by the attributes of the thldMonitor MO as described earlier, only a limited amount of information can be

reflected using the attributes. The reason is that the thldMonitor MO or any MO can only have a fixed number of attributes at one time. All the attributes of a MO are predefined in the GDMO file, and the only way of modifying the number of attributes in a MO is by altering the GDMO file. However, whenever the GDMO file of a MO class is altered, the MO class is treated as a new class and it needs to be added to OSIMIS again, using the 10 steps described in section 4.2.1. Therefore, one can notice that it is very time-consuming if the number of attributes of the thldMonitor MO have to be modified often.

Second Approach

After considering the shortcomings of the first approach, a second approach is made successful by improving the access of the event reports information gathered from *evsink* in the mid-level manager station. In the second approach, instead of having the updating procedure of the thldMonitor MO accessing the event reports information from *evsink* through a text file, the whole *evsink* program has been integrated into the updating procedure of the MO. Therefore, the updating procedure will not have the access problem of the event reports information, as in the first approach. Figure 5.4 shows the interactions between entities in the mid-level manager station in the second approach.

However, although integrating the *evsink* into the updating procedure can solve one of the major shortcomings of the previous approach, the other shortcoming, which is the limitation of event reports information that the thldMonitor MO can reflect, is still left unsolved by the second approach.

Figure 5.4: Interactions in the Mid-Level Manager Station of the Second Approach.

Third Approach

Starting from the first approach, the thldMonitor MO is designed to act as a “storage” for the event reports information of the mid-level manager. In fact, it is a not good idea to use a MO for storing the event reports information as it lacks the flexibility of expanding the “storage”. So, after considering this problem, the layout of the third approach is:

- Modify the original *evsink* program so that it not only has the capability to receive event reports from agents, but can also manipulate the report data into useful information, which is stored in a data structure inside the program itself.
- The modified version of *evsink* will also be responsible for monitoring the event reports received from the low-level agent. Therefore, a Monitor Metric MO is no longer needed.

- The thldMonitor MO will act in the role of a “messenger”. After the modified *evsink* determines that the high-level manager needs to be notified about a certain attribute or event, it sends off the corresponding information to the thldMonitor MO, which in turn, sends off an event report concerning the given situation to the high-level manager.
- Moreover, if the high-level manager wants to alter the conditions under which he or she should be notified, the information of the changes can be placed in the thldMonitor MO, from which the modified *evsink* will be able to access the information and apply the required changes.

Figure 5.5 shows the layout of the third approach.

Figure 5.5: The Third Approach of Implementing Hierarchical M2M Communications.

This approach was selected, after evaluating its capabilities and feasibility, as the one to be implemented in this research project for realizing a hierarchical M2M communications scheme using the OSIMIS platform.

5.2.2 Building Up the Selected Implementation

According to the layout of the selected implementation shown in the previous section, there are two entities needed to be built or modified for accomplishing the implementation. This section describes how these two entities can be built.

5.2.2.1 Building the thldMonitor MO

The thldMonitor MO is responsible for informing the high-level manager of a certain situation regarding the low-level agent. Also, the MO is used for reflecting the conditions under which the high-level manager is going to be notified. Therefore, here are the attributes that are required for accomplishing the two tasks just mentioned:

- thldIDescr --- An attribute for storing the description of a certain situation of the low-level agent.
- thldIValue --- An attribute for storing the threshold value under which the high-level manager is going to be notified.

Then, using the 10 steps described in section 4.2.1, a thldMonitor MO can be created and be available for use in the OSIMIS hierarchical M2M communication scheme. Now, the only entity left to be taken care of is the *evsink* program.

5.2.2.2 Modifying the Evsink

Outlining the modification

The original *evsink* program was modified so that it can manipulate the received reports data into useful information and notify the high-level manager according to the given conditions. Basically, the modified *evsink* is cycling through several states or tasks during its execution, and they are shown below:

1. Right after the execution, the *evsink* reads in the designated threshold of the high-level manager (i.e. the attribute "thld1Value" of thldMonitor MO).
2. The *evsink* listens for any event reports emitted from the low-level agent.
3. When the *evsink* receives an event report, it extracts the required information from the report, summarizes the information and stores the result in a data structure.
4. The *evsink* then scans the contents of the data structure and determines whether or not the high-level manager needs to be notified. If yes, the *evsink* puts a description of the given situation to the attribute "thld1Descr" of the thldMonitor MO, and an event report with this description is forwarded to the high-level manager. Otherwise, the *evsink* goes back to the second state of listening for other incoming event reports.

Implementing the modification

For the purpose of simplicity, the initial implementation of the modified *evsink* is only going to monitor the QualityOfService (QoS) alarms. The modified *evsink* keeps track of every attribute causing a QoS alarm. Then, if it finds out that the number of QoS alarms for a certain attribute has crossed the threshold defined by the high-level manager

(i.e: the `thld1Value`), it will inform the `thldMonitor` MO about the name of the attribute. At that point, an event report describing the given situation with the attribute name is forwarded to the high-level manager.

Furthermore, for the same reason, the currently modified *evsink* only allows the high-level manager to modify its designated threshold once (i.e: the `thld1Value`). Extensions to these limitations are described in section 5.3.5.

In order to modify the original *evsink* program to work for the current need, two extra C procedures are added to its original program codes. The first one is `READATTR( )`, whose purpose is reading in the value of the attribute "`thld1Value`" of the `thldMonitor` MO. The second procedure `PROCESS_EVENTS ( )` has a purpose of extracting the required information from an event report when one is received. It determines if the high-level manager needs to be notified about certain situations.

The first procedure `READATTR( )` retrieves the attribute "`thld1Value`" of the `thldMonitor` MO using *mibdump*:

```
if ((commandp = popen("mibdump SMA pegasus -s 1stlevel -f
                      '(objectClass=thldMonitor)'" , "r")) == NULL)
    fprintf(stderr, "Cannot open pipe to store mibdump\n");
else {
    nitems = fread(buf+1, 1, 1024, commandp);
    pclose(commandp);
    buf[nitems] = '\0';
}
```

Then it extracts the value of the attribute from the response:

```
ptr=buf; ptr+=2;           // Locate the attribute thld1Value from
str = strstr(ptr, "thld1Value"); // the response of mibdump

str+=13; tail=str;         // Extracts the value of the attribute
for (int i=0; tail[i]!='\n'; tail++); // thld1Value
rest=tail+1; tail='\0';

thld=strtol(str, &endptr, 10); // Convert the resulting value to long integer type
                                // for using in the PROCESS_EVENTS ( )
```

The second procedure `PROCESS_EVENTS( )` extracts the name of the attribute causing the QoS alarm from the received event report:

```

    head = strstr(str, "thresholdInfo");           //      Locate the thresholdInfo from a report

    head += 15; tail = head;                       //      Extract the name of the attribute
    for (;tail[0]!=' '; tail++);                   //      causing the alarm
    rest=tail+1; tail[0]='\0';
    attr = strdup(head);

```

Next, it adds the information of the extracted attribute into the data structure - in this case, a one-dimensional array or a string. If it is a new attribute, its name and a new counter for it are added into the string:

```

    if ((ptr = strstr(thldrec, attr)) == NULL) {     // Check if the attribute already
        printf("Couldn't find attr %s in record\n", attr); // exists in the string thldrec
        strcat(attr,"0");                          // If no, add the attribute name
        strcat(attr,thldrec);                       // with a new counter of value
        thldrec=strdup(attr);                       // zero into the string
    }

```

Then, whether the attribute is new or not, the counter of the attribute is updated. If the value of its counter crosses the threshold designated by the high-level manager, a description of the given situation along with the attribute name is forwarded to the `thldMonitor MO` using `mset`:

```

    if (((++ptr[0]-48) % threshold) == 0) {         // Update the counter and check
        printf("The number crosses the threshold\n"); // if it crosses the threshold

        setstr=strdup("mset SMA pegasus -c thldMonitor -i thldMonitorId=first -w
                        thld1AimDescr='Threshold crossed -- QoS of '");

        strcat(setstr,attr);                       // Forward the description using
        system(setstr);                             // mset
    }

```

5.3.3 A Working Example

This section presents an example of the method for using the selected implementation of the hierarchical M2M communications scheme. For the purpose of simplicity in this demonstration, there is only one management entity in each level of the hierarchy. Also, since there are only two workstations, namely "pegasus" and "artie", in the network in which this example is tested, one of the stations has to act the role of the low-level agent as well as the high-level manager. Therefore, during this process, the word "pegasus" is referred to as the mid-level manager station, while the "artie(LA)" and "artie (HM)" are referred to as the low-level agent station and the high-level manager station, respectively.

Moreover, although an OSI Agent such as the SMA is a good candidate to be chosen as the agent application on the low-level agent station, a SUN-SNMP agent is chosen as the agent application in this example. The purpose of this choice is to illustrate the way, in which a SNMP agent can be integrated into this hierarchical M2M communications scheme through the use of IQA. Furthermore, in this demonstration, the SNMP attribute "snmpOutTraps" is selected to be the one which the high-level manager wants to monitor. The overall layout of the demonstration is shown in Figure 5.6.

Figure 5.6: The Layout of the Demonstration.

For clarity, the demonstration is divided into three phases which are presented in the following:

Phase one: Setting up

At artie(LA): Activate IQA by typing "iqa" at the system prompt.

At pegasus: Create a Monitor Metric MO at artie(LA) to monitor the attribute "snmpOutTraps" of the SNMP agent by,

```
mcreate IQA artie
-c monitorMetric
-a observedObjectInstance="systemId=artie
                        @systemId=artie@snmpId=NULL"
-a observedAttributeId="snmpOutTraps"
-a granularityPeriod="secs:5"
-a severityIndicatingGaugeThreshold="{Low:42 Switch:On \
                                    High:45 Switch:On }"
```

Then activate SMA by typing "sma" at the system prompt.

At artie(HM): Create a thldMonitor MO instance with a threshold value of two at the mid-level manager by,

```
mcreate SMA pegasus
-c thldMonitor
-i systemId=pegasus@thldMonitorId=first
-w thld1Value=2
```

Next, activate the original version of *evsink* for getting the notifications from the mid-level manager by,

```
evsink SMA pegasus
```

At pegasus: Activate the modified version of *evsink* to retrieve the threshold value for notifying the high-level manager and monitor the event reports from the low-level agent by,

```
mdf_evsink IQA artie
```

Phase two: Sending event reports to mid-level manager

At artie(LA): When the attribute “snmpOutTraps” of the SNMP agent crosses the threshold values defined in its corresponding Monitor Metric MO, the MO sends off an event report on behalf of the attribute to the mid-level manager.

At pegasus: The modified *evsink* receives the report and extracts the attribute name causing the alarm from the report,

```
EventReport - qualityofServiceAlarm
class: monitorMetric
inst: {scannerId=1}
time: 19960623185921.096431--4
report: AlarmInfo - probableCause: thresholdCrossed specificProblems:
(resourceRequestRate) perceivedSeverity: indeterminate thresholdInfo:
severityIndicatingGaugeThreshold 45 monitoredAttributes:
{observedObjectInstance:
systemId=artie@systemId=artie@snmpId=NULL observedAttributeId:
snmpOutTraps}
```


The attribute is snmpOutTraps

Currently, the thldrec string is EMPTY // *thldrec is an string storing*
 // *the event reports information*
 Couldn't find attr snmpOutTraps in thldrec

The modified *evsink* cannot find the information of the attribute “snmpOutTraps” in the *thldrec* string because it is the first QoS alarm report for this attribute. Therefore, it adds a new entry consisting of its name and a new counter to the string. Then it updates the counter and checks to see if the value crosses the threshold value designated by the high-level manager,

The resulting thldrec string is snmpOutTraps1
 The counter value doesn't cross the threshold

Listening for another incoming report....

Since the threshold value of notifying the high-level manager is set to be “two” in this demonstration, no event report about the QoS alarm is forwarded to the high-level manager in this case.

Phase three: Notifying the high-level manager

At artic(LA): When the attribute “snmpOutTraps” of the SNMP agent again crosses the threshold values defined in its corresponding Monitor Metric MO, the MO sends off another event report on behalf of the attribute to the mid-level manager.

At pegasus: The modified *evsink* receives the report and again extracts the attribute name from the report,

EventReport - qualityofServiceAlarm
 class: monitorMetric
 inst: {scannerId=1}

The attribute is snmpOutTraps
Currently, the thldrec string is snmpOutTraps1

Then it finds out that the attribute "snmpOutTraps" is not new. Next, it updates its counter and checks to see if the value crosses the threshold of the high-level manager.

The resulting thldrec string is snmpOutTraps2
The number crosses the threshold

Since this is the second QoS alarm for the attribute "snmpOutTraps", the designated threshold is crossed. Thus, the modified *evsink* puts a description of this situation in the attribute "thld1Descr" of the thldMonitor MO using *mset*.

tailor : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/osimistailor
oidtable : /home/pegasus/carlos/osimis/proxy/iqa/etc/test/oidtable
OIDs 374 : attributes 490 : classes 52

Set Result - id 0x000001, class thldMonitor, instance
{thldMonitorId=first}, time 19960624144453.434002--400

thld1AlmDescr: Threshold crossed -- QoS of snmpOutTraps

Finally, the thldMonitor MO emits an event report with a description of the given situation to the high-level manager.

At artie(HM): The event report from the thldMonitor MO is received through the original version of *evsink* as follows:

EventReport - attributeValueChange
class: thldMonitor
inst: {thldMonitorId=first}
time: 19960623185820.009949--4
report: AttrValChangeInfo: SourceIndicator: managementOperation
{attrId: thld1AlmDescr newValue:
Threshold crossed -- QoS of snmpOutTraps oldValue: }

As a result, from the information given in the event report, the high-level manager knows exactly which attribute in the low-level agent has crossed the designated threshold.

5.3.4 Performance of the Selected Implementation

First, in terms of bandwidth usage, the selected implementation is very efficient, because all the pollings of the agent application are done locally, in a single management entity. In the case of the demonstration, the IQA and the SNMP agent have both been located at *artie(LA)*, so that all the polling that the IQA, as a proxy agent, has done on the SNMP agent has been localized. The transmission between two different management entities occurs only when an event report is forwarded.

Second, the frequency of event report forwarding between management entities can easily be altered. In that case between the high-level manager and the mid-level manager, the frequency of event reports being emitted can be altered by just a single *mset* statement to the attribute “*thldValue*” of the *thldMonitor* MO at the mid-level manager entity.

Similarly, in the case seen between the mid-level manager and the low-level agent, a single *mset* statement to the attribute “*severityIndicatingGaugeThreshold*” of the *Monitor Metric* MO at the low-level agent is the only requirement needed for varying the frequency of having event reports emitted. Even the frequency of the localized polling between the IQA and the SNMP agent can be varied by altering the value of the attribute “*granularityPeriod*” of the same MO.

Third, the selected implementation can logically monitor a virtually unlimited number of attributes simultaneously for a low-level agent. The only thing that limits this number is the capacity of the data structure used for storing the reports information. In the case of the demonstration, a string is used and it can hold information for more than one attribute in the following manner:

The *thldrec* string is "snmpOutTraps4snmpGenErrs2snmpInPkts5....."

The modified *evsink* at the mid-level manager can easily determine which attribute has crossed the designated attribute defined by the high-level manager, using the counter associated with each attribute in the *thldrec* string.

In addition, since it can integrate both OSI and SNMP agents, the whole implementation can be extended to handle a sizable hierarchical M2M communications network, like the one shown in Figure 5.7.

Figure 5.7: A Hierarchical M2M Communications Network.

5.3.5 Recommendations

In order to handle a hierarchical M2M communications network, several modifications of the selected implementation are needed. Each of the modifications is described below:

- *Monitoring for more than one kind of alarm*

Currently, the selected implementation only monitors all the event reports of QoS alarms. In a sizable hierarchical M2M communications network, several other alarms such as "AttributeValueChange" and "ObjectDeletion" may need to be monitored as well.

Also, special arrangements may be required when monitoring certain alarms. For instance, when an alarm of "LinkDown" occurs at the lower-level management entity, the high-level manager may want to be notified immediately, rather than checking whether or not this alarm's number has crossed the designated threshold, and then determining if it needs to notify the high-level manager like other alarms.

- *Using an advanced data structure for storing the reports information*

When monitoring event reports for more than one lower-level management entity, a one-dimensional array is obviously inadequate for storing all the reports information efficiently. Therefore, a more advanced data structure such as a two-dimensional array or a linked list may be needed.

In fact, other than the number of lower-level management entities a mid-level manager wants to monitor, the number of different kinds of alarms that require attention affects the choice of the kind of data structure for storing the reports information.

- *Adding more information in the higher-level event reports*

In the current implementation, only the name of the attribute causing the alarm at the lower-level management entity is sent to the higher-level manager. However, in a hierarchical M2M communications network, there is usually more than one lower-level management entity. Therefore, some sort of identification information is needed when an event report is being forwarded to the higher-level manager. For instance, as recalled from the demonstration shown previously, the higher-level manager receives an event report about the attribute "snmpOutTraps" of the lower-level agent as follows:

```
EventReport - attributeValueChange
class: thldMonitor
inst: {thldMonitorId=first}
time: 19960623185820.009949--4
report: AttrValChangeInfo: SourceIndicator: managementOperation
{attrId: thld1AlmDescr newValue:
    Threshold crossed -- QoS of snmpOutTraps oldValue: }
```

In this case, simply stating the name of the attribute causing the alarm in the event report is adequate, because there is only a lower-level management entity in the demonstration. However, in a hierarchical M2M communications network, an event report sending to a higher-level manager should have sufficient information for the high-level manager to know exactly which attribute of which lower-level management entity is causing the alarm. For instance,

```
EventReport - attributeValueChange
class: thldMonitor
inst: {thldMonitorId=first}
time: 19960623185820.009949--4
report: AttrValChangeInfo: SourceIndicator: managementOperation
{attrId: thld1AlmDescr newValue: Threshold crossed -- QoS Alarm
    systemId=artie@systemId=artie@snmpId=NULL@attr=snmpOutTraps oldValue: }
```

from this event report, the high-level manager can know exactly that it is the attribute "snmpOutTraps" of the SNMP agent at the station "artie" causing the alarm.

- *Notifying the high-level manager anytime as needed*

Currently, the modified *evsink* only allows the high-level manager to alter the designated threshold once. However, in a real world situation, there are many circumstances that a high-level manager needs to alter the threshold often, in order to adapt any arising situations of the monitored sub-network.

The C++ procedure `READATTR( )` is the one responsible for getting the threshold value designated by the high-level manager. In the current implementation, the procedure `READATTR( )` gets executed only when the modified *evsink* starts to monitor the event reports. This means that once the modified *evsink* obtains the threshold value from the procedure at the beginning stage, any updates to the threshold will not be noticed by the modified *evsink*.

Therefore, in order for the modified *evsink* to obtain the updated threshold value, the procedure `READATTR( )` has to be executed from time to time - before the other procedure `PROCESS_EVENTS( )` processes the information of the incoming event reports. The following procedure "do_wait", inside the modified *evsink* program, is responsible for waiting for the incoming event reports:

```
static int do_wait (int msd, MSAPIndication* mi)
{
    char * inst;
    PE pdu;

    M_WaitReqAux(msd, NOTOK, mi, &pdu);
ADD    readattr( );

    #if defined(DEBUG)
        if (pduDump != (FILE *) 0)
            ppMSAPIndication(mi);
    #endif
    .
    .
    switch (mi->mi_type) {
```

```

        .
        .
        case MI_EVENT_REP: {
        .
        .
            process_events(reportval);
        }
        .
    }

```

The procedure `M_WaitReqAux( )` determines if there is an event report available. Then the rest of `do_wait( )` performs the error checking and finds out the type of the incoming event report. Therefore, placing the procedure `READATTR( )` just below `M_WaitReqAux( )` allows the threshold value of the high-level manager to be updated, right after an incoming event report is available and before the information of the report is processed by `PROCESS_EVENT( )`.

Moreover, instead of having only one data structure (i.e.: the string "thldrec") in `PROCESS_EVENT( )`, which stores the information of attributes causing the alarms, another data structure is needed for tracking each attribute against the high-level manager threshold value. Unlike the first data structure which accumulates all the information of each attribute causing the alarms, the second one resets the counter of a certain attribute, after it crosses the high-level manager threshold and an event report is forwarded. An example of how the second data structure works is shown in the following:

e.g.: 1) At a certain time,

- the high-level manager threshold is 3
- the contents of the **first string** is "snmpOutTraps2mailObj1"
- the contents of the **second string** is "snmpOutTraps2mailObj1"

2) When an event report of "snmpOutTraps" is received,

- the high-level manager threshold is 3
- the contents of the **first string** is updated to "snmpOutTraps3mailObj1"
- the contents of the **second string** is updated to "snmpOutTraps3mailObj1"

=> an event report is forwarded to the high-level manager about the
=> attribute "snmpOutTraps"

- the counter of "snmpOutTraps" is then reset and the contents of the **second string** becomes "snmpOutTraps0mailObj1"

3) After the high-level manager is notified about the attribute "snmpOutTraps", the high-level manager decides to monitor this attribute more closely by modifying the threshold from 3 to 1. Then, when another event report of "snmpOutTraps" is received,

- the high-level manager threshold is changed to 1.
- the contents of the **first string** is updated to "snmpOutTraps4mailObj1"
- the contents of the **second string** is updated to "snmpOutTraps1mailObj1"
- =>** the high-level manager is notified again about the "snmpOutTraps"
- the counter of "snmpOutTraps" is then reset again and the contents of the **second string** becomes "snmpOutTraps0mailObj1"

The mid-level manager, in using two data structures to keep track of the event reports information this way, can conserve the overall information of alarms received, while the high-level manager can adapt to any situations arising in the management network.

CHAPTER 6

CONCLUSIONS

6.1 Summary of Results

The objectives of this work were: to explore OSIMIS for providing CMIP services; and to achieve a hierarchical manager-to-manager communications scheme using OSIMIS. Therefore, this research has the following major goals in using the OSIMIS platform:

- To realize an OSI network management system with both an OSI agent and a SNMP agent.
- To provide event report forwarding between management entities.
- To provide thresholding capabilities to both an OSI agent and a SNMP agent.
- To implement a hierarchical manager-to-manager communications scheme by integrating the OSIMIS features.

Realization of an OSI Network Management System

In a network management system, the first requirement is an agent application. In OSIMIS, two OSI agents are already provided. Using one of them, namely SMA and the support of the OSIMIS GMS, a network manager only needs to worry about the codes for interfacing a real resource to its corresponding MO, leaving everything else such as the interfacing between the MO and the agent, and the handle of the communications protocol (CMIP) to OSIMIS. Also, most of the generic manager applications provided by OSIMIS

are flexible and very easy to use. Once a MO is already added to the MIB, the manager applications can access its information with only one single command line.

However, the procedures needed for adding a new MO class to the OSIMIS platform are quite tedious. The same problem also occurs when modifying the information of an existing MO class. A stand-alone shell program integrating all the procedures and tools involved in the addition and the modification of MO classes is very much needed, for implementors who want to build a network management system of any useful size using OSIMIS.

For working with a SNMP agent, OSIMIS provides a generic application gateway, namely IQA, which can integrate any SNMPv1 agents into the OSIMIS platform. Once the SNMP agent is integrated, all the MIB information of the agent can be easily accessed like an OSI one using the generic manager applications. Also, accessing the SNMP agent through the OSIMIS IQA can give the network manager the OSI scoping and the filtering capabilities when managing the SNMP agent. In addition, the traps emitted from the agent are automatically translated to event reports by the IQA and forwarded to the manager application.

However, using the IQA has a similar problem as the one in SMA, which is the tedious procedures of integrating a SNMP agent into the OSIMIS platform. Although the demonstration of the IQA described in this work has only used a predefined OSI representation of the SNMP MIB-II, the reader can notice that from [6] and [7], it is very time-consuming, if a network manager needs to integrate several SNMP agents with

different MIBs into OSIMIS. Therefore, a separate shell program is also needed in this case.

Event Report Forwarding Between Management Entities

Using the event sink application *evsink* provided by OSIMIS, it is simple for a network manager to monitor the events occurred on an agent. Moreover, the network manager is not limited to activate just one *evsink* at one time. Several *evsinks* can be activated simultaneously on a management station for monitoring a number of agents, or a number of different MOs on a single agent at the same time. In addition, the *evsink* automatically creates an EFD MO at the corresponding agent to allow the network manager to select desirable event reports. This capability has a great effect in controlling the flow of management information and it is extremely important in bandwidth-constrained environments.

The *evsink* is supposed to be able to select event reports based on the type of the event and the attribute causing the event. But only the former were able to be accomplished in this work. The latter needs access to a special syntax table of the attributes whose location within the OSIMIS platform is currently unknown.

Thresholding Capabilities for Agent Application

On the OSIMIS platform, the only thing a network manager needs to do for monitoring any MOs in an agent application, is to create a Monitor Metric MO for each designated MO. The Monitor Metric MO scans the designated MO and sends off event reports on behalf of it when an event occurs. In OSIMIS, a network manager can activate or suspend monitoring for any MOs anytime with just one single command line. For an

OSI agent, using a Monitor Metric MO to provide a designated MO with thresholding capabilities is much more simpler and flexible than hard-wiring the thresholds in the corresponding MO class definitions.

For a SNMP agent, the ability to provide thresholding capabilities to SNMP objects is particularly useful because there is no SNMP specialized way to provide that in the SNMP context. In OSIMIS, a network manager can give thresholding capabilities to any SNMP object by just creating a Monitor Metric MO for it through the IQA. Similar to an OSI agent, the network manger can suspend or modify the thresholds any time by only one single command line. This ability has a great effect in integrating a SNMP agent into an OSI network management environment.

Implementation of a Hierarchical Manager-To-Manager Communications Scheme

An initial implementation was achieved in this work. It can currently support a hierarchy consisting of three management levels. In the current implementation, the low-level agent application emits all the event reports to the mid-level manager, which manipulates the reports data into useful information, stores the information in a data structure, and notifies the high-level manager when necessary. Also, the high-level manager can alter the condition of when to be notified by the mid-level manager. In addition, using the OSIMIS IQA, not only OSI agents can be monitored under this implementation, but also SNMP agents as well.

Furthermore, the initial implementation can be extended to support a hierarchical manager-to-manager communications network with more management levels, each of which consists of any number of management entities. This can be achieved by modifying

the data structure for storing the events information, and some of the codes in the modified version of the OSIMIS *evsink* used in the implementation.

6.2 Suggestions for Future Work

There is no doubt that OSIMIS is a very powerful platform for implementing OSI network management. The demonstrations of the OSIMIS applications presented in this work can be treated as a guide for implementing network management systems of any size using the OSIMIS platform. The hierarchical manager-to-manager communications network based on the implemented manager-to-manager communications scheme described in this work is one of the examples. On the other hand, as discussed earlier, shell programs are definitely needed for building any network management systems using OSIMIS.

Also, the current major driving force behind OSI network management is the deployment of the Telecommunications Management Network (TMN). The overall vision of TMN is to use independent management networks to manage the telecommunications networks in today's multivendor environment through standardized interfaces[9], and OSI network management technology was selected as the basis for the TMN.

The bulk of TMN's standards deal with interfaces because the manner in which management systems interact is governed by interfaces. The Q3 interface is the flagship interface for connecting management systems in TMN, and the Qx is the Q3 interface's underdeveloped brother which is similar to Q3 but with less functionality[9]. For

integrating existing networks (or non-TMN systems) into TMN, a Q-Adapter is used to convert their existing interfaces to a Q3.

On the current OSIMIS platform, the interface realized by the IQA is only a Qx rather than a Q3. The reason is that, the information model produced by the automatic translation between SNMPv1 and CMIS/P, according to the NMF IIMC rules[17] that the IQA uses, is syntactically at least, different from the relevant CCITT one. For example, translating the SNMP ATM MIB gives something syntactically very different from the GDMO ATM one, although semantically these are similar. Therefore, a mediation function is needed to convert between the two models, so that the OSIMIS IQA can present a Q3 interface for non-TMN systems and integrate them into the TMN environment.

REFERENCES

- [1] Stallings, W., SNMP, SNMPv2 and CMIP: The Practical Guide to Network Management Standards, Addison Wesley, 1993
- [2] Pavlou, G., Knight, G., McCarthy, K., and Bhatti, S., "The OSIMIS Platform: Making OSI Management Simple", Proceedings of the Fourth International Symposium on Integrated Network Management, USA, 1995, p. 480 - 498.
- [3] Pavlou, G., "The OSI Management Information Service User's Manual Version 1.0, for System Version 3.0", London, England, 1993
- [4] Pavlou, G., J. Cowan, and J. Crowcroft, "A Generic Management Information Base Browser", Upper Layer Protocols, Architectures and Applications: Proc. IFIP TC6/WG6.5, ed. B. Plattner, pp. 221-232, North-Holland, Amsterdam, 1992.
- [5] Pavlou, G., K. McCarthy, G. Mykoniatis, and J. Sanchez, "Intelligent Remote Monitoring", Bringing Telecommunications Services to the People - IS&N 95, ed. A. Clarke, M. Campolargo and N. Karatzas, pp. 430-444, Springer Verlag, 1995.
- [6] Gorman, B., "Using the OSIMIS Internet Q Adaptor (IQA) Software, Version 1.0", email: bernard_gorman@craycom.co.uk, Advanced Technology Group, Cray Communications, Hertfordshire, United Kingdom, January, 1996
- [7] Pavlou, G., McCarthy, K., and Bhatti, S., "Exploiting the Power of OSI Management for the Control of the SNMP-Capable Resources Using Generic Application Level Gateways", Proceedings of the Fourth International Symposium on Integrated Network Management, USA, 1995

- [8] Pavlou, G., "Implementing OSI Management", Tutorial presented at the 3rd IFIP/IEEE ISINM, San Francisco, April, 1993
- [9] Glitho, R. H., and Hayes, S., "Telecommunications Management Network: Vision vs. Reality", *IEEE Communications Magazine*, March, 1995, p. 47 - 52.
- [10] Bichler, M., "Network Management Systems", Busn 470 Individual Paper, <http://www.cc.ndsu.nodak.edu/~draper/470/projects/mb.html>, 1995
- [11] ANZ GOSIP (The Australian and New Zealand Government Open Systems Interconnection Profile) Network Management Guidelines, August, 1995
- [12] ITU X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, July, 1991
- [13] Object Management Group, The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, Draft, December, 1991
- [14] Network Management Forum, Open Management Interoperability Point (OMNIPoint) 1.0, 1993
- [15] Rose, M. T., Onions, J. P., and Robbins, C. J., "The ISO Development Environment User's Manual Version 7.0", PSI Inc./ X-Tel Services Ltd., July, 1991
- [16] X/Open, OSI-Abstract-Data Manipulation and Management Protocols Specifications, January, 1992
- [17] Lee, L., Forum 026 - Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Issue 1.0, October, 1993

END

1 8 0 6 9 7

FIN