MODELING AND IMPLEMENTATION OF AN ATM-BASED LAN NETWORK MANAGEMENT SYSTEM USING OBJECT MODELING TECHNIQUE (OMT)

(16)

BY

THANH TIN DANG

A Thesis
Submitted to the Faculty of Graduate Studies
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

ISBN 0-612-13058-4

THE UNIVERSITY OF MANITOBA

FACUTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

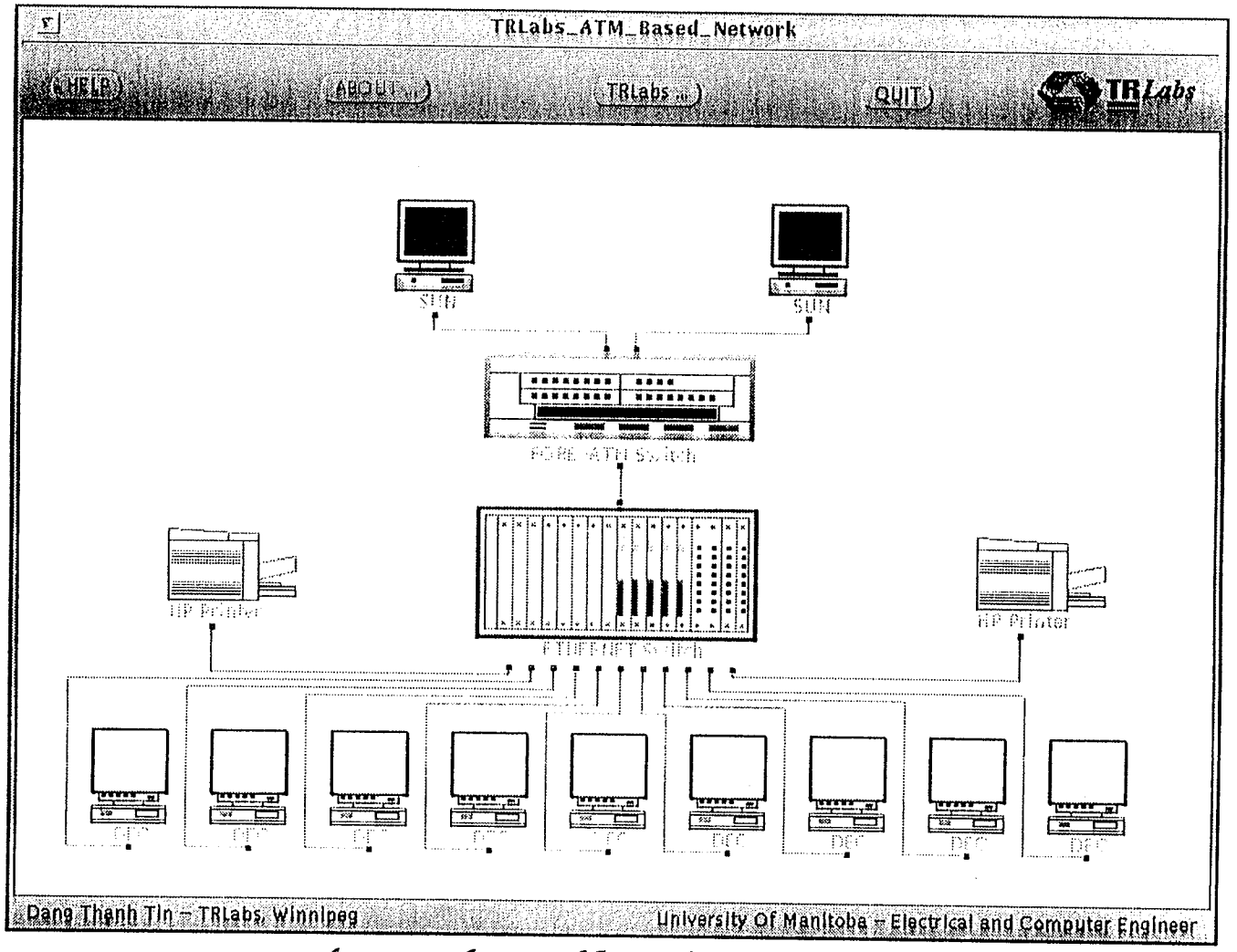
MODELING AND IMPLEMENTATION OF AN ATM-BASED LAN NETWORK MANAGEMENT SYSTEM USING OBJECT MODELING TECHNIQUE (OMT)

BY

THANH TIN DANG

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

MASTER OF SCIENCE



Network Traffic Monitor Desktop

I hereby declare that I am the sole author of this thesis, titled "***Modeling and Implementation of an ATM-Based LAN Network Management System Using Object Modeling Technique OMT***"

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

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

Tin Thanh Dang, 1996.

ABSTRACT

Network management has been an important issue in computer networking operation and maintenance. The rapid growth of computer and data networking technology and human needs for information processing has resulted in increasing complex networks and associated difficulties in network management. In the last decade, the newly developed ATM (Asynchronous Transfer Mode) technology has become a mainstream component telecommunications. ATM technology provides both the fastest and most comprehensive networking, being suitable for all types of data, from information data to speech and images. As a result, the management of ATM network has become a central problem for network researchers.

This thesis investigates Object Oriented Technology(OOT) as an approach for designing and analysing software applications for an ATM-Based LAN network. Object Oriented Technology models a software application as a system consisting of objects which interact with each other by passing messages to perform system operations. Of the many OOT methods that have been developed by different software researchers, the method extensively studied and applied in this thesis is the Object Modelling Technique (OMT), developed by James Rumbaugh.

A software development tool, StP (Software through Pictures), based on the OMT method, is also used in developing the network management software, which includes a graphical user interface. The software is implemented in C++ and Tcl based languages, and the tool "scotty", a Tcl-based interpreter with extensions to obtain information about

the network. The network considered for management by this thesis is a FORE-ATM LAN located at **TRL**abs (**T**elecommunication **R**esearch **L**aboratories) - Winnipeg. The software is a graphical monitor of the network system which consists of the information on the number of cells/packets entering/leaving as well as some important traffic statistics for each of the network devices.

The thesis concludes that OMT is an efficient and advantageous approach for software development. As the system management for the newly developed ATM networking technology is a very important concern, the study of ATM networking should be explored, not only locally but globally, to develop much more sophisticated ATM management systems.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to the following, without them, this work would not have been possible.

My advisors, Dr. R. D. McLeod, for technical advice, encouragement, help and suggestions in completion of the thesis, and Dr. D. Blight, for his technical assistance and clarifications of problems that I have encountered during the progress of the thesis.

TRLabs (**T**elecommunication **R**esearch **L**aboratories) has provided funding and all facilities for my studies.

My colleagues and staff members at **TRL**abs and at the University of Manitoba.

Lastly and specially, I owe every thing to my parents for moral support and encouragement during my study.

This work is dedicated to my parents!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

PREFACE

This research was sponsored by Telecommunication Research Laboratories (**TRLabs**), Winnipeg, Manitoba. The essential goal of the research was to investigate the application of the Object Oriented Technologies (OOT) in software development for a telecommunication related project.

The research started with the study and investigation of Object Oriented Techniques. Two OOTs were also included in study: the Object Modeling Technique developed by James Rumbaugh and Object Oriented Design and Analysis developed by Grady Booch. The OMT methodology was selected as the approach for software development in the remainder of this study. Along with the study of OOT, other areas in telecommunication were also studied with special concentration on the Asynchronous Transfer Mode protocol for communication.

The main direction of the thesis was initially to design and develop software for an ATM switch using the OMT methodology. The research direction was later changed to network management utilizing the facilities available at **TRLabs** and the interest of myself and my advisors. The final objection of the research was to design and implement a software application using the OMT methodology for ATM-Based LAN traffic monitoring. The research incorporated network management concepts and SNMP (Simple Network Management Protocol). Because of time constraints, the system designed and implemented was a simple centralized system primarily illustrating network traffic monitoring func-

tions.

The thesis document consists of a number of Chapters which cover several inter-related topics: the Object Modelling Technique (OMT) (Chapter 2) and the software package, Software through Pictures/Object Modelling Technique (StP/OMT) (Chapter 3), Asynchronous Transfer Mode (ATM) concepts (Chapter 4), Network Management concepts and SNMP (Chapter 5), and the Management Information Base (MIB) for ATM devices (Chapter 6). The remainder of the thesis presents the design, implementation and features of the network traffic monitor. This is followed by the conclusions.

In the future, a study following up this research is proposed to continue investigating the Object Modeling Technique methodology for system design in the ATM network management area. This should be extended to include the design and implementation of a distributed network management system for a larger ATM network through the use of Intelligent Agents.

CHAPTER 1
INTRODUCTION

In recent years, computer networks have been growing very rapidly in both size and complexity in response to the increased demands of transporting data from one place to another. The data required to be transmitted is not only computer-generated, such as computer images and text data, but also analog data, such as voice information, and it would be rather inefficient and redundant to have different network protocols for handling each of the type of data.

In the late 80's, with the development of Asynchronous Transfer Mode (ATM) broadband technologies, different types of information could be digitized and transmitted over a single protocol.

Along with the improvement of technology and information processing and transportation, the issue of network operation and maintenance plays a very important role. A network management system is usually software implementation. The management system monitors the network operations and reports the information about the physical and operational status of the network as requested by the system administrator/manager. It also allows authorized persons to set or reconfigure the system as desired for better performance.

In this thesis, the object oriented approach for software development is investigated for designing and analysing software applications in telecommunication. This thesis proposes and develops an object oriented approach for software to analyze and design a network

management system for an ATM-based LAN. The functions of this software application are to monitor the traffic of such a system. Various information such as the number of cells/packets entering and leaving the device interfaces, and some important statistical information of individual devices of the system are reported textually and graphically. The configuration of the devices can also be set manually by the system administrator according to the accessibility of the information on the devices.

The following sections of this chapter briefly introduce the topics and concepts that will be presented in further detail in the following chapters of the thesis. A summary of each chapter is also included in this chapter.

1.1 OBJECT ORIENTED TECHNOLOGY

In software production, it takes many steps through software development cycles to complete a product. It is important to assure that each step is well studied and designed to result in a well-performing product. Procedural approaches for software development is effective and efficient for reasonably large and complex software systems, but becomes impractical and inefficient in both time and labour for a very large and complex systems. The birth of a new approach, known as Object Oriented Technology (OOT), has been an evolutionary step in software development research and has replaced the older, procedural approach for many applications. The object oriented approach changes the way a designer views the software system. The system being designed is viewed as an organization of objects [from which the name Object Oriented comes from] that interacts among each other to perform system operations by passing messages between them. It forces the

designer to look at every aspect of the system with an object oriented view, with respect to the object's interaction and object's behaviour. Object Oriented Technology, OOT, provides a layout of the system that can be more effectively understood, because the system can be easily visualized throughout the development cycle. In addition, OOT provides many other advantages in the software development process, such as encouraging code reuse, ease of analysis, reduced development cycles and simplified system integration. OOT also supports autonomous software components that can be independently developed. In Chapters Two and Three, an OOT methodology, namely the Object Modeling Technology (OMT) developed by James Rumbaugh, will be discussed and presented in detail to provide insight into OOT.

1.2 ASYNCHRONOUS TRANSFER MODE (ATM) NETWORKS

Asynchronous Transfer Mode (ATM) is an efficient high bandwidth, low delay switching and multiplex broadband communication technology [Minoli D., Vitella M. 94]. ATM is at the heart of telecommunication technology which can deliver important advantages over existing networks operating over different scales, such as LAN and MAN technologies. ATM provides scalable bandwidth and a guaranteed of quality of service (QoS). However for the trade-offs associated with the advantages of ATM, it is a very complex technology. In ATM networks, the data are segmented into cells of 48 bytes in size. Each cell is generated with a a 5-byte cell header (each cell is 53-bytes long) that contains information about the data before it is transmitted to its destination. The cell headers are stripped and the cells are reassembled before the receiver obtains the information. ATM is

a connection oriented network and a virtual circuit needs to be set up first in order for data to be sent to its destination. The ATM circuit is made up of a combination of virtual paths and virtual channels. The concepts of virtual path and virtual channel and a detail discussion of ATM networks can be found in Chapter Four.

1.3 NETWORK MANAGEMENT SYSTEMS

Increases in network size and complexity have made it impossible for humans to manage these systems by themselves, necessitating an automated network management tool. The functions of a management system includes Fault Management, Account Management, Configuration Management, Performance Management, and Security Management [Stalling W. 93]. Each of these management areas are discussed in further detail in Chapter Five. As well, the performance of the system as well as the efficiency of resource utilization are primarily controlled by the management system. The behaviour of the system can be studied by analysing the information obtained from the management system.

1.4 PROPOSED NETWORK MANAGEMENT SYSTEM

This thesis proposes a design and implementation of a ATM- based LAN network management system. The approach used to design and analyze the software is the Object Oriented Technique. The system has a graphical user interface that allows the user to obtain necessary information about the current status of a real physical operational system. The information obtained, such as the amount of traffic, the traffic rate and the error transmis-

sion rate entering and leaving an individual device of the network, can be displayed in both textual and graphical formats.

1.5 STRUCTURE OF REPORT

This report is divided into several chapters.

Chapter 2:

Discusses in detail the object oriented technology approach used, which is the Object Modeling Technology (OMT) developed by Dr. James Rumbaugh. It also includes some suggestion on how the object models can be coded with the C++ language.

Chapter 3:

Introduces the software package StP/OMT (Software through Pictures/Object Modeling Technique), which is the tool used to design and analyze the software system. An example of an object model is given to demonstrate how the object model can be constructed and how C++ code of the object model can be generated with this software. The chapter concludes with some thoughts about StP/OMT.

Chapter 4:

Introduces ATM technology, discusses the concepts behind ATM, and shows how ATM is an advantageous technology for today and future telecommunications.

Chapter 5:

Introduces the concepts of network management. The SNMP (Simple Network Management Protocol) is addressed in detail. The chapter also includes MIB tree and definitions of the syntax of managed objects.

Chapter 6:

Introduces the various MIB trees developed by different vendors for the ATM protocol. This chapter also includes the MIB sub-tree mib-2. As well, it describes how the ATM cells are multiplexed in the ATM and AAL layers.

Chapter 7:

Introduces the network considered for development of the network traffic monitor. It also describes the design and analysis approach for the system, including three models of the OMT approach for the software system. As well, the chapter also discusses the different languages used to implement the software. It also includes an introduction to the "Scotty" tool used for retrieving network information from the devices.

Chapter 8:

Discusses the structure, behaviour and limitations of the developed network traffic monitor.

Chapter 9:

Concludes the thesis with some final thoughts and comments on the approach used for designing and analysing the network management software as well as a proposal for further research in this area.

CHAPTER 2
OBJECT MODELING TECHNIQUE (OMT)

Object Oriented Technology (OOT) has been studied and developed by software development scientists for many years. Many versions of OOT currently exist, such as Object Oriented Analysis and Design [developed by Booch], and Object Modeling Technique [developed by Rumbaugh]. OMT is the candidate OOT chosen for the research in this thesis. OMT was utilized to design and develop a software application for network management. It is a modeling concept in which the designer thinks of the system being designed as an organization of objects. The object is the fundamental structure of the system, containing both the static data structure and the dynamic behaviour in a single entity. The operations of the system are performed by the changes in dynamic behaviour of each object and the messages passing between objects within the system. The main motivation behind OMT is its organization and capacity for early verification. Attention to detail and modeling greatly aids detection and correction of errors early in the design cycle, resulting in more manageable software development downstream in terms of reliability and manageability. An object in OMT is generally associated with the following characteristics [Rumbaugh J. 91]:

OBJECT IDENTITY: Each object in the system must be uniquely defined, and the quality that distinguishes the object from other objects in the system is the identity of the object. An example of identity is that a particular host can be uniquely defined by the name of that host.

OBJECT CLASSIFICATION: Objects that share common data structure (attributes) and behaviour (operations) are grouped into a class. An object can be defined from a class by specifying the value(s) for the attribute(s) of the class. This specification is called object instantiation. The object is the instance of the class.

POLYMORPHISM: The concept of polymorphism arises when different classes have operations with the same operation name, but the operations behave differently from one class to another. An example of polymorphism can be the operation of "*move*" for the classes "*car*" and "*plane*". Both classes have the same operation name, "*move*", but the ways that the objects from these classes perform the operation are different.

INHERITANCE: The inheritance reflects the origin of the class. If class A has been derived from a class B, then class A is called "sub-class" of class B or class B is called "super-class" of class A. The sub-class inherits all the attributes and operations from its super-class, with some additional attributes, operations and modifications to the operations of the superclass. An example of inheritance is the derivation of class "*ATM-Host*" from class "*Host*". The class "*Host*" defines a class with all the attributes and operations of a general host. The class "*ATM-Host*" is redefined from class "*Host*" with some addition attributes, functions required for an ATM-Network.

The software system designed by OMT methodology consists of three major models to describe a functional system: Object Model, Dynamic Model and Functional Model. The relationships between the three models in OMT are presented in **Figure** 2.1.

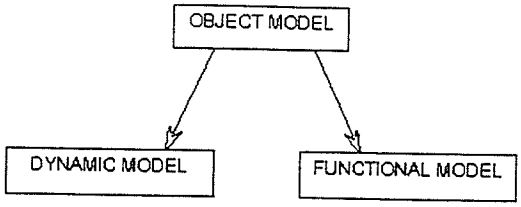


Figure 2.1 Three models of OMT.

In the **Figure** above, the Dynamic Model and Functional Model are shown to be derived from the Object Model. In the next three sections of this chapter, each of the three models is discussed in detail, with respect to what these models offer and how they are presented.

2.1 THE OBJECT MODEL

The Object Model describes the static structure of the object such as the identity, the attributes, the operations, and the relationships of the object with other objects in the system. In OMT, the object model is graphically presented, and the model is called the object diagram. The main component of the object model is its class, a common representation of objects that share the same properties and characteristics. The graphical representation of a class is a box with 3 sections as shown in **Figure** 2.2.

The components of the class are class name, class attribute(s) list and class operation(s) list. The name identifies the class so as to distinguish it from other classes. The attributes of a class are its pure values, and the operations are the functions (procedures) applied to or by the other objects in object model.

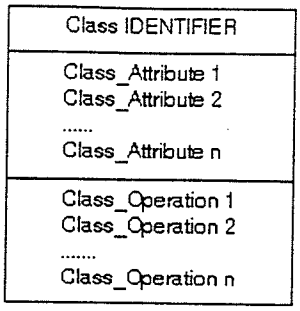


Figure 2.2 Graphical representation of a class.

A class can have an object instance which belongs to the class and has its attributes physically defined, as assigned values. In **Figure** 2.3, the object instantiated from the class "*Host*" has the attributes "*Name*" and "*Address*" defined as specific values "*truffle*" and "*192.168.1.21*".

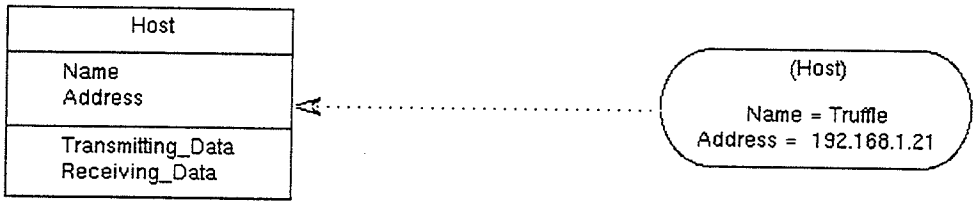


Figure 2.3 Example of instantiation.

2.1.1 Associations and Links

One kind of relationship between the object instances in a system is the link, which is the physical or conceptional connection between these object instances. An association is a relationship between classes which consists a group of links with common structure and

semantics. Each association has a name that describes the relationship explicitly as well as how the classes are semantically related. An association is modeled as a line connected between the two classes and labelled with a name. **Figure** 2.4 gives an example of an association between class "*Manager_Entity*" and class "*Agent_Entity*" in a network management system. The association has a name of "*manages*" that describes how the class "*Manager_Entity*" manages the class "*Agent_Entity*".

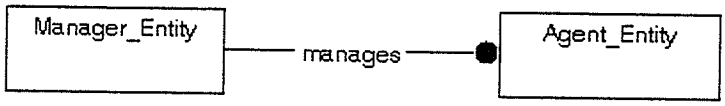


Figure 2.4 Example of association.

Multiplicity is an optional number attached to each end of an association connection to specify how many instances of one class may relate to one instance of the associated class. In OMT, many types of multiplicity are used to model the association. The notation for each type of the multiplicity is shown in **Figure** 2.5. In the previous **Figure** 2.4, the black dot on the association end of the "*Agent_Entity*" class indicates that one object instance of "*Manager_Entity*" class can associate with many object instances of class "*Agent_Entity*", that is, one management system can manage many agents in the network system. A class may have more than one association with other classes to describe the multi-relationship between them.

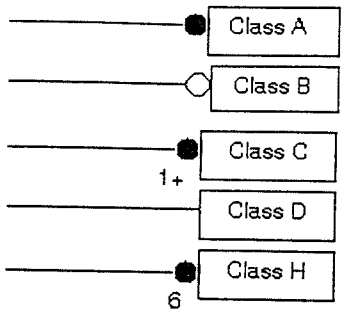


Figure 2.5 Notation for multiplicity.
[Class A: Many (zero or more), Class B: Optional (zero or one)]
[Class C: One or more, Class D: Exactly one, Class H: Numerically specified]

An association may have its own attribute(s) and operation(s) which describe the relationship more explicitly. This kind of association can be modeled as a class, with attribute(s) and operation(s) that fully describe the association. The attributes that are included in the association can not be attached to the class objects at the ends of the association without information being lost during the operation. **Figure** 2.6 gives an example of an association modeled as a class. The association is between classes "*Manager_Entity*" and class "*Agent_Entity*" and is modeled as the class, "*manages*". The association class is modeled with two attributes, "*Level managing*" and "*Traffic amount*".

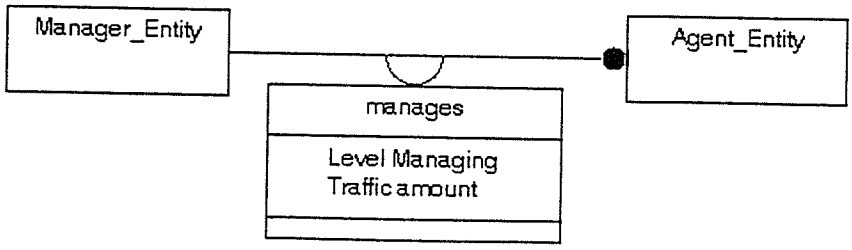


Figure 2.6 Example of modeling association as a class.

2.1.2 ROLES

Each class may have a specific role in the association with other classes in the object model. The role of each class in the association is modeled with a name in the end of association to uniquely identify the role of the class in that association with other. The example of role name is shown in the **Figure** 2.7, in which the "*Manager_Entity*" that manages all the devices in the network system has a role of manager. All the devices of the network which are modeled as class "*Agent_Entity*" and have a role of client. The role must be unique to distinguish the object among other objects. The role is helpful in distinguishing classes with multi-association between them.

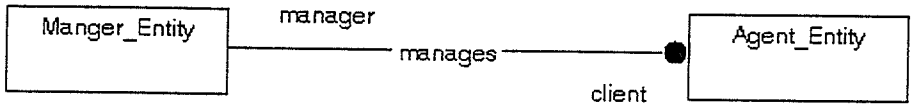


Figure 2.7 Example of role.

2.1.3 SELF-ASSOCIATION AND ORDER

A class may have an association with itself and its association may be multiple. In the "many" association, the objects can be explicitly "ordered". **Figure** 2.8 describes an example of a file system in a computer to demonstrate the reflection and order concepts of object modeling. The file system indicates that a directory can contain many subdirectories and many files (0 or more files in a directory). The directory is therefore modeled as a self-association which may contain many directories. Both the subdirectories and the contained files can be arranged within the directory in order. The files can be ordered in terms of file name, file name extension, size of file or time of modification. The order is modeled by

"{ordered}" next to the multiplicity dot.

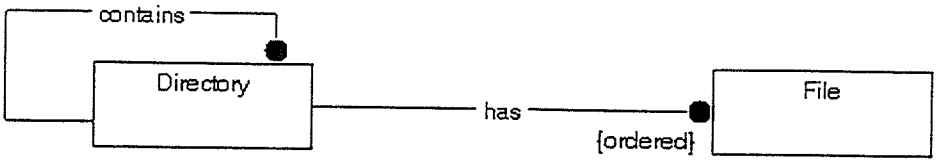


Figure 2.8 Example of order and self-association.

2.1.4 QUALIFIER

Model association with a "qualifier" can reduce the effective multiplicity. The qualifier is an attribute that refers to the "many" side of the association. For example, the association between the classes "*Manager_Entity*" and *"Agent_Entity"*, it is a one-to-many association. However, the objects in the many side can be uniquely distinguished by the attributes, "*address*". The introduction of the qualifier of "address" thus reduces the one-to-many association to a one-to-one association. The **Figure** 2.9 is example of qualifier in an association as indicated above.

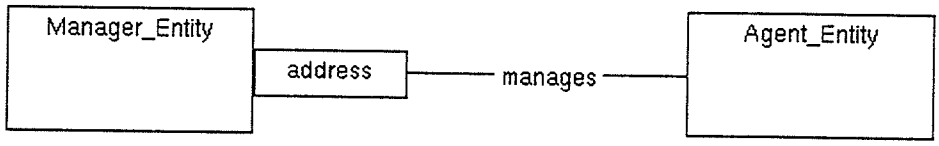


Figure 2.9 Example of a qualifier.

2.1.5 MULTIPLE ASSOCIATION

An association can be modeled between more than two classes. It is recommended that association of more than three classes are avoided because of the complexity. For a multiple association between more than two classes, a list of variables is needed to specify the combination of instances of classes participating in the association and uniquely identify the link between them. These variables are known as the candidate keys. **Figure** 2.10 presents an example of ternary association (association between three classes), with a list of candidate keys to uniquely define a single link. The unique link between three classes "*Project*", "*Language*" and "*Student*" can be defined by the candidate keys {student name, project title, language used for the project}.

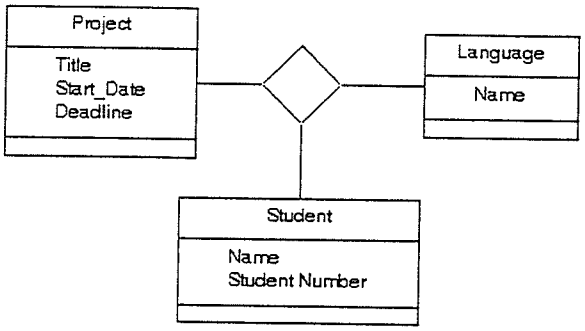


Figure 2.10 Example of ternary association.

2.1.6 AGGREGATION

A special form of association that describes an object as being a part or component of another object is known as an aggregation relationship. The notation for the aggregation relationship is shown in the **Figure** 2.11, which describes the class "*Computer_System*", as the assembly with the class "*Monitor*", and class "*Key_Board*" as its components, and the

class "*Key_Board*" with the class "*Key_Button*" as its component. The concept of multiplicity can also be applied in the case of aggregation, because a object class may have many of same components. For example, an object instance of class "*Key_Board*" may have many object instances of class "*Key_Button*" as components. One of the properties expressed through the concept of aggregation is propagation, that is where an operation is propagated automatically to some of its component(s) when it is applied to the assembly.

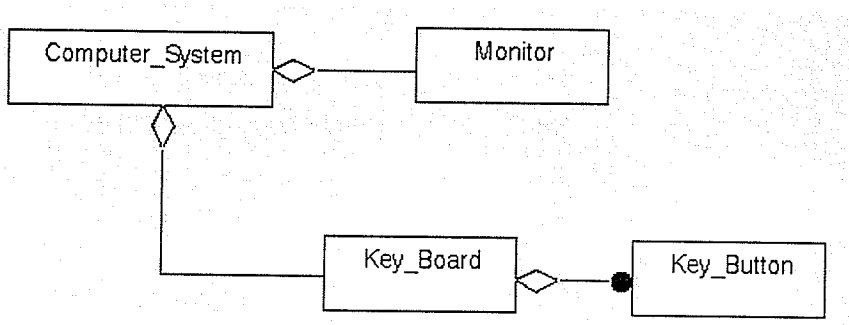


Figure 2.11 Example of aggregation.

An example of operation propagation is that when a key button (operation on class "Key_Button") is pressed, this event will affect the key-board (the class "Key_Board" is affected) and propagate to the computer, such that key press may affect the behaviour of the class "*Computer_System*".

2.1.7 GENERALIZATION

As briefly mentioned at the beginning of this chapter, generalization is a relationship between a class (super-class) and one or more its refined versions (sub-class). The sub-class

not only inherits the properties and characteristics from its super-class, but also can add its own specific features. **Figure** 2.12 gives an example of generalization from one class to its refined version and how the generalization is modeled in OMT. Classes can be refined in many levels and at each levels, a new version of the class is defined. The generalization chain is called the generalization hierachy tree or inheritance hierachy tree. A class can be derived from more than one class, a situation known as multiple inheritance. An example of inheritance is demonstrated in **Figure** 2.12: the class "*Host*" is the base class with the general attributes and operations for all of the network devices. More generalized classes are refined from this base class such as the "*TCP/IP-Host*" and "*ATM-Host*" classes. The "ATM_Device" class, for example contains all the attributes and operations held by the "*Network Device*" class and may have some additional attributes and operations that are necessary for the ATM protocol.

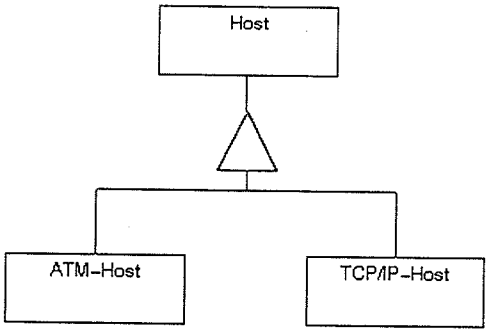


Figure 2.12 Example of inheritance.

Two characteristics of inheritance are discrimination and non-disjoint membership. When two classes are inherited from one super-class and these two sub-classes are distinguished by an attribute from the super-class, this attribute is known as the "*discriminator*". The

other characteristic, "*non-disjoint membership*", happens in case of multiple-inheritance, where a class is inherited from two or more classes and these super-classes happen to have the same name for an operation or attribute.

2.1.8 Abstract Class

A class may or may not have a direct object instance. Direct instance means that the class object is directly instantiated from the class definition. Those classes that do not have direct instances are called abstract classes and those classes that can have direct instances are called concrete classes, meaning that they are instantiable. An abstract class does not have a direct instance, while its descendent class(es) (sub-class) do. A concrete class may have an abstract class or a concrete class as its super-class. An abstract class is usually a base class, with object instances instantiated from its sub-classes. In general, abstract classes define the functions/methods to be inherited by its sub-classes.

Data that describes other data, such as a catalogue which describes a manufactures products is modeled as meta class.

2.1.9 Constraints

Constraints are functional relationships between the entities (class, object, attributes, link and associations) of an object model which restrict the value that the entity can hold. **Figure** 2.13 demonstrates an example of how constraints on an attribute and associations are modeled in OMT is given as in **Figure** 2.13. In the example **Figure** 2.13 [a], the class "*ATM Host*" has a constraint on the limitation of its interface speed.

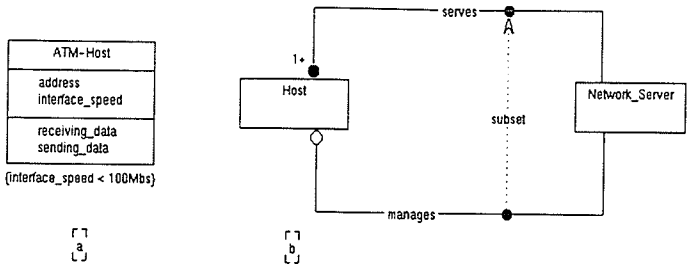


Figure 2.13 Example of constraints.

Figure 2.13 [b] depicts a constraint between the two associations of two classes, "*Host*" and "*Network_Server*". The first association between these two classes, "*serves*", indicates that a server provides services to many hosts within the system. The second association, "*manages*", shows that there is either zero or one host acting as a system manager with respect to the operation/performance of the "*Network_Server*". The dotted line between these two association is the constraint between them. This constraint indicates that the association "*manages*" is a subset of the association "*serves*", since the host acting as the system manager is a member of the set of hosts providing services "*Network_Server*".

2.1.10 Derived Entity

In the object model, some of the entities can be completely defined by the other entities in the system. The use of a derived entity is redundant but improves the ease of comprehension. A derived entity is denoted by a slash and contains the explanation as to how the entity is derived. **Figure** 2.14 demonstrates an example of derived attributes. The class "*Host*" has three attributes, "*Last_Polled_InCells*" which indicates the number of incoming cells at the host port from the last polling action, "*Polled_InCells*" which

indicates the current number of incoming cells at the host port, and *"Rate_InCells"* which indicates the rate of incoming cells at the host port. *"Rate_InCells"* is defined as a derived attribute since it can be completely determined by the other two attributes of the class through the formula provided, using the time interval of polling as a unit.

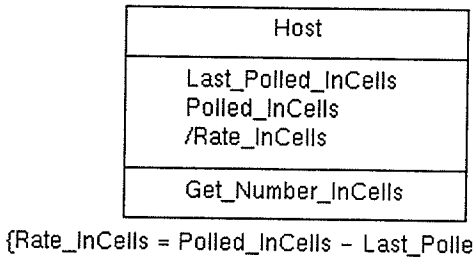


Figure 2.14 Example of derived attribute.

2.1.11 CONCLUSION

The object model can segment a large and complex system into smaller and more manageable sub-systems. Each sub-system can be then developed and tested independently for the most part. This capability provides the high degree of integration achievable using the object oriented approach in software system development.

This concludes coverage of the object model in the Object Modeling Technique (OMT) approach. Most of the major concepts in building an object model for a software system have been discussed. In Chapter 7, a complete design of the object model for a network management system will be presented. The next sections of this chapter will be devoted to a discussion of the concepts of dynamic models and functional models in OMT.

2.2 THE DYNAMIC MODEL

The static structure and the relationship between the objects in the system are modeled by the object model. The object model only presents the structure and relationships at a single moment in time [Elzakker Th. van 94]. The aspects of the system that change over time are modeled through the dynamic model which is the second model discussed of OMT. The dynamic model captures the control for sequences of operations that occurs without knowing what the they do, what they operate on or how they are implemented. A separate dynamic model is presented for each object to describe the internal changes and the dynamic behaviour of the object over time. The dynamic behaviour of the system as the whole can be visualized by the interactions among all of the dynamic behaviours of all of the objects.

The main components of the dynamic model are "*state*" and "*event*". "Event" is the action at a point in time. It does not express duration for the length of the action and is considered instantaneous. The event is a one-way transmission of information, without any returned value expected. The state of an object corresponds to the interval between two events occurring that affect the object. It is an abstraction of the attribute value and the link value of the object, those attributes that do not affect the behaviour of the object are ignored when defining the state.

2.2.1 STATE DIAGRAM

The *State Diagram* describes the relationship between the events and the state of the object over time. The changing of the object from one state to another is triggered by a specific

event and is called a *transition*. The state is often accompanied by a name. **Figure** 2.15 demonstrates an example of a state diagram for a computer terminal object. The rounded boxes are the states of the object, which are "*Off*" and "*On*". The events "*turn_on*" and "*turn_off*" cause transitions of state in the computer terminal from state "*Off*" to "*On*" and from state "*On*" to "*Off*", respectively.

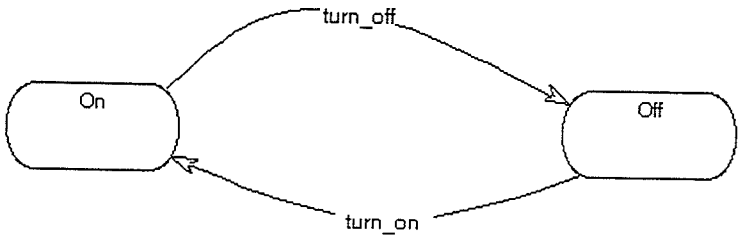


Figure 2.15 Example of a state diagram for a computer.

A dynamic behaviour of an object often starts with an initial state and terminates where the object exits. The initial and final states of an object' s dynamic behaviour are denoted with the special notations depicted in **Figure** 2.16. In this example, the "*SuperState*" is a simplification for a dynamic model which entered in an initial state and exited in a final state.

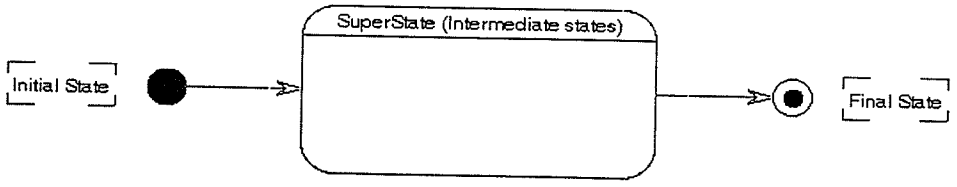


Figure 2.16 Notation for initial and final states.

A transition can be triggered with a condition, so that the event fires only when the condition, called a *guard*, is met. When an event occurs, it may optionally pass attributes from one state to another and perform an action on the transition. The notation for an unstructured state diagram is shown in **Figure** 2.17.

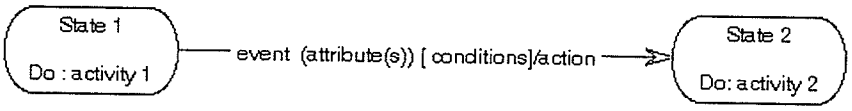


Figure 2.17 Notation for an unstructured state diagram.

When the object enters a particular state, the state may initiate some continuous activities or processes which stop when the object exits the state upon some event. In **Figure** 2.17, the process/activity is indicated by the notation "do:" in the state box. An event/action that occurs without changing the current state is called an internal action and is represented by a slash inside the state box. A transition without an event name is an *automatic transition,* which fires automatically when the activities associated with the source state are completed. Usually an automatic transition is accompanied by some sort of condition, so that when the activity in the state is completed, the condition is met and the automatic transition fires. If the current state has more than one automatic transition with different conditions for each transition to perform, the state will remain active until one of the conditions is met or until an event causes another transition to fire. An event can be sent among the objects so that the objects within the system interact to perform the system operations. An object can receive one or more events from one or more objects. The order of receiving the event affects the final state of the object. An example of transmitting and

receiving events to and from another class, is given in **Figure** 2.18. In this example, there is an event that passes data from class *"Manager_Entity"* to class *"Host"*.

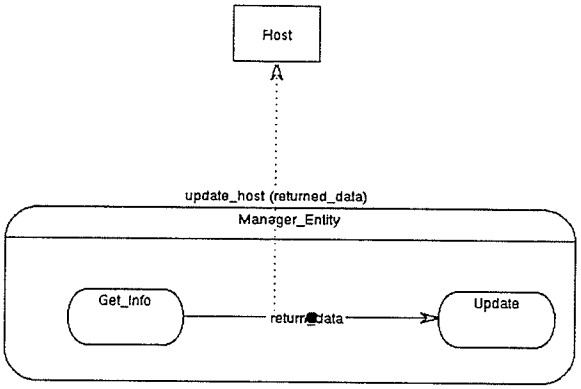


Figure 2.18 Example of sending event to object.

2.2.2 CONCURRENCY

The behaviour of the system performed by the interaction among objects and may result in many of their dynamic states changing simultaneously. A dynamic model for an object thus can not possibly describe the whole system's dynamic behaviour, resulting in the need for concurrency among dynamic models. The objects in a system are concurrent inherently and can change state independently [Rumbaugh J. 91]. Within a single object, concurrency may arise when the object can be partitioned into subsets of attributes and links with each subset having its own state diagram and change their states independently.

2.2.3 SYNCHRONIZED ACTIVITIES

Splitting control of an action into two or more parts is used to describes two or more activities performed concurrently. Merging control from two or more activities describes

the completion required of these activities before the next state of the object can be obtained. The completion order of these activities may not be important, but the completion of all activities must be met. **Figure** 2.19 gives an example of splitting and merging controls. In this example, the event control is split in to 2 different control signals and each of the signal evokes an activity. Each of the split events activates one of the activities, "*turn_on_screen*" and "*turn_on_computer*". The state "*System_On*" of the dynamic model will not be reached until both of the activities are completed.

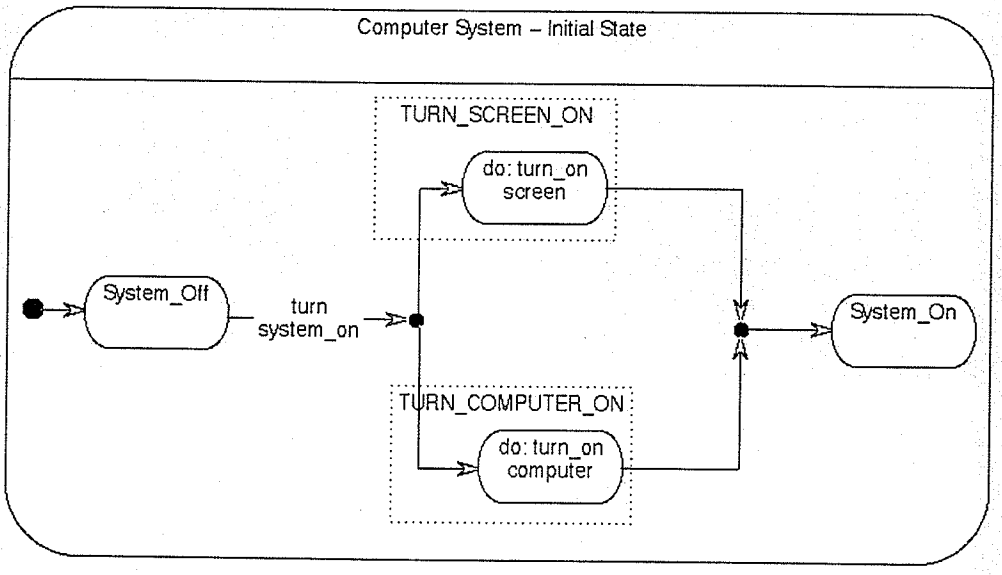


Figure 2.19 Example of splitting and merging control.

The above example also demonstrates the use of concurrency. The states "TURN_SCREEN_ON" and "TURN_COMPUTER_ON" are concurrent. Splitting controls can only exist in concurrent situations.

2.2.4 CONCLUSION

The above section has described the dynamic model for system development using OMT. It has presented the major concepts of the dynamic model which allows it to describe the allowable sequence of changes within the object and the interactions between objects in the system as a whole.The next section discusses the last model of the OMT methodology, the functional model.

2.3 THE FUNCTIONAL MODEL

The object model presents the static structure and relationship of objects in the system, and the dynamic model presents the dynamic behaviour of the objects and the system as the whole. The third and last discussed model of OMT is the Functional Model, which describes the computation performed within the system without specifying how or when these operations/computations take place. The functional model specifies the implementation of the behaviour of the object. It specifies the meaning of methods and constraints in the object model and the actions in the dynamic model. The functional model is graphically represented by a multiple Data Flow Diagram (DFD). The DFDs describe data flows in the system and the functional relationships of values computed by the system such as input, output values and internal storage. The four major components in the DFD are: *process, data flow, actor* and *data store,* each of which is further discussed in detail in the following sections.

2.3.1 PROCESS

A process describes the computations/transformations/evaluations of the input values. The lowest level process is a pure function without any side effects and the highest level process is the entire data flow diagram itself. **Figure** 2.20 gives an example of a process. In this example, there are three processes which are notated with an oval shape: *"Addition"*, *"Check_Sum"*, and *"Further_Process"*. There are also actors, "*Data A"* and *"Data B"*, and data flows, both of which will be defined later in this section. The process accepts data flows as inputs,"*A1"* and "*B1"*, and produces a data flow as an output, "*sum*". The process performs a calculation/evaluation on the input data as required to in order obtain a result that can be input to another process or a final output.

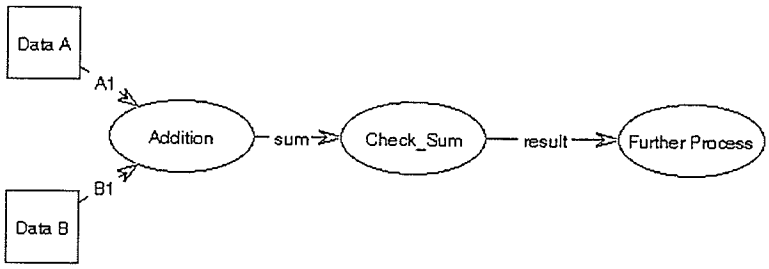


Figure 2.20 Example DFD

2.3.2 DATA FLOW

A data flow is a connection between the input/output of two objects or processes. In the above example, the flow of values "*A1"*,"*B1"*, *"Sum* "are known as data flows. A data flow can be treated as either input to a process or output from a process. The flows of "*A1"* and *"B1"* are the input data flows to the process. "*Sum*" acts as both input and output data flow, it is output from process "*Addition*" and input to process "*Check sum*".

An aggregation data value may have split components, with each component going to a different process. Also, many data flows can be merged into one data flow and input to another process. **Figure** 2.21 gives an example of splitting and merging a data flow. In this example, the data flow "*result*", which is the output of the process "*Integer_Division*", is two-component data. The components of this data flow "*result*" are the "*quotient*" and "*remainder*" resulting from integer division. These components of the data flow "*result*" are split and each component is input to separate processes "*Quotient_Process*" and "*Remainder_Process*". The results of these processes,"*quotient_result*" and "*remainder_result*", are then merged together and input to the final process, "*Output*". The data flow represents an intermediate data value within a computation which is not changed by the data flow.

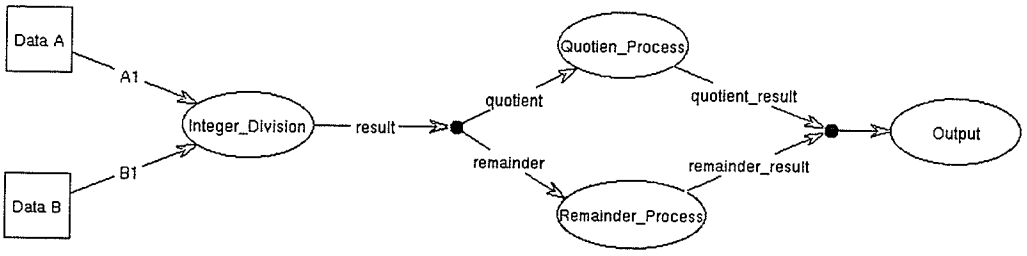


Figure 2.21 Example of splitting and merging data.

2.3.3 ACTORS

Actors are active objects that produce or consume values from or to a data flow. They are attached to the ends of the data flow and denoted by a rectangular box. An example of an actor is shown in **Figure** 2.20, where actors "*Class A*" and "*Class B*" produce the data "*A1*"

and "*B1*" as input into the process "*Addition*".

When the data of a actor/data store is accessed and updated (with new values), the two data flows in and out of the actor for accessing and updating can be combined to make a two-way data flow, as shown in **Figure** 2.22. In this example, the password is accessed from the "*Password_Data_Base*", and when the process "*Change Password*" completes the password validation and replaces the old password with the a new password, it sends back the updated password to the actor, "*Password Data Base*" to be stored. Since the password data base is modeled as an actor, it can manipulate the stored data, for instance encrypting the password for security purposes.

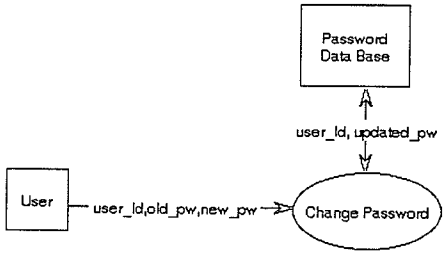


Figure 2.22 Example of two-way data flow.

2.3.4 DATA STORE

A data store is a passive object that holds data and does not perform any operation on the data. It only responds to requests to store and to access the data contained within itself. The orders of data storing and accessing are not necessarily the same.Stored data can be partially or wholly retrieved. It can also be modified by adding or deleting elements. **Figure** 2.23 gives and example of accessing a data store. In this example, the MIB (Management Information Base) is the data store that contains information of each of the managed objects

for the management system. When the "*Manager_Entity*" sends a request to obtain a managed object value from a network device such as a host, the "*Manager_Entity*", modeled as an actor, sends the address and the name of the requested managed object to the "*Get_Request*" process. The "*Get_Request*" process obtains the managed-object identification of the managed object from the MIB using the managed object name as the key, since each managed object has a unique object name and object identification. The "*Get_Request*" process will get the desired managed object value from the device address and managed object identification. The result of this process is sent to the "*Display_Managed_Object_Value*" process to be output graphically or textually by the manager system.

The difference between an actor and a data store is that data can be generated and processed by an actor, but only stored and retrieved by the data store. The two major operations on the data store are "*retrieving*" and "*storing*" a value. The data is updated by retrieving, and then modified by a process and stored back to the data store.

A data flow diagram may contain many high level processes and they can each be expanded into a data flow diagram itself. Each of the new data flow diagrams can contain all the major components of the functional model.

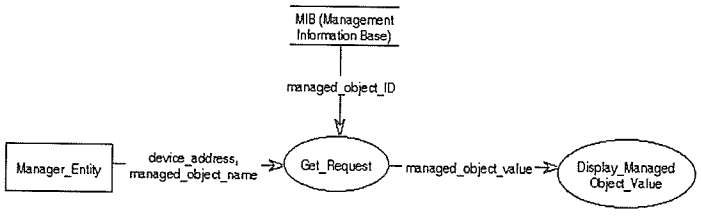


Figure 2.23 Example of data store.

The data flow can not only pass the value from process to process, but can also pass a boolean value to control whether the next process is to be evaluated/processed or not. The result of a process is a decision as to whether the next process will be executed or ignored or some other subsequent process will be executed upon the value of this data, known as control data. The control is denoted by a dotted line as shown in **Figure** 2.24. In this example, the "*Login_Validation*" process produces a control data "*valid_user*" that will determine if the "*Login*" process can take place. If the "*Login_Validation*" does not produce a valid_user, the "*Login*" process is ignored.

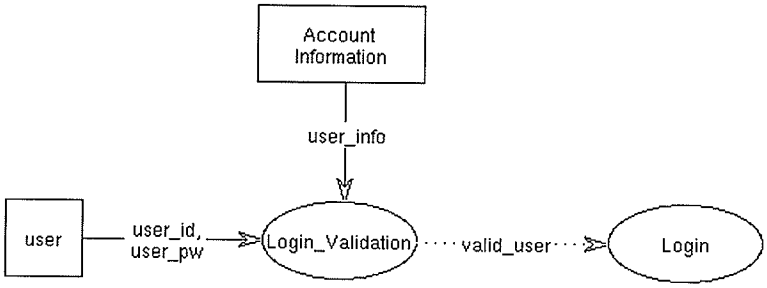


Figure 2.24 Example of control data.

The data flow can pass the object as data from a process to a data store, the flow of an object is denoted by a special notation. An example of a data object flow is depicted in **Figure** 2.25. The "*Create_Managed_Object*" process creates the "*managed_object*" which is to be stored in the data store "*Managed Object Data Base*". Each "*managed_object*" is an object in itself.

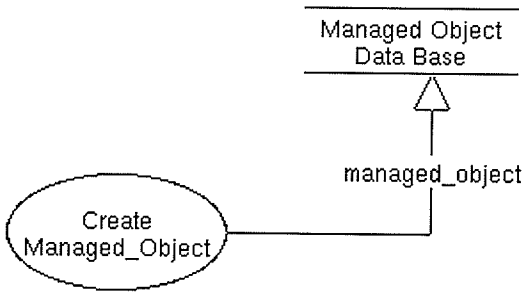


Figure 2.25 Example of an object data flow.

2.3.5 CONCLUSION

The operations/methods of the object are modeled with the functional model, which describes in detail the how the operations actually perform. In OMT, the functional models are not necessary for those functions that are comparably simple. The functional model can be decomposed into many levels depend on the complexity of the function.

2.4 OMT IMPLMENTATION

The OMT approach subdivides the software system into three models with each of the models representing an aspect of the system: the static structures and relationships between the classes in the system by the object model, the behaviour of the classes by the dynamic models and the functionality of the classes by the functional models. Together they represent a operational system. This section contains a brief introduction to the implementation of the object model using the object oriented programming language C++. Even though OMT is an object oriented approach for system design and analysis, the choice

of language and methods implementation system designed by OMT needs not be an object oriented programming language.

In this section, the implementation of a system (in C++ language) is illustrated. It briefly suggests how to create classes and how to implement the relationships between the classes in C++. For this discussion the object model as shown in **Figure** 2.26, is used, and contains the following information:

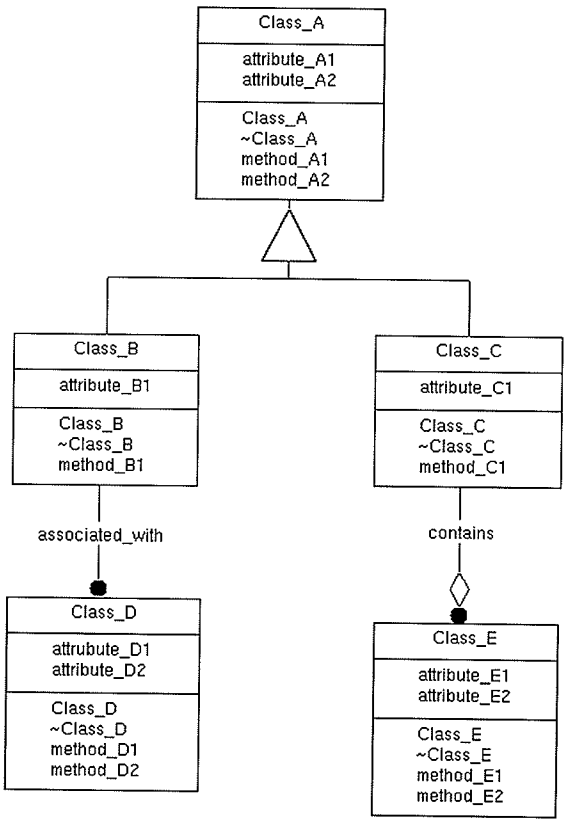


Figure 2.26 Example of Object Model.

2.4.1 OBJECT MODEL

This example object model demonstrates how an object model can be manually implemented in C++. The object model for this system consists of five classes, namely *"Class_A"*, *"Class_B"*, *"Class_C"*, *"Class_D"*, and *"Class_E"*. The relationships between the classes in this object model are association, aggregation and inheritance. In the following sections, a common way to define these classes and implement their relationships in C++ is presented. In the next chapter, the same object model will be used to demonstrate how this model is implemented in C++ using the software package Software through Picture/Object Modeling Technique which will be introduced later.

2.4.2 CLASS DEFINITION

For class *"Class_A"*, the definition is as follows:

```
class Class_A
{
        private:
                Type_A1 attribute_A1; // attribute_A1 has type of Type_A1
                Type_A2 attribute_A2; // attribute_A2 has type of Type_A2
        public:
                Class_A (variable_llist); // class constructor
                ~Class_A()                      // class destructor
                Type_Method_A1 method_A1(variable_list_A1); // operation method_A1 has returned
                                                            // type of Type_Method_A1
                Type_Method_A2 method_A2(variable_list_A2); // operation method_A2 has returned
                                                            // type of Type_Method_A2
}
```

2.4.3 OBJECT INSTANTIATION

The object instance of a class is created by declaring the variable name with the class name as the type of the variable. This will automatically construct the object using a constructor. If the constructor needs parameters, the following may used:

```
Class_A Object_A;
Object_A = new Class_A(variable_list);
```

where the variable Object_A is of the type Class_A. In the second statement, a constructor is used to create a object instance Object_A of class Class_A.

In the second statement above, "Object_A = new Class_A(variable_list);", the constructor is evoked to construct the object. Usually, some of the initial values of the attributes are assigned. The destructor of the class is evoked when the procedure/functions which contains the creation of the object exits. The purpose of object destruction is to release the memory that is allocated to these objects that is no longer needed.

2.4.4 METHOD IMPLEMENTATION

For the constructor and destructor, the implementation is as follows:

```
Class_A:: Class_A (varaible_list) // constructor of the class
{
        C++ code for constructing the class;
}

Class_A:: ~Class_A (void) // destructor of the class

{
        C++ code for destructing the class
}
Type_Method_A1 Class_A:: method_A1(variable_list_A1) // Implementation of method
{
        C++ code for this method;
}

Type_Method_A2 Class_A:: method_A2(variable_list_A2) // Implementation of method
{
        C++ code for this method;
}
```

2.4.5 INHERITANCE

As in the above object model, classes "*Class_B*" and ""*Class_C*" are inherited from "*Class_A*", so these two sub-classes have attributes and operations inherited from the super-class. The implementation of the sub-class can be as follows:

```
class Class_B: public Class_A
{
        private:
                Type_B1 attribute_B1; // attribute_B1 has type of Type_B1
        public:
                Class_B (variable_list): Class_A(variable_list) // Inline constructor
                {
                        C++ code
                }
                ~Class_B;
                Type_Method_B1 method_B1(variable_list_B1); // operation method_B1 has returned
                                                            // type of Type_Method_B1
}
```

If the operation from the super-classes are inherited and modified by the sub-class, the operations/methods in the super-class should be declared as virtual, and the code for these operations/methods are re-implemented in the sub-class.

2.4.6 ASSOCIATION

The implementation for aggregation is similar to the implementation of association, since the aggregation is special form for association. In the above object model, the class "*Class_B*" is associated with "*Class_D*" and class "*Class_E*" is aggregated from class "*Class_C*". The following suggests an implementation of the association and aggregation relationships for the above object model.

```
class Class_D
{
        private:
                Type_D1 attribute_D1;
                Type_D2 attribute_D2;
        protected:
                Class_B *associated_with;
        public:
                Class_D(varaible_list);  // constructor
                ~Class_D ();                    // destructor
                Type_Method_D1 Class_D:: method_D1 (varialble_list_D1);
                Type_Method_D2 Class_D:: method_D2 (varialble_list_D2);
}
```

The class "*Class_C*" contains class "*Class_E*" as its component, so the class "*Class_E*" can be implemented as an attribute of class "*Class_C*". The suggested implementation as following:

```
class Class_E
{
        private:
                Type_E1 attribute_E1;
                Type_E2 attribute_E2;
        public:
                Class_E(varaible_list);  // constructor
                ~Class_E ();                    // destructor
                Type_Method_E1 Class_E:: method_E1 (varialble_list_E1);
                Type_Method_E2 Class_E:: method_E2 (varialble_list_E2);
}

class Class_C: public Class_C
{
        private:
                Type_C1 attribute_C1; // attribute_C1 has type of Type_C1
        protected:
                Class_E *contains; // attribute_C1 has type of Class_E
        public:
                Class_C (variable_list): Class_A(variable_list) // Inline constructor
                {
                        C++ code
                }
                ~Class_C ();
                Type_Method_C1 method_C1(variable_list_C1); // operation method_C1 has returned
                                                            // type of Type_Method_C1
}
```

2.5 CONCLUSION

This chapter discussed one of the OOT approaches for software design and analysis, the Object Modeling Technique, OMT, developed by James Rumbaugh. OMT consists of three models with each of the models corresponding to one aspect of the system, and together presenting an operational system. The chapter included a section on the implementation of the objects and their relationships in an object model using object oriented programming language C++. The next chapter will introduce the software package StP/

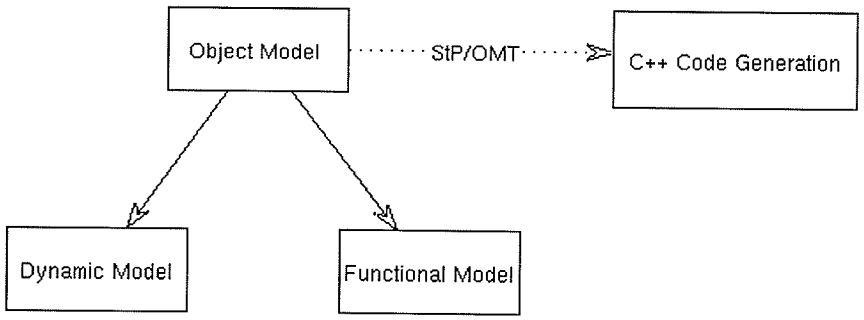
OMT [Software through Pictures/ Object Modeling Technique], which is a tool for developing software systems using the OMT.

CHAPER 3
SOFTWARE THROUGH PICTURE (StP/OMT)

Software through Pictures/Object Modeling Technique (StP/OMT)is a tool that is used as an aid for developing software systems designed by the OMT methodology discussed in the previous chapter.

The major functions of StP/OMT are:

- Analysing and defining object, dynamic and functional models.
- Capturing full definitions of classes (attributes and operations) for each of the defined objects in the object model.
- Capturing full definition of states (actions and activities) in dynamic models.
- Capturing data transformations in functional models.
- Navigating between the object model, dynamic models and functional models during thedesign process.
- Validating the OMT models for consistency and completeness.
- Generating C++ code for the system developed based on the static structure of the object model as shown in figure below



StP/OMT is a graphical-user interface tool with features to design the three models of the OMT methodology. It provides both drawing (graphic) and table (text) editors for modeling object, dynamic and functional models. The table editor for the object model is called the class table, and for the dynamic model is called the state table. The software also provides the OMT browsers, object annotation editors, script, as well as OMT tool and conversion utilities. The **Figure** 3.1 displays the desk top for the StP/OMT, in which each icon represents a tool, such as for object modeling, dynamic modeling, functional modeling, etc.

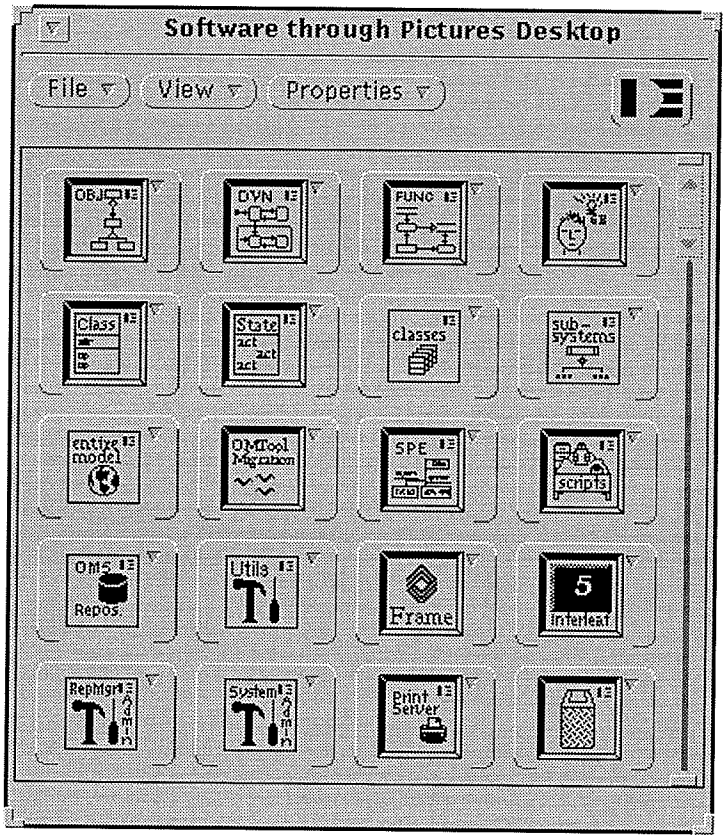


Figure 3.1 Desktop of StP/OMT

3.1 OBJECT MODELING WITH StP/OMT

Figure 3.2 shows the StP/OMT object drawing editor. It contains all the components that can be used to model object models as described in the previous chapter.

From this editor, the designer can navigate to other StP editors or design tools, such as the dynamic model, the functional model, the class table editor, or other programming environments. Since one model of the designed system does not have to be completed before the next model can be started, many models, such as object models and dynamic models, can be concurrently designed.

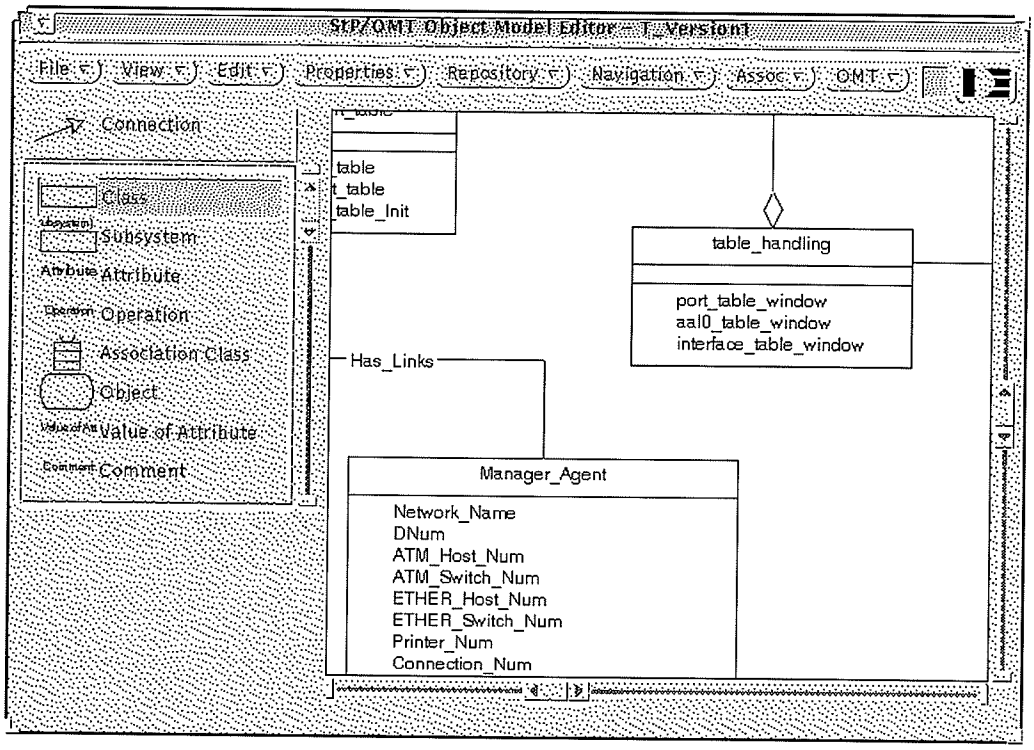


Figure 3.2 Object Model Graphical Editor

The object can be fully defined by using the class table editor. The desktop for the class table editor is shown in **Figure** 3.3. It contains the information on the specified designed objects of the system which are necessary for the C++ code generation of the object. Along with the object model drawing editor, other editors can be navigated to from the class table editor. To create a class table for each class, the class is selected in the object model and the "*Class Table*" option in the "*Navigation*" menu is chosen.

The main information this editor presents is the types of attributes and operations for the class. Attributes can be assigned with initial values. Operations includes a list of parameter that the operation passes when the operation is evoked. The attribute(s) / operation(s) can have their accessibility specified as private, public or protected by the class table edi-

tor.

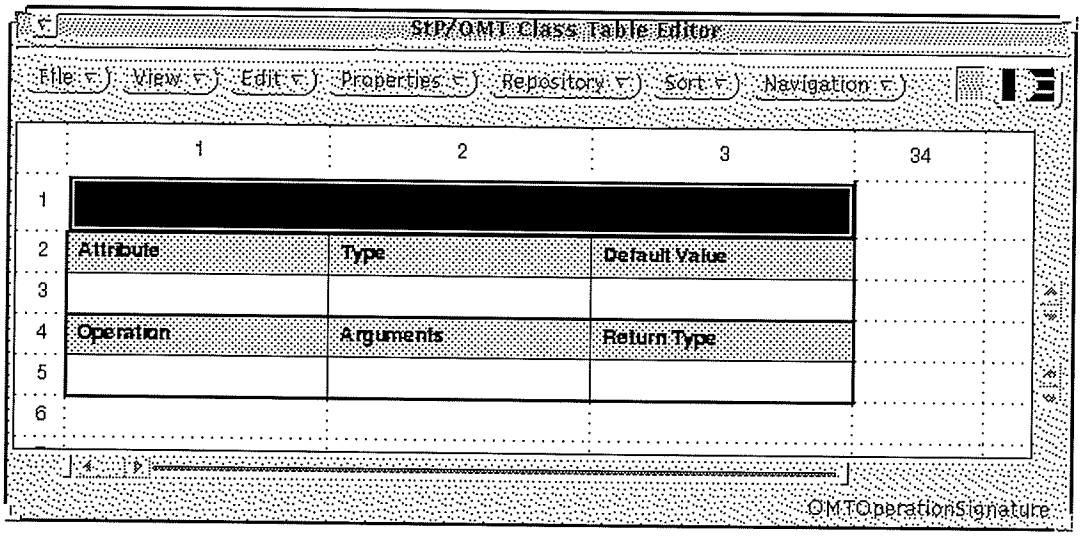


Figure 3.3 Object Model Table Editor

3.2 DYNAMIC MODEL WITH StP/OMT

The dynamic behaviour of the object can be modeled by both the drawing editor, as shown in **Figure** 3.4, and the state table editor, as shown in **Figure** 3.5. The designer can also navigate to other editors from both dynamic drawing and state table editors.

It is suggested that there be a dynamic model for each defined class in the object model. To create the dynamic model for the class in the object model, the class is selected and the "*Dynamic Model*" under the "*Navigation*" menu is chosen. The dynamic model can consist of many states and each state can be modelled textually by the state table editor. In the state table editor, the main information provided about the state is the action on the entry to the state, the activity(ies) during the state of the object, and the action(s) that causes the exit from the state. It also includes the internal events and actions that take place during

the state, but do not cause the object to exit the current state.

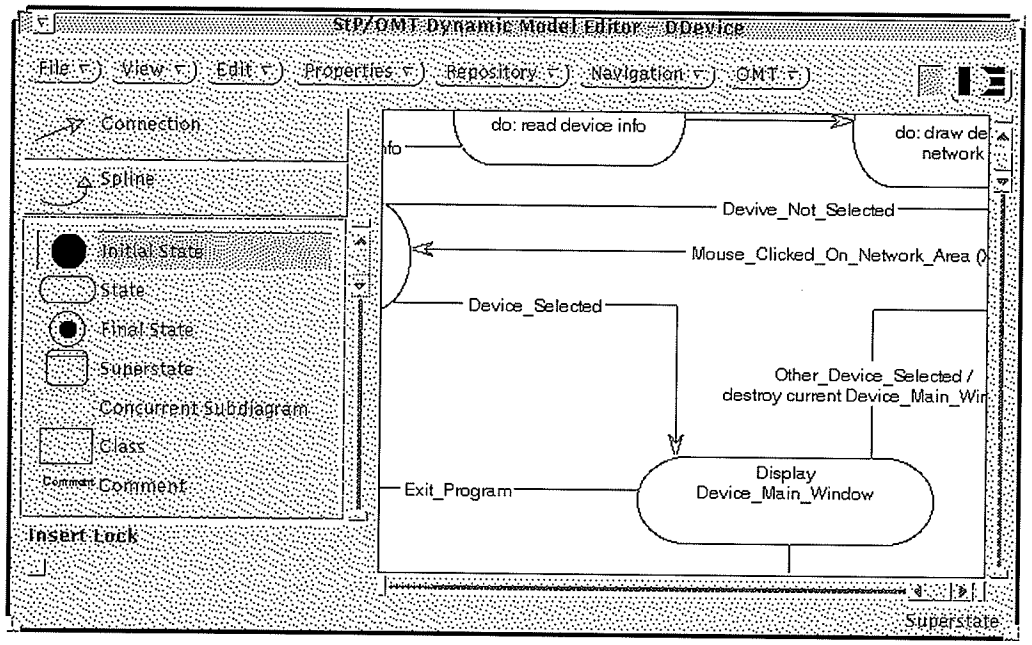


Figure 3.4 Dynamic Model Graphical Editor

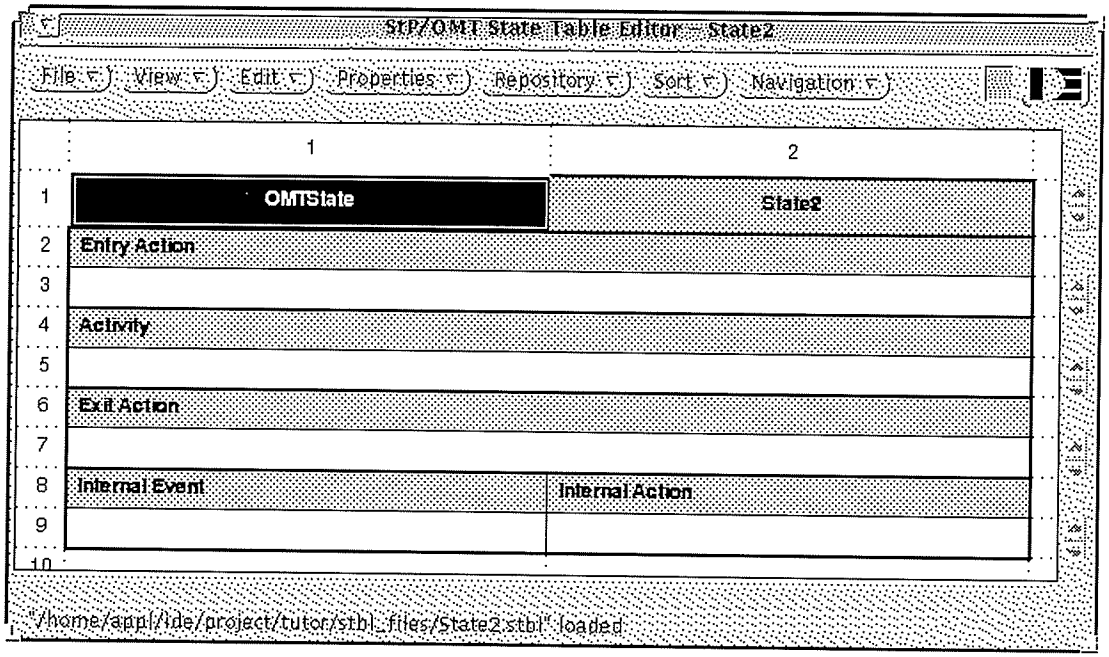


Figure 3.5 Activity and action Editor.

3.3 FUNCTIONAL MODEL WITH StP/OMT

Figure 3.6 gives the desktop for the functional editor of StP/OMT. Again, other editors can be navigated from the functional editor. For nearly every function/method of the class, there should be a functional model to explain or display how the function is performed graphically. To start the functional model, the function/method is selected from the object model editor or class table editor and the "*Functional Model*" under "*Navigation*" menu is chosen.

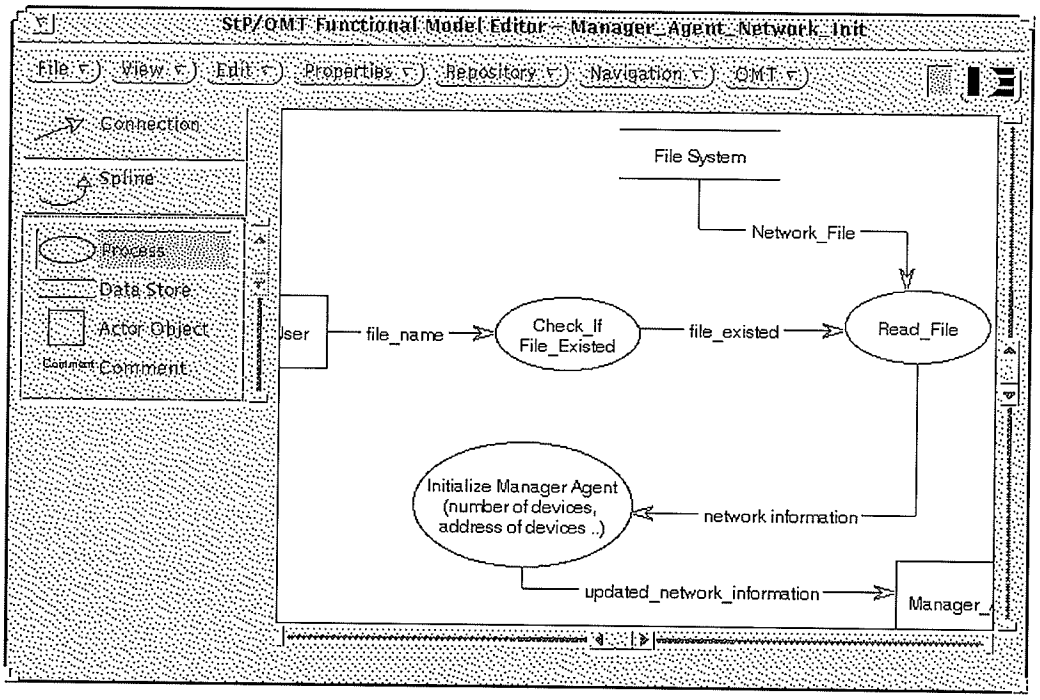


Figure 3.6 Functional Model Graphical Editor

In each of the editors mentioned above, StP/OMT provides a system checker for the completeness and consistency of the system between models.

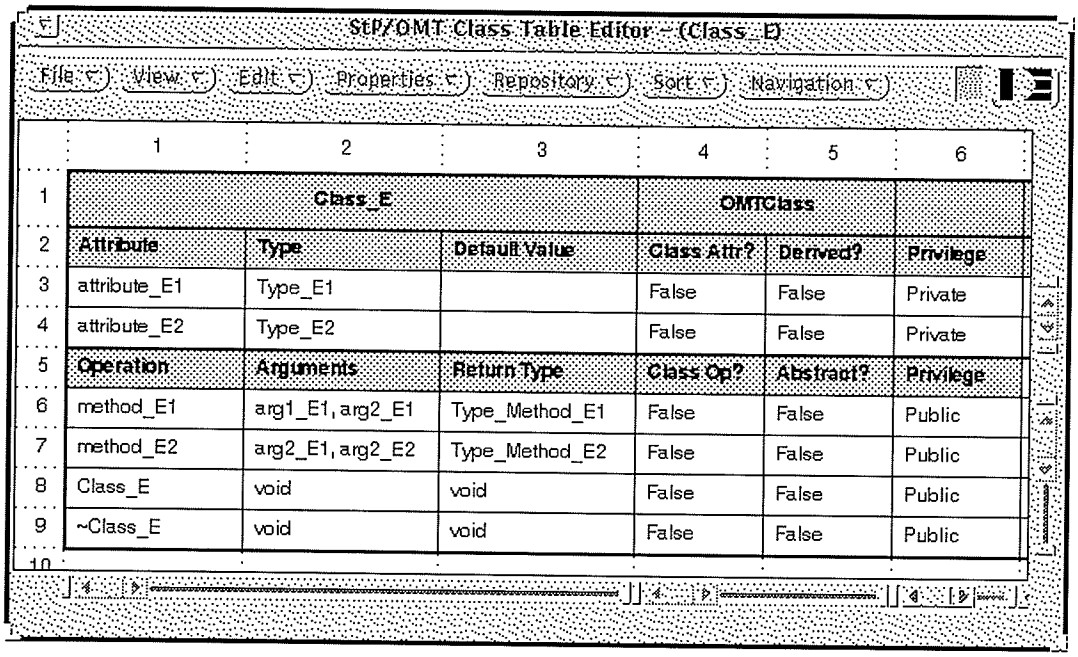
3.4 C++ CODE GENERATION

The object model of the design is used for generating C++ code for the system. Class templates in the object models and their static components, such as attributes, function prototype, and relationships, such as association, aggregation and inheritance, which are defined in the class table editor can be automatically generated. These templates can be generated for each class or for the system as the whole.

The StP/OMT provides a script language compiler that the designer can use to customize how the C++ code is generated.

Each class in the object model is generated with two C++ code files, the interface (header) file and the implementation file. The interface file contains the declaration of the class with its attributes, relationships with other classes (as desired) and the function(s) / procedure(s) headers. The implementation file contains the prototype for the implementation of the function(s) / procedure(s) of the class. For the actual implementation of the function(s) / procedure(s), the designer must annotate each of the function(s) / procedure(s) before generating the C++ code or after the templates are generated.

Before the C++ code can be generated, the object class information must be entered textually in the class table editor. The C++ code generated is based on the information provided in the class table. An example of the class table is given in **Figure** 3.7. The class without a defined class table will not be generated C++ because not enough information has been provided for generating code.



	1	2	3	4	5	6
1	Class_E			OMTClass		
2	Attribute	Type	Default Value	Class Attr?	Derived?	Privilege
3	attribute_E1	Type_E1		False	False	Private
4	attribute_E2	Type_E2		False	False	Private
5	Operation	Arguments	Return Type	Class Op?	Abstract?	Privilege
6	method_E1	arg1_E1, arg2_E1	Type_Method_E1	False	False	Public
7	method_E2	arg2_E1, arg2_E2	Type_Method_E2	False	False	Public
8	Class_E	void	void	False	False	Public
9	~Class_E	void	void	False	False	Public
10						

Figure 3.7. Example of entering data class table editor

The C++ code can be generated with different options, for how each class is implemented or how the relationships between classes are generated. **Figure** 3.8 shows the options for generating the C++ code.

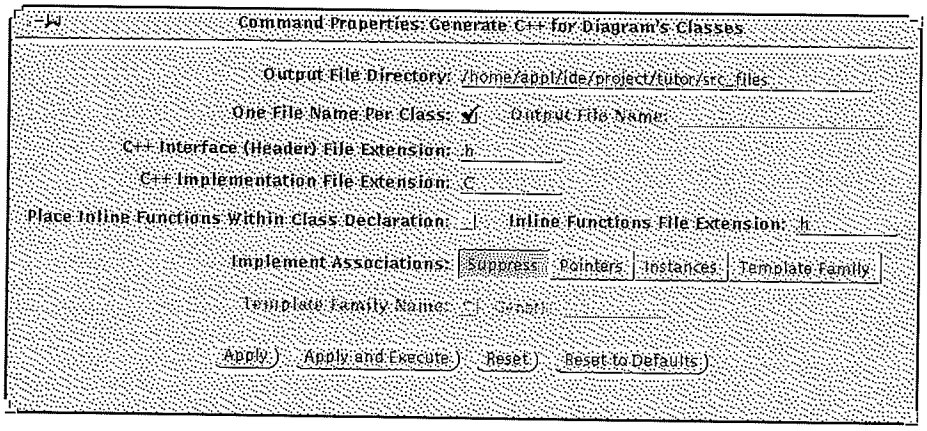


Figure 3.8. C++ Properties

The example object model in Chapter Two (**Figure** 2.27) is used in this chapter as an

example to generate the C++ code. All the classes have their information entered into the class table editor and the model object is updated as in **Figure** 3.9. The little box on the top-right of each class indicates that this class had its table class defined. Each of the attributes of the classes has the type indicated in the class attribute section. The generated code also includes the date and time of generation. The advantages of the automated code generation are apparent from the files reproduced below. The structural generation is largely a mechanical process otherwise subject to manual coding and syntax errors. The functional design is still left to the designer who can concentrate on the implementing the function as opposed to the structure and module interface.

3.4.1 INTERFACE FILE (*.H)

```
// StP/OMT -- created on Sun Feb 04 14:32:42 1996 for tdang@enterprise from system tutor
// C++ class interfaces

#ifndef _example_h_
#define _example_h_

// stp/omt class declarations
class Class_A;
class Class_B;
class Class_C;
class Class_D;
class Class_E;
// stp/omt class declarations end

// stp/omt class definition 17746
class Class_A
{
// stp/omt class members
public:
   void Class_A(void);
   void ~Class_A(void);
   Type_Method_A1 method_A1(arg1_A1,arg2_A1);
   Type_Method_A2 method_A2(arg1_A2, arg2_A2);
protected:
private:
   Type_A1 attribute_A1;
   Type_A2 attribute_A2;
// stp/omt class members end
```

```
};
// stp/omt class definition end

// stp/omt class definition 16804
class Class_B : public Class_A
{
// stp/omt class members
public:
   void Class_B(void);
   void ~Class_B(void);
   Type_Method_B1 method_B1(arg1_B1, arg2_B1);
protected:
   Class_D* associated_with;
private:
   Type_B1 attribute_B1;
// stp/omt class members end
};
// stp/omt class definition end

// stp/omt class definition 16817
class Class_C : public Class_A
{
// stp/omt class members
public:
   Type_Method_C1 method_C1(arg1_C1, arg2_C1);
   void Class_C(void);
   void ~Class_C(void);
protected:
   Class_E* ptrClass_E;
private:
   Type_C1 attribute_C1;
// stp/omt class members end
};
// stp/omt class definition end

// stp/omt class definition 16807
class Class_D
{
// stp/omt class members
public:
   Type_Method_D1 method_D1(arg1_D1, arg2_D1);
   Type_Method_D2 method_D2(arg1_D2, arg2_D2);
   void Class_D(void);
   void ~Class_D(void);
protected:
   Class_B* associated_with;
private:
   int attrubute_D1;
   Type_D2 attribute_D2;
// stp/omt class members end
};
// stp/omt class definition end

// stp/omt class definition 16815
class Class_E
{
// stp/omt class members
public:
   Type_Method_E1 method_E1(arg1_E1, arg2_E1);
```

```
    Type_Method_E2 method_E2(arg2_E1, arg2_E2);
    void Class_E(void);
    void ~Class_E(void);
  protected:
    Class_C* ptrClass_C;
  private:
    Type_E1 attribute_E1;
    Type_E2 attribute_E2;
  // stp/omt class members end
  };
  // stp/omt class definition end

  // stp/omt footer
  #endif
  // stp/omt footer end
```

3.4.2 IMPLEMENTATION FILE (*.C)

```
#include "example.h"

// stp/omt operation 17746::19206
void
Class_A::Class_A(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 17746::19208
void
Class_A::~Class_A(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 17746::19210
Type_Method_A1
Class_A::method_A1(arg1_A1,arg2_A1)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end
```

```
// stp/omt operation 17746::19212
Type_Method_A2
Class_A::method_A2(arg1_A2, arg2_A2)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16804::19405
void
Class_B::Class_B(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16804::19407
void
Class_B::~Class_B(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16804::19409
Type_Method_B1
Class_B::method_B1(arg1_B1, arg2_B1)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16817::19705
Type_Method_C1
Class_C::method_C1(arg1_C1, arg2_C1)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end
```

```
// stp/omt operation 16817::19707
void
Class_C::Class_C(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16817::19709
void
Class_C::~Class_C(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16807::19906
Type_Method_D1
Class_D::method_D1(arg1_D1, arg2_D1)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16807::19908
Type_Method_D2
Class_D::method_D2(arg1_D2, arg2_D2)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16807::19910
void
Class_D::Class_D(void) : attrubute_D1(100)
{

        // stp/omt code

        // stp/omt code end

}
```

```
// stp/omt operation end

// stp/omt operation 16807::19912
void
Class_D::~Class_D(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16815::20206
Type_Method_E1
Class_E::method_E1(arg1_E1, arg2_E1)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16815::20208
Type_Method_E2
Class_E::method_E2(arg2_E1, arg2_E2)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16815::20210
void
Class_E::Class_E(void)
{

        // stp/omt code

        // stp/omt code end

}
// stp/omt operation end

// stp/omt operation 16815::20212
void
Class_E::~Class_E(void)
{

        // stp/omt code

        // stp/omt code end
```

```
}
// stp/omt operation end
```

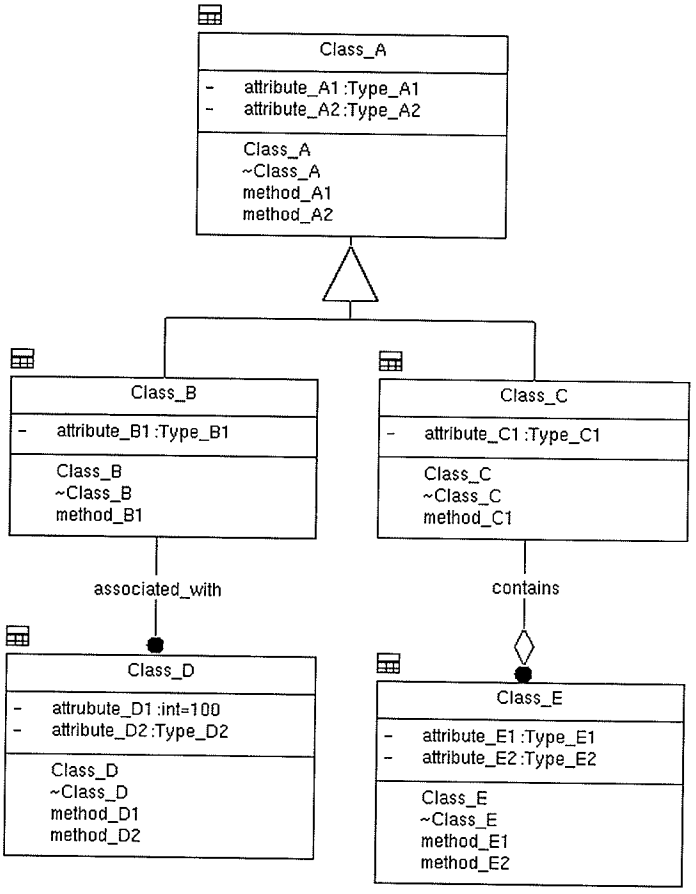


Figure 3.9 Object Model with class table defined .

3.5 CONCLUSION

This chapter presented a brief introduction to the software StP/OMT that is used to design software systems developed through the OMT approach. For further information on the StP/OMT and how to use the software, please refer to the StP/OMT software manuals.

Even though the StP/OMT supports the C++ generation of the system being designed using the OMT methodology, the C++ generated is based only on the static structure of the object model. It lacks support for the dynamic behaviour implementation of the system, not taking the dynamic model and functional model into consideration for code generation. The dynamic models and functional models, which layout the step-by-step basis of the behaviour and functions of the system are used by the system designer only to present the system to other people graphically, and followed to generate code for the system manually. It is hoped that with the improvement of software development, software scientist in the future will create a better and more detailed syntax for developing dynamic and functional models, so that software like StP/OMP will not only generate code based on the graphical representation of object models, but will also have the capability to generate code for the whole system, producing an operational software based on three models.

The next chapters introduce the ATM networking technology and network management.

CHAPTER 4
ASYNCHRONOUS TRANSFER MODE (ATM)

4.1 INTRODUCTION

ATM is the abbreviation for Asynchronous Transfer Mode. The birth of ATM in the mid '80s was a significant step in history of telecommunication. This chapter will first briefly introduce the Open System Interconnection (OSI) protocol and ATM technology.

4.2 OSI PROTOCOL

The seven-layer Open System Interconnection(OSI) Reference Models (OSIRM) provides the most useful organization for unifying communication practices and standards. The basic OSIRM is shown in **Figure** 4.1 [McDysan D. E. Spohn D. L. 95]. Each layer in the model is responsible for some aspect of communication and together they accomplish the user transformation of information from point to point in the network. The flow of data starts at the highest level of the data originator to level one where it is transmitted across a network of immediate nodes over the interconnecting physical media to the level one layer of the receiver. The data is then transformed from the lowest layer level to the highest layer level at the receiving site.

The following paragraphs briefly describe each layer and its responsibility in the intercon-

nection network.

4.2.1 PHYSICAL LAYER

This layer provides the transparent transmission of the data bit stream from point to point over the network. The transmission may be in one direction only (simplex mode), in both directions with one direction at a time and the direction controlled by a change in signalling in the physical layer (half duplex), or both directions simultaneously (full duplex).

4.2.2 DATA LINK LAYER

This layer provides error detection and possibly error correction ability. It also provides a reliable protocol interface across the physical layer.

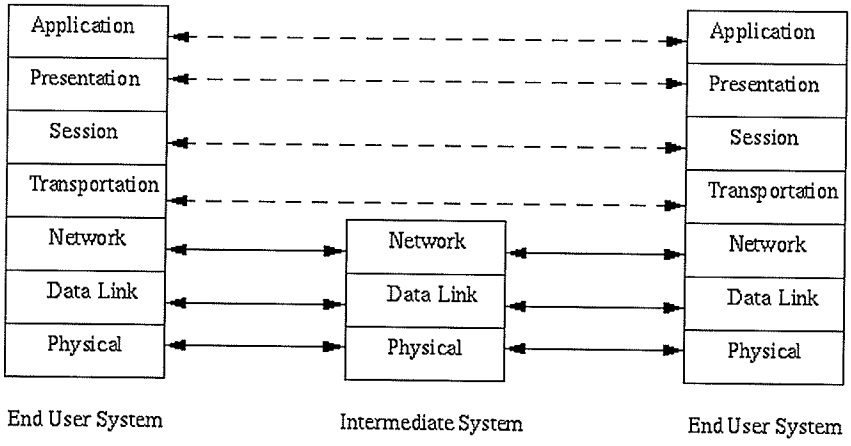


Figure 4.1 The seven layer Open System Interconnection Reference Model

4.2.3 NETWORK LAYER

This layer provides reliable, in sequence delivery of data between transport layer entities.

4.2.4 TRANSPORT LAYER

This layer is responsible for data segmentation, reassembly and multiplexing over a single network layer interface. It is also responsible for end-to-end control of transmitted data and optimization of the use of network resources. It can possibly perform some degree of error detection and correction.

4.2.5 SESSION LAYER

This layer is essentially the user interface to the network. It provides a connection between the stations that allows them to communicate directly.

4.2.6 PRESENTATION LAYER

This layer determines the presentation of data to the user. It delivers information to the communicating application, resolving syntactical differences but preserving the meaning.

4.2.7 APPLICATION LAYER

This layer enables the application process to access the OSI.

4.3 ASYNCHRONOUS TRANSFER MODE PROTOCOL

Asynchronous Transfer Mode is connection-oriented and cell-based switching and multiplexing technology. The connection has to be set up prior to the transmission of data information. **Figure** 4.2 [McDysan D. E. Spohn D. L. 95] shows the layer representation for the ATM protocol: the Physical ATM layer corresponds to the Physical layer and the Data-

Link layer of the OSIRM. The higher layers of ATM protocol correspond to the OSI model layer three and higher.

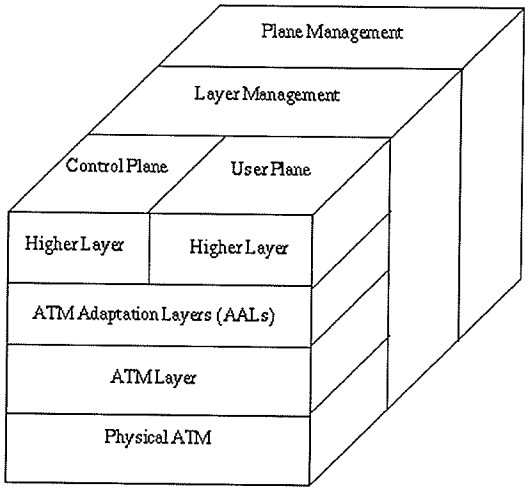


Figure 4.2 ATM Protocol Layer Representation

4.3.1 ATM - PHYSICAL LAYER

This layer consists of 2 sub-layers: the Physical Medium Dependent (PMD) which provides the actual transmission of the bits in the ATM cells, and the Transmission Convergence (TC) sub-layer which transforms the flow of ATM cells into a flow of bits and bytes for transmission over the physical medium. The TC is also responsible for generating the Header Error Check (HEC) on the ATM cell header at its originator site and uses the HEC to detect and correct errors on the receiving ATM cell. When the ATM layer has not provided a cell, the TC sends idle cells to match the speed of the physical interface.

4.3.2 ATM LAYER

The ATM-Layer interface is divided into two interfaces: these are the User-Network Inter-

face (UNI) and the Network-Network Interface (NNI). The UNI is the interface between users [customer premises equipment] and the network element, and the NNI is the interface between the network elements.

The functions of the ATM layer includes [McDysan D. E. Spohn D. L. 95]:

- Cell construction.
- Cell reception and header validation.
- Cell delaying, forwarding and copying using Virtual Path Identifier (VPI) /Virtual Channel Identifier (VCI).
- Cell multiplexing and demultiplexing using VPI/VCI.
- Cell payload type discrimination.
- Interpretation of pre-defined reserved header values.
- Cell loss priority processing.
- Support for multiple Quality of Services (QoS).
- Usage parameter control (UPC)
- Explicit forward congestion indication.
- Generic flow control.
- Connection segment and removal.

[For further information about each of the above functions, please refer to the literature McDysan D. E., Spohn D. L., "ATM: Theory And Application", McGraw Hill, 1995].
This layer also provides the transport of ATM cells between the end points of a virtual connection. The functionality of this layer divides into three levels as described below:

TRANSMISSION PATH:

At this level, the payload of the transmission is assembled [at a receiving device] and segmented [at transmitting device]. Required at this level are cell delineation which identifies the 53-bytes of each ATM cell transferred between the peer communicating ATM-entities, boundary with the screen and error control function which calculates the HEC over the PHY_SDU (Service Data Unit) before transmission.

DIGITAL SECTION LEVEL:

This level extends between the network elements [NE's] that assemble and disassemble a continuous bit or byte stream, and refers to the exchanges or physical transfer points in a network that are involved in the switching data system.

REGENERATING SECTION LEVEL:

This level regenerates the digital signal along a transmission path that is too long to be used without such generation.

4.3.3 ATM CELL

The ATM protocol is a cell-based switching and multiplexing technology. Each ATM cell has a fixed length of 53 octets (bytes) [a 5-octet cell header and a 48-octet cell payload], and is shown as in **Figure** 4.3.

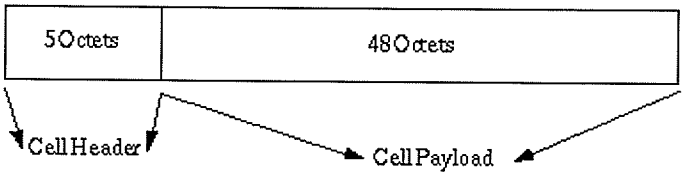


Figure 4.3 ATM Cell

In an ATM connection setup, three facets are involved: the physical link (physical transportation), the virtual path, and the virtual channel. The physical link may contain multiple virtual path links (VPs), with each VP identified by a Virtual Path Identifier (VPI). Each of these VPs may also contain a various number of virtual channel (VCs) connection links, with each VC identified by a Virtual Channel Identifier (VCI). **Figure** 4.4 presents the relationship between the physical link, the virtual path and virtual channel.

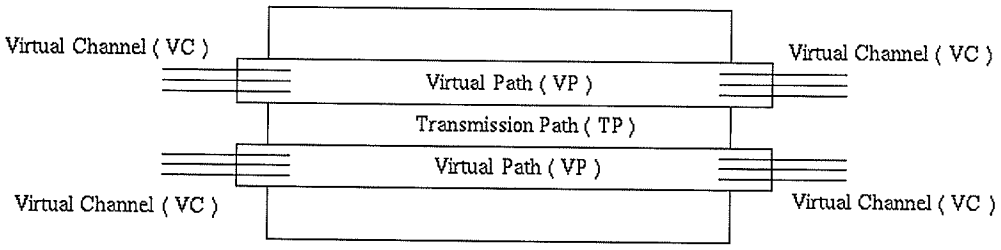


Figure 4.4 Relationship between Transmission Path (TP), Virtual Path (VP) and Virtual Channel (VC)

The ATM cell is routed from source to destination through table switching between the virtual path(s) and virtual channel(s) indicated by the VPIs and VCIs in the ATM cell header.

The 5-octet header contains information about the information in the cell payload. The ATM cell has two cell-header format, with a different type for each interface, UNI and NNI. **Figure** 4.5 shows the two formats for ATM cell headers.

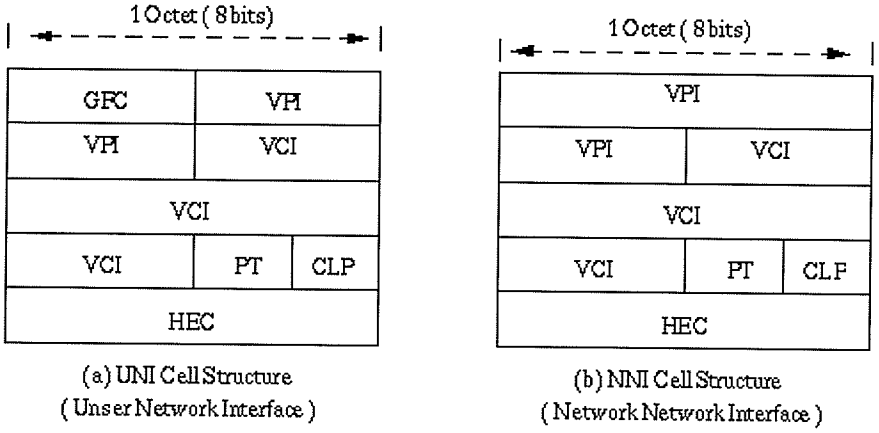


Figure 4.5 Structure of ATM Cell header.

Generic Flow Control (GFC): provides contention resolution, and simple flow control for shared-medium access arrangements at the customer premises equipment.

Virtual Channel Identifier (VCI): establishes a connection using the translation table at the switch nodes that map an incoming VCI on an outgoing VCI (the bandwidth is not utilized unless the user information is actually transmitted).

Virtual Path Identifier (VPI): establishes a virtual path connection for one or more logically equivalent VCIs in terms of route service characteristics, allowing simplified network routing functionality and management.

Payload Type (PT): indicates the type of the cell traversing the same virtual circuit. This field discriminates between the user information and the management information.

Cell Loss Priority (CLP): explicitly indicates cells of lower priority by setting this bit to

one. Cells will be delivered or dropped during high traffic sessions depending on the status of this bit.

Header Error Checker (HEC) : performs the Cyclic Redundant Checking (CRC) calculation on the first four octets of the cell header for error detection and correction (one error bit correction).

4.3.4 ATM Adaptation Layer (AAL)

This layer of the ATM protocol provides necessary functions to the services which are not provided by the ATM layer. These functions are dependent on the type of required services. **Table** 4.1 shows the classes of services of ATM, with each service class using an appropriate type of ATM Adaptation Layer.

Service Class	Description	AAL Type	Bit Rate
A	Circuit simulation. For continuous bit rate traffic with time synchronization between sender and reciever.	AAL1	constant
B	Variable bit rate services with time synchronization between sender and receiver (for variable bit rate audio and video).	AAL2	variable
C	Connection oriented data services.	AAL3,AAL4, AAL5	variable
D	Connection less data services.	AAL3,AAL4, AAL5	variable

Table 4.1 Service classes of ATM

AAL1 is a constant rate bit service. AAL2 is a variable bit rate service. AAL3 and AAL4 are a "non-assured" variable bit rate services. AAL5 is a light weight variable bit rate service.

The AAL is subdivided into 2 sub-layers: the convergence sublayer (CS) and the Segmentation and Reassembly sublayer (SAR). The convergence sublayer of the AAL detects and sends the protocol data unit to or from the higher layer. It performs the encapsulation of user data before the SAR function segments the data into 48-octet cells. The CS also sends requests and acknowledgement for retransmission of lost or wrong cells, detects loss of cells, and provides flow control information.

The Segmentation and Reassemble sublayer (SAR) segments a data stream into 53-octet ATM cells and reassembles the cells into the data stream. The SAR also performs the CRC in the ATM cell information field. [For further information on the functions of the SAR and CS sublayers of AAL, please refer to Minoli D., Vitella M. 94]

4.3.5 Quality of Services (QoS)

ATM can carry various types of media information over a single network infrastructure. This results in different qualities for different type of services, which affect each other since they all compete for traffic rights over the available network bandwidth. Each ATM service class requires a different set of QoS factor values.

The Quality of Service (QoS) of each service class are characterized by the following factors [Armitage G. J 94]: peak and average cell rate, burstiness, cell loss probabilities, end-to-end cell delay and cell delay variance. The role of each of the above factors is determined by the type of traffic. Therefore, the connections are set up for a particular type of

traffic, the resource availability allocations are required to guarantee the QoS of the traffic.

4.3.6 MANAGEMENT PLAN

This plane is subdivided in to 2 sublayers: plan management which is responsible for coordinating functions between all plans, getting the status information on each plan and informing that status to other plans; and layer management which is responsible for management functions on performance, operation, administration, resources management and parameters for each of the user plan layers. These functions also includes performance monitoring, defect and failure detection, system protection, performance information and fault localization.

4.3 ATM-BASED LOCAL AREA NETWORK (LAN)

ATM has been the heart of communication not only for wide area interconnections of a heterogeous network type, but also for Local Area Networks (LANs) as well. ATM has become the solution for the requirements of multimedia, and of a variety of new applications such as desk to desk conferences which now exist within a local network.

An ATM-Based LAN has several components, including host, ATM switches, internetworking devices (such as routers, and gateways), and interfaces to the public network.

Each ATM switch is connected to many hosts and the connections between the switch and hosts are point - to - point links. The ATM switches are connected together and the physical connection between any two switches are also point - to - point link. The following **Figure** 4.6 illustrates an example of a ATM-based LAN.

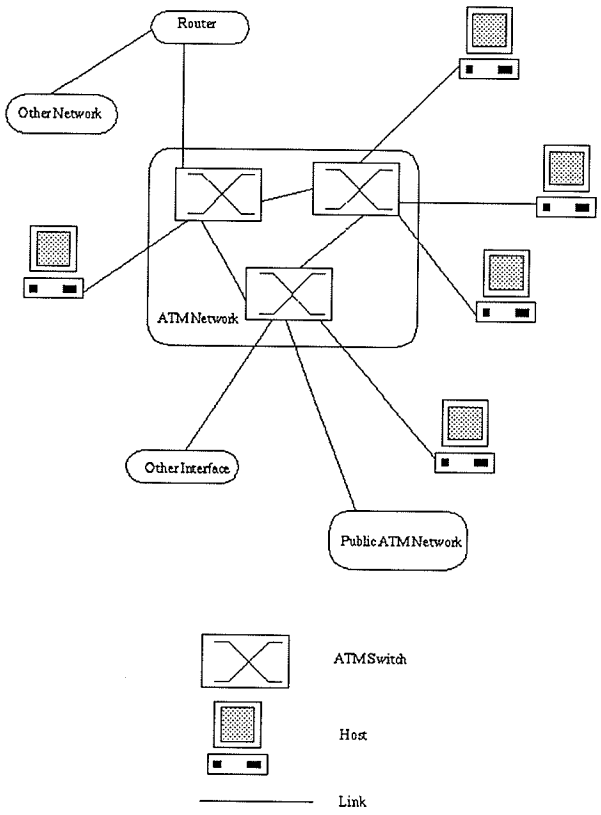


Figure 4.6 Example of ATM-Base LAN

4.4 CONLUSION

This chapter introduced the Open System Interconnection protocol with a brief discussion on each layer of the OSI protocol. This was followed by an introduction of ATM technology and brief discussion of ATM protocol concepts, such as the ATM cells and layers of the ATM protocol. With the concepts of ATM cell and fast switching technology, ATM networks can support many different types of information, and many different data

protocols can transmit over ATM networks. **Figure** 4.7 gives summary of ATM protocol multimedia communication [May M.D, Thompson P.W, Welch P.H 93]. Finally, this chapter includes an introduction to the ATM-based LAN. The next two chapters will introduce and discuss network management and ATM management.

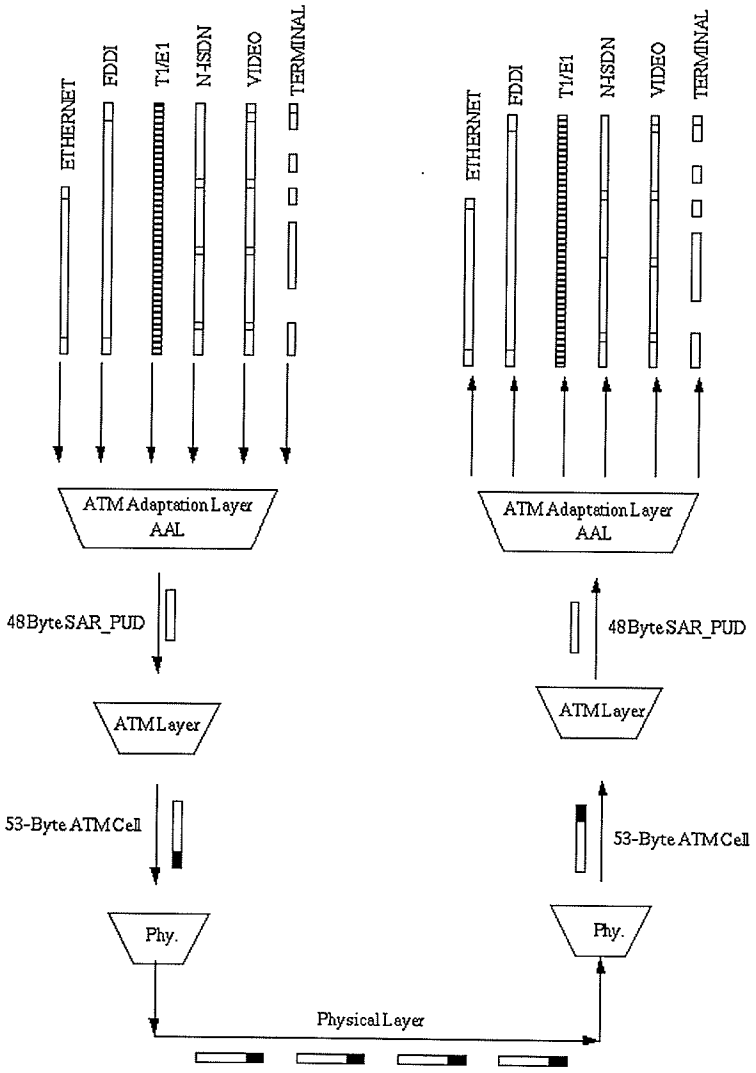


Figure 4.7 ATM Protocol Summary

CHAPTER 5
NETWORK MANAGEMENT AND SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

5.1 INTRODUCTION

Network management can be considered to be a system used to observe the operations and functionality of the network, controlling and administering the network to operate it most efficiently and properly. In general, network management deals with many aspects of the network including the following:

5.1.1 FAULT MANAGEMENT

Fault management deals with the proper operations of the system, not only that the network as a whole works properly, but that each essential component works individually and is in proper order. The management must determine a failure in any component as soon as possible after it occurs. The fault must be located and isolated in order for the system to continue to function temporarily without interference. The system is then reconfigured or modified in a way to minimize the impact from the failure of the component. The faulty unit can be replaced or repaired to restore the network to its initial state. Faults include the failure to operate correctly, or excessive error, introduced to the network.

5.1.2 Account Management

Account Management deals with the usage of the system. It controls and limits the access privileges allowed to the user to avoid the abuse of the system. The system manager may assist the user to improve the efficient use of the network.

5.1.3 Configuration Management

Configuration Management deals with the initialization and maintenance of the network, as well as additions and updates to the status of the components and their relationship during the network operation. Some of the functions can select and assign resources for a particular requested application from the user to update the status of the network.

5.1.4 Performance Management

Performance management deals with network operations, monitoring their performance to keep informed about the system activities, and controlling them to enable the network to perform at the highest level of efficiency with a limited amount of resources. The monitoring of information is for analysis and the result of the analysis is used as feedback to improve the performance of the network.

This thesis presents the design and implementation of a local area network monitoring system, therefore this aspect of network traffic monitoring is central to its concerns. The information monitored is the amount of traffic in and out of the components of the real physical system (such as hosts, switches, printers, etc.).

5.1.5 SECURITY MANAGEMENT

Security Management deals with the safe operation of the system, preventing network access from unauthorized parties by monitoring and controlling access to network system. It is concerned with the generation, distribution and storing of the encryption keys.The information of the system must be accessible to only authorized parties and the system assets must be modifiable and available to its authorized parties.

5.2 NETWORK MANAGEMENT ARCHITECTURE

The management systems of networks can have different architectures. In general within a system there is at least one host chosen to act as the manager (a centralized management system). For larger and more complex networks, many hosts can function as system managers (a distributed management system). In the case of a distributed management system, each manager is responsible for a section of the network or for an aspect of the management system (as briefly described in the previous section).

Each of the network components (network elements) contains a collection of software that performs the local information storage and responds to requests from the system manager. The information stored within each network element known as the Network Management Entity [NME], consists of the status of the element, and information for management purposes.

The system manager contains a software application that allows the authorized user to manage the network system through an operator interface. The software application, or network management agent [NMA], can generate information requests to the network ele-

ments and obtain information for further analysis. The system manager also allows the authorized user to set and reconfigure the network as desired. **Figure** 5.1 suggests an example of a network system with one manger host.

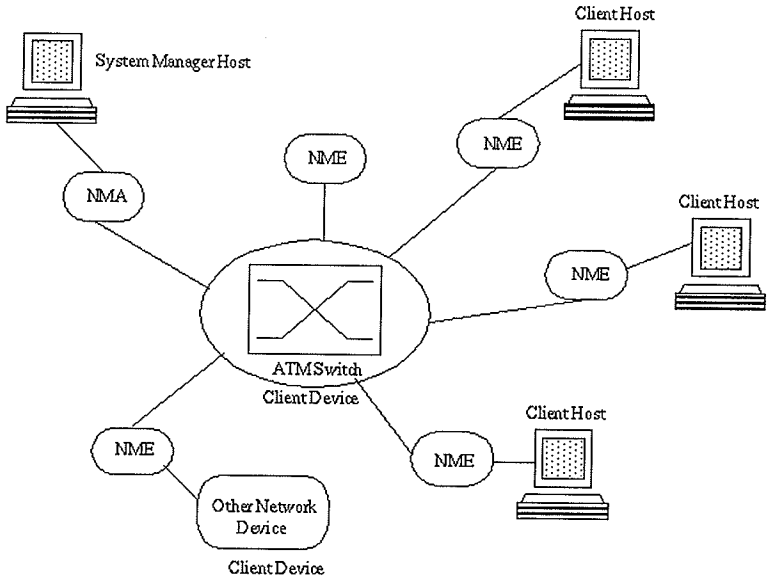


Figure 5.1 Example of network management system.

The example above depicts a local area network consisting of a number of hosts, an ATM switch and other devices. One of the hosts acts as the system manager and the other elements of the network are clients that are managed by the system manager. The system manager host can create information requests on other elements of the network, such as the current status of the other hosts, the number of data traffic in and/out through the host interface, or the number of ATM-cells with errors in cell headers going through one port of the ATM switch. Not only capable of retrieving information from the network elements, the system manager host can also send commands to set the status of the client elements,

such as the number of paths on the port of the ATM-switch, in order to improve the efficiency of network performance.

5.3 SIMPLE NETWORK MANAGEMENT PROTOCOL [SNMP]

Simple Network Management Protocol [SNMP] is a means for managing TCP/IP on an Ethernet network. Since 1989, it has been broadened to become completely independent of TCP/IP and can be implemented over any network media and protocol suite.

The concept of SNMP includes manager agent(s) and a management information base [MIB]. The agent software contains management data about the status and statistical information of the device and returns this data to the manager as requested. The manager has software to make requests for the various types of information from the agent that the manager may wish to know. **Figure** 5.2 presents a simple SNMP protocol. The system may consist of more than one host acting as the system manager for the distributed system. The system manager sends information requests to the client and gets response from the client.

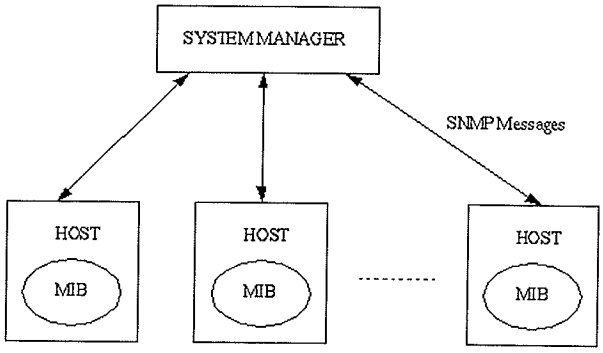


Figure 5.2 SNMP Protocol.

The three major requests that the manger makes to the agents are "*Get*" to obtain scalar-information from the agent, "*Set*" to update a scalar object value in an agent, and "*Trap*" to notify an agent of an unexpected event, such as a failure or a system restart.

5.3.1 MIB STRUCTURE [RFC 1155]

The Management Information Base [MIB] contains a set of information on the resources for each node in the network. Each piece of information for the node is called a managed object. Since these managed objects can be retrieved and set to scalar values, they are defined in terms of a primitive type-value.

5.3.2 MANAGED OBJECTS

The MIB is developed based on the syntax of Abstract Syntax Notation One ASN.1. Its architecture is a hierarchy tree starting with a root. A part of a MIB tree is shown as in **Figure** 5.3.

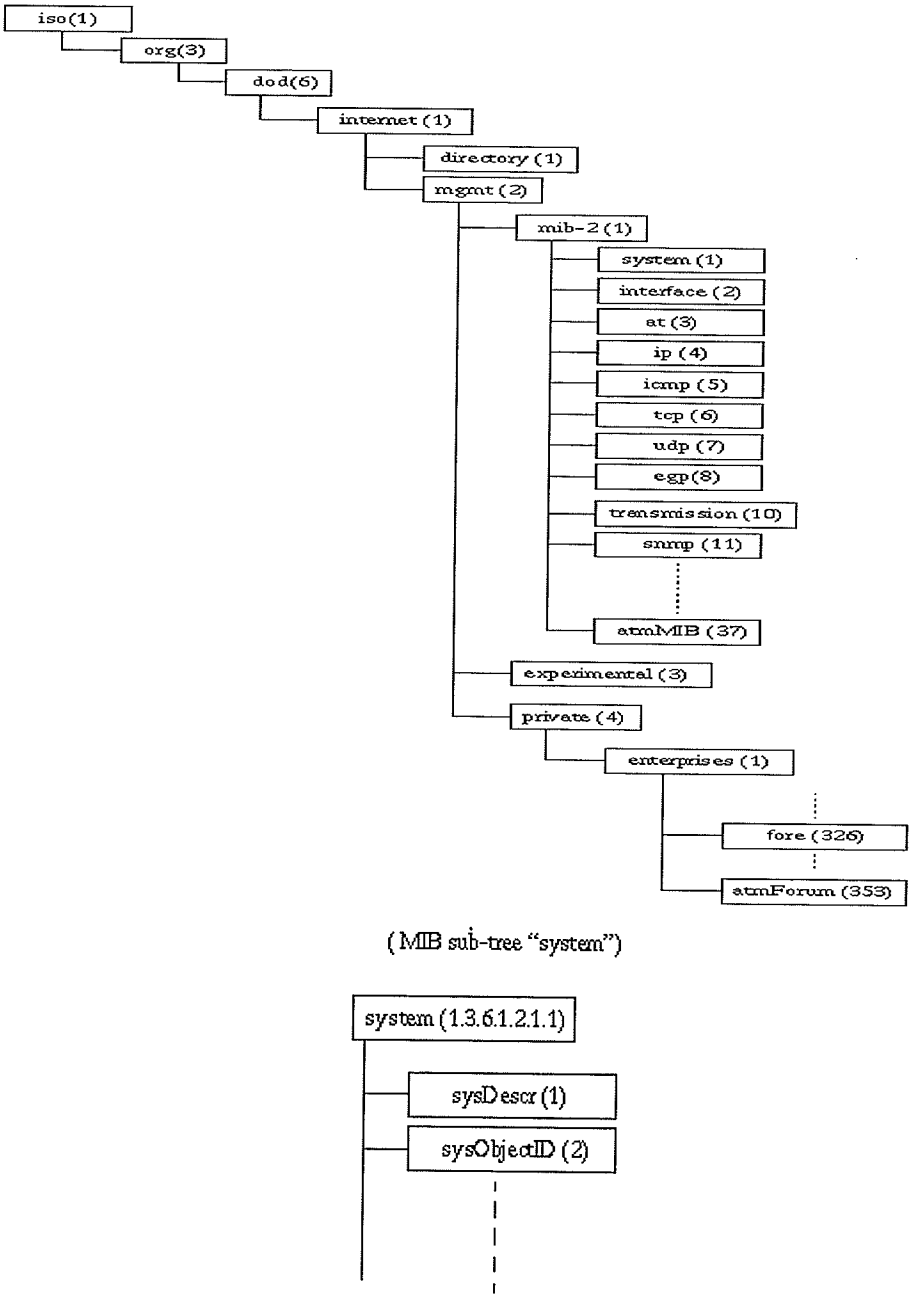


Figure 5.3 MIB Tree.

The managed objects, or scalar-valued objects such as "sysDescr", "sysObjectID" in the sub-tree "system", are the leaves of the tree. Only the leaves in the MIB tree are accessible

for system managing purposes.

Each managed object is uniquely defined within the MIB tree. The managed object is identified by the OBJECT IDENTIFIER type of ASN.1. The managed object identification is a series of numbers or text starting from the root of the MIB tree and going down the tree until the managed object is reached. Examples of the identifications for managed objects "sysDescr" and "sysObjectID" are given in **Figure** 5.4.

```
sysDescr OBJECT IDENTIFIER ::= { iso org(3) dod(6) internet(1) mgmt(2) system(1) 1}
or sysDescr OBJECT IDENTIFIER ::= { 1.3.6.1.2.1.1 }

sysObjectID OBJECT INDENTIFIER :: = { iso org(3) dod(6) internet(1) mgmt(2) system(1) 2 }
or sysObjectID OBJECT INDENTIFIER :: = { 1.3.6.1.2.1.2}
```

Figure 5.4 Example of managed object identification.

5.3.3 ASN.1 MANAGED OBJECTS

Each managed object is defined by ASN.1 and consists of 5 fields: Object, Syntax, Definition, Access and Status. In this section, each fields of the managed object is briefly described.

5.3.3.1 Object

This is a textual name defining the name of the managed object which corresponds to the object identifier. For exampled, the "sysDescr" is a managed object which describes the system.

5.3.3.2 Syntax

The syntax is the abstract syntax for the object. It corresponds to the structure of the data of which managed object can hold. The primitive object types are: INTERGER, OCTET STRING, OBJECT IDENTIFIER and NULL. Some other defined types can be derived from the primitive type. The following define the most frequently used types for the managed objects in the MIB:

- ipAddress: 32-bit internet address, represented by an OCTET string of length 4 in network byte-order.

- Counter: 32-bit non-negative integer which is monotonically increasing until it reaches

its maximum (2^{32} -1 = 42494967295) and then wraps around and starts from 0.

- Gauge: 32-bit non-negative integer that may increase or decrease. The value latches

when it reaches its maximum and remains latched until it is reset.

-TimeTicks: non-negative integer counting the time in hundreds of second since some epoch.

- AtmAddress: an octet string of size of 8 byte.

Other types can be defined from the primitive types as desired.

5.3.3.3 Status

The status for the managed object can hold one of these values: mandatory, optional or obsolete. The value of "Mandatory" means that the object should be implemented for every device. "Optional" indicates that the managed object may be necessary. "Obsolete"

indicates that the managed object has been replaced by new a managed object.

5.3.3.4 Access

This indicates accessibility to the object. Accessibility levels are: read-only, read-write, write-only, non-accessible. A managed object with access level of "read-only" allows the user to only retrieve the value of the managed object. A "read-write" managed object allows the user both read ability and write ability on the value of the managed object. A "non-accessible" managed object does not allow the user to do any operation on its value.

5.3.3.5 Description

This is a text description of an object within the system, describing, for instance, the purpose of the managed object and what type of value the managed object holds. An example of a managed object declaration is as follows:

```
numberOfPorts OBJECT-TYPE
        SYNTAX          INTEGER
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                        "The number of ports on this ATM switch."
        :: = {portGroup 1 }
```

5.3.4 TYPE CONSTRUCTION

Data structures can be constructed using a combination of other primitive types. A type list can be defined as:SEQUENCE {<type 1>, <type 2>,...., <type N>} where <type N> is a primitive type.

The managed object of the MIB are constructed in two forms, the simple managed object

mentioned previously and table managed objects. A table managed object is a two-dimension array [a table]. Each row represents the information for one instance of the managed object. For each row, the entries can be a simple managed object or a table itself. The table can be constructed as follow: SEQUENCE OF < Entry > where <Entry> is a type - list. An example of a table is shown in **Figure** 5.5

```
atmLayerTable OBJECT-TYPE
        SYNTAX          SEQUENCE OF AtmLayerEntry
        ACCESS          not-accessible
        STATUS          mandatory
        DESCRIPTION
                        "A table of ATM layer information."
        ::= { atmLayerGroup 1 }

atmLayerEntry OBJECT-TYPE
        SYNTAX          AtmLayerEntry
        ACCESS          non-accessible
        STATUS          mandatory
        DESCRIPTION
                        " A table entry contianing ATM layer information"
        INDEX           { atmInterface }
        ::= { atmLayerTable 1 }

AtmLayerEntry ::=
            SEQUENCE {
                atmInterface                INTEGER,
                atmTransmittedCells         Counter,
                atmReceivedCells            Counter,
                atmOutOfRangeVPIs           Counter,
                atmUnconnectedVPIs          Counter,
                atmOutOfRangeVCIs           Counter,
                atmUnconnectedVCIs          Counter
            }


 atmInterface OBJECT_TYPE
            SYNTAX          INTEGER
            ACCESS          read-only
            STATUS          mandatory
            DESCRIPTION
                    "This object identifies the ATM interface."
            ::= {atmLayerEntry 1}
 atmTransmittedCells        OBJECT_TYPE
            SYNTAX          Counter
            ACCESS          read-only
            STATUS          mandatory
            DESCRIPTION
                            "The number of ATM cells transmitted."
            ::= {atmLayerEntry 2}
```

```
atmReceivedCells OBJECT_TYPE
        SYNTAX          Counter
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                "The number of ATM cells received."
        ::= {atmLayerEntry 3}
atmOutOfRangeVPIs OBJECT_TYPE
        SYNTAX          Counter
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                "The number of ATM cells received with VPI values that were
                out of range."
        ::= {atmLayerEntry 4}
atmUnconnectedVPIs OBJECT_TYPE
        SYNTAX          Counter
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                "The number of ATM cells received with VPI values that
                were in range but not correspond to active connections."
        ::= {atmLayerEntry 5}
atmOutOfRangeVCIs OBJECT_TYPE
        SYNTAX          Counter
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                "The number of ATM cells received with VCI values that were
                out of range."
        ::= {atmLayerEntry 6}
atmUnconnectedVCIs OBJECT_TYPE
        SYNTAX          Counter
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                "The number of ATM cells received with VCI values that were
                in range but not correspond to active connections."
        ::= {atmLayerEntry 7}
```

Figure 5.5 Example of table of managed objects.

The "atmLayerTable" consists of "SEQUENCE OF" of "atmLayerEntry" [contains zero or more atmLayerEntry type]. Each "atmLayerEntry" is a row for this table and each row is a SEQUENCE of scalar-valued entries. This SEQUENCE is fixed (predefined). The atmLayerEntry is a row consisting of seven scalar-valued entries which are: "atmInterface", "atmTransmittedCells", "atmReceivedCells", "atmOutOfRangeVPIs", "atmUnconnectedVPIs", "atmOutOfRangeVCIs", and "atmUnconnectedVCIs". Each of these entries is defined as a simple object.

The INDEX part of the entry definition indicates the specific scaler-value needed to distin-

guish the rows in the table. In above example, the INDEX happens to be the "atmInterface". Each ATM host or device may have many interfaces and each interface has an ATM Layer, so this "atmInterface" managed object differentiates among the many interfaces of the device. In the table, only the scalar-valued entries can be accessible.

5.4 CONCLUSION

This chapter introduced management support for a network. The Simple Network Management Protocol (SNMP) was presented through a discussion of the Management Information Base (MIB), its structure, and on the structure of the managed object expressed with Abstract Syntax Notation One (ASN.1). Some examples have been given to demonstrate these concepts of the managed object structures in both table and simple managed object formats. In the next chapter, the MIB for specific systems and several versions of the MIB for ATM-Based network are discussed.

CHAPER 6
ASYNCHRONOUS TRANSFER MODE (ATM) NETWORK MANAGEMENT

This chapter introduces the MIB structure for ATM network management. The ATM-related MIB has been developed by different vendors and researchers leading to different ATM - sub MIBs existing in the SNMP MIB tree. These MIBs contain the ATM-related managed objects that can be accessed by the Simple Network Management Protocol (SNMP) for monitoring and influencing ATM system behaviour. ATM managed objects are basically used to the manage ATM interfaces, VP,VC, ATM cross connections, and AAL entities and connections.

6.1 RECOMMENDED FOR COMMENT 1695 [Developed by Network Working Group 1994]

The RFC 1695 arranged the ATM managed objects in to 6 groups.

(1) ATM interface configuration group.
(2) ATM interface DS3 PLCP group.
(3) ATM interface TC sublayer group.
(4) ATM interface virtual link (VPL/VCL) configuration.
(5) ATM VP/VC cross connect group.
(6) AAL5 connection performance group.

The following sections briefly describes each of the above groups. **Figure** 6.1 shows the atmMIB with its main tables and entries.

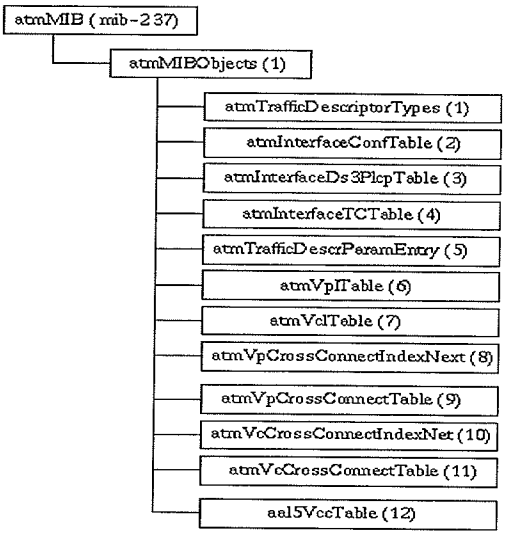


Figure 6.1 The sub-tree for atmMIBObjects managed object

The managed objects that describe the traffic descriptor types and parameters are shown in **Figure** 6.2.

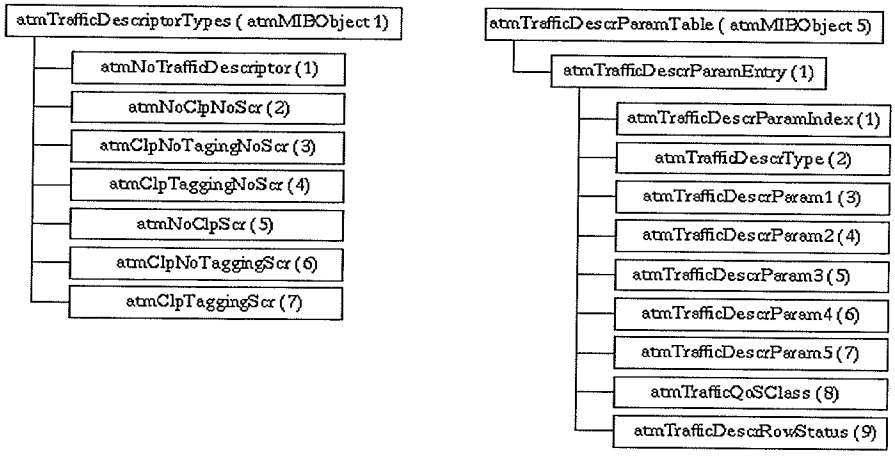


Figure 6.2 The sub-tree for traffic descriptor types and parameters managed objects

6.1.1 ATM Interface Configuration Group

This group of the ATM management system contains the managed objects with information on the ATM cell layer and the configuration of the local ATM interface. This information includes port identifier, interface speed, number of transmitted and received cells, number of cells with uncorrectable HEC errors, physical transmission type, operation and administration status, active VPI/VCI fields and maximum number of VPCs [Virtual Path Connection] and VCCs [Virtual Channel Connections]. The **Figure** 6.3 shows the sub-tree for managed objects of this group.

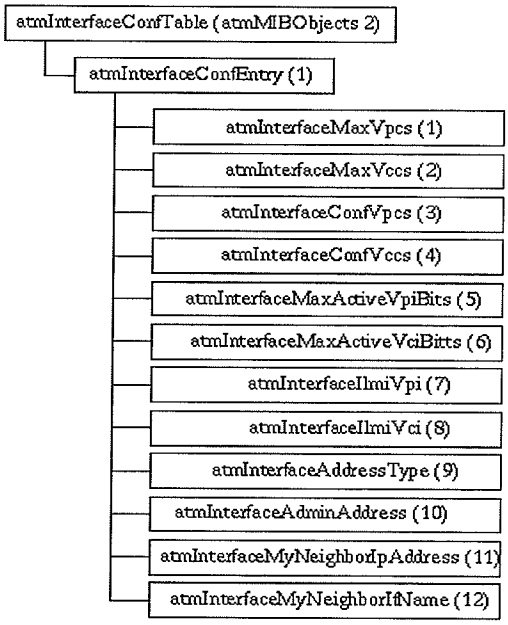


Figure 6.3 Interface configuration group.

6.1.2 ATM Interface DS3 PLCP and TC Sub-Layer Groups

These two groups provide the physical performance statistics for the DS3 or SONET transmission path, including the statistics on the bit rate error rate and errors per second.

The managed objects for these two groups are shown in **Figure** 6.4.

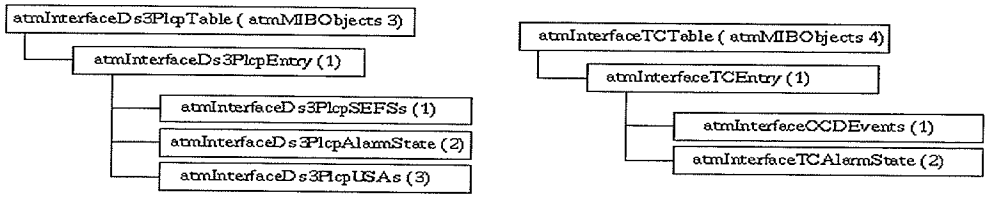


Figure 6.4 Interface DS3 and TC Sub-layer groups.

6.1.3 ATM Interface Virtual Links & Cross Connections Groups

These two groups provide the management of the ATM virtual path and virtual connection, virtual links and cross connections. The difference between the virtual link group and the cross connect group is that the cross connect group is implemented on the switch and network only, and the virtual link group is implemented on the end system (e.g. host) as well as the switch and network. **Figure** 6.5 displays the managed objects for virtual links (Virtual Path (VP) and Virtual Channel (VC)) groups.

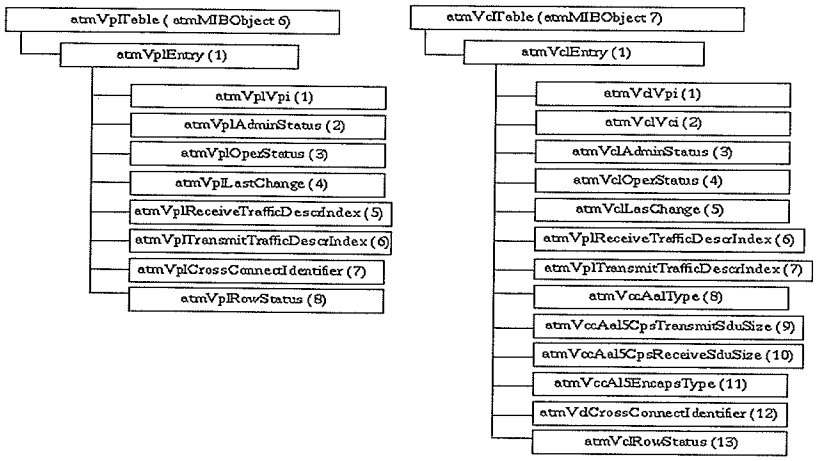


Figure 6.5 Interface virtual link groups.

Figure 6.6 displays the managed objects for the virtual link (Virtual Path (VP) and Virtual Channel (VC)) cross connection groups.

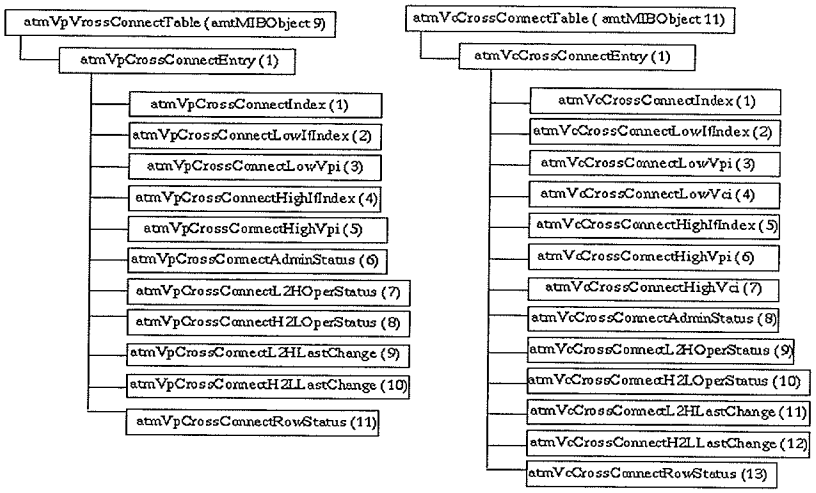


Figure 6.6 Interface virtual link cross connection groups.

6.1.4 AAL5 CONNECTION PERFORMANCE GROUP

This group contains managed objects with information on the number of received cells, number of transmitted cells, the number of octets passed to the AAL layer, the number of octets received from the AAL5 layer and the number of errors in AAL CPCL PDU. The **Figure** 6.7 displays the managed objects for the AAL5 group.

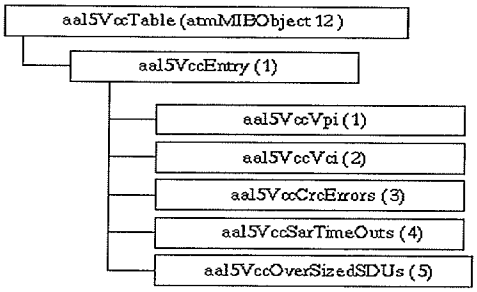


Figure 6.7 AAL5 group.

6.2 INTERM LOCAL MANAGEMENT INTERFACE MIB [Developed by the ATM Forum]

The Interm Local Management Interface [ILMI] MIB arranges the ATM managed objects into the following two groups or sub trees, "atmForumAdmin" and "atmForumUni". The "atmForumAdmin" sub-tree contains managed objects concerning with the information arranged in the tree sub-groups, "atmFTransmissionTypes", "atmfTrafficDescrTypes" and "atmfMediaTypes". The "atmForumUni" sub-tree contains information about the user network interface and these managed objects are arranged in the following groups:

(1) Physical Layer Group.
(2) ATM Layer Group.
(3) ATM Layer Statistic Group.
(4) Virtual Path Connection Group.
(5) Virtual Channel Connection Group.

Figure 6.8 shows the sub tree of the ATM Forum.

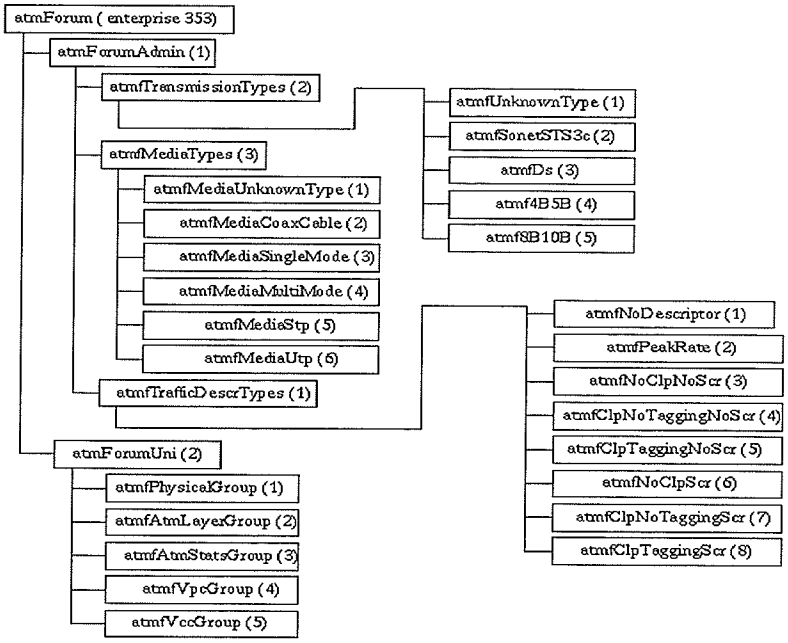


Figure 6.8 ATM Forum MIB tree.

6.2.1 PHYSICAL LAYER

This group contains the managed objects for the information related to each of the ATM physical interfaces. **Figure** 6.9 shows the managed objects defined in the Physical Layer group of the ATM Forum.

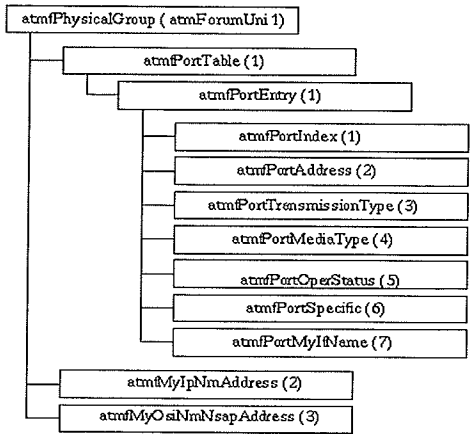


Figure 6.9 Physical Layer Group

6.2.2 ATM LAYER GROUP

This group contains information on each ATM layer of the managed ATM device. The managed objects include the number of maximum and current value of the virtual path, the virtual channel, etc. **Figure** 6.10 shows the managed objects defined in the ATM Layer group of the ATM Forum.

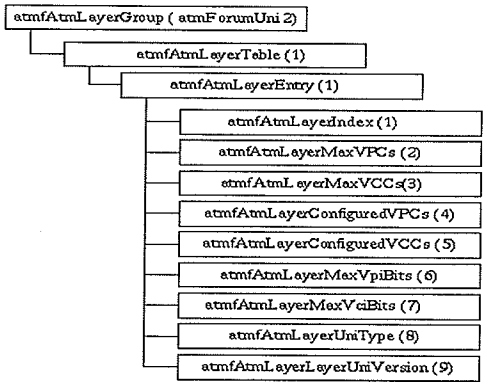


Figure 6.10 ATM Layer Group

6.2.3 ATM LAYER STATISTICS GROUP

This group contains managed objects with statistic information concerning of the performance of the ATM device, such as the number of cells sent or received by the ATM Layer entity, and the number of cells dropped on the receiving site due encountered errors. **Figure** 6.11 shows the managed objects for the ATM statistics group.

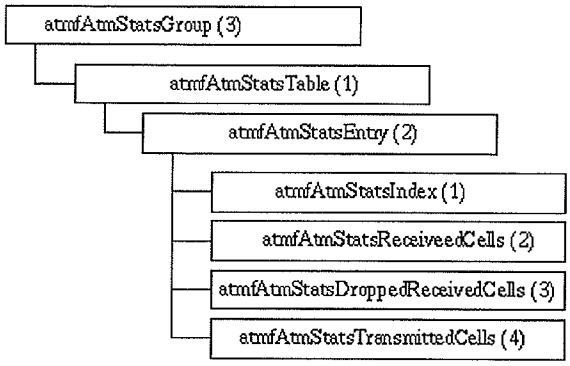


Figure 6.11 ATM Statistic Group.

6.2.4 VIRTUAL PATH/ VIRTUAL CHANNEL CONNECTION GROUPS

These groups contain information about the virtual path/virtual channel. For each virtual path, virtual channel, the managed objects include: the port index that indicates the physical port supporting this virtual path/virtual channel, the VPI/VCI, the operational status for this virtual path/virtual channel reflecting the end to end state for each individual link, traffic descriptions for both the transmitting and the receiving direction. **Figure** 6.12 shows the MIB sub-tree containing the managed objects for virtual link connections (virtual path link and virtual channel link).

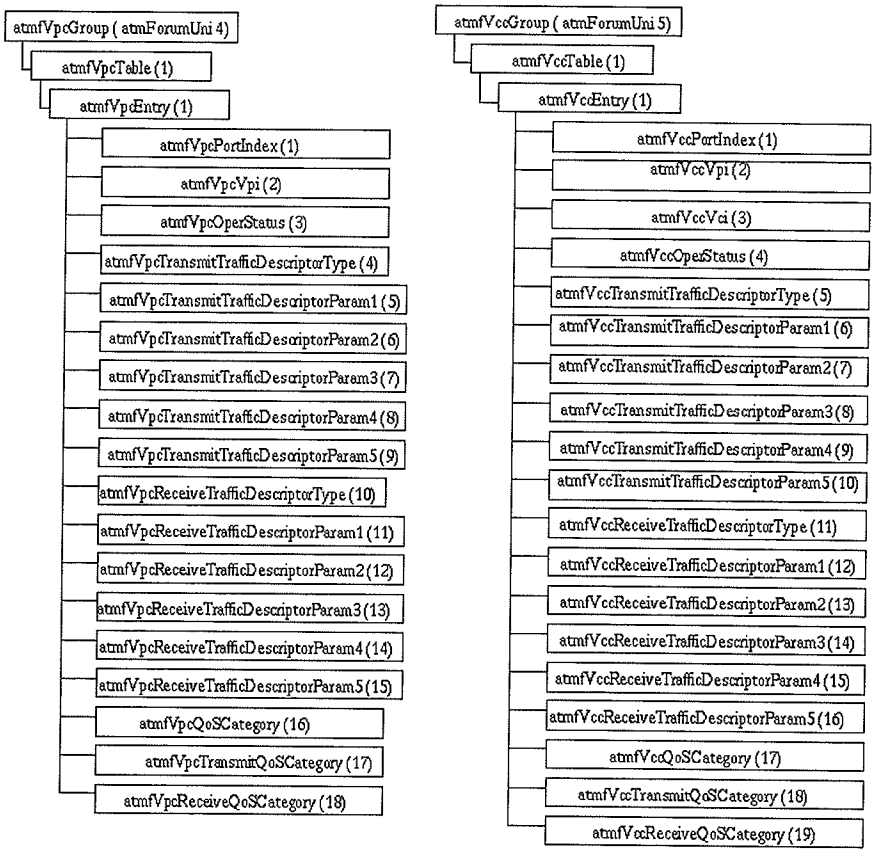


Figure 6.12 ATM Virtual Link Groups

6.3 FORE-ATM

In this section, the managed objects for the FORE-ATM device will be presented. The network management developed in this thesis is for a FORE-ATM based network.

6.3.1 Introduction To Fore-ATM ASX-100 System

FORE is the vendor of the ATM switch ASX-100 with Switched Virtual Circuit (SVC) signalling. The ASX-100 which supported IP-based services operating over an ATM -

SVC back bone using a prestandard protocol has been upgraded to standard ATM SVC signalling [McDysan D. E., Spohn D. L. 95]. **Figure** 6.13 presents an example of a FORE ASX-100 switch based local area network with three interconnected ASX-100 switches via. PVC VPCs using the Simple Protocol for ATM Network Signalling (SPANS) [McDysan D. E., Spohn D. L. 95]. The SPANS SVC is an ATM signalling protocol with a tunnelling capability that can be evoked from standard UNIX software socket calls for IP.

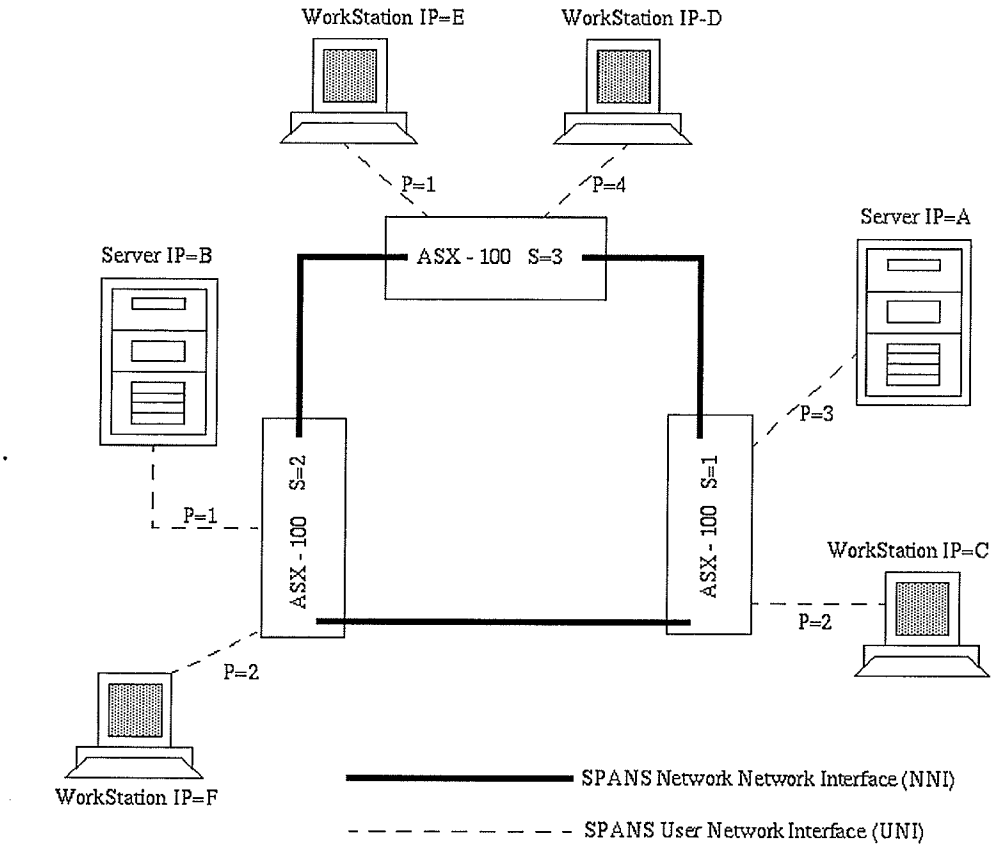


Figure 6.13 ASX-100 Switch Based LAN

In the above **Figure** 6.13, each switch and port of a switch is defined by a unique number S and P, respectively. Each of the devices connected to the switch, in this case the servers

and workstations, are addressed by the combination of the switch number S and the port number P of the switch. These devices run UNIX OS and have IP addresses.

Each server/workstation interface is manufactured by FORE with the device driver hardware and software containing the functions displayed in **Figure** 6.14 [McDysan D. E., Spohn D. L. 95].

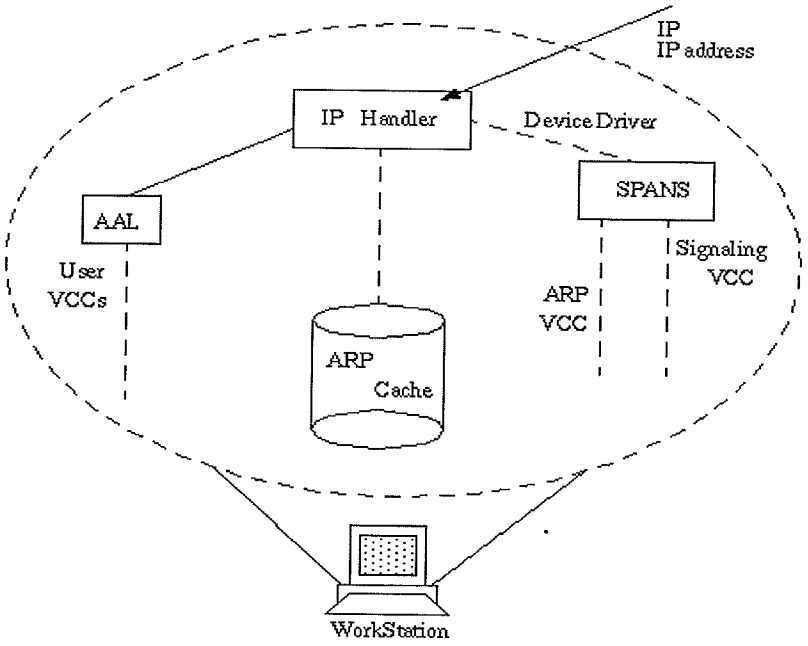


Figure 6.14 Fore Device Driver Hardware and Software Functions

As shown above, when the workstation equipped with the FORE System driver software at IP address A, attempts to make a data transmission to the destination IP address of B, some IP packages are passed to the driver software. The Address Resolution Protocol (ARP) cache defines the VPI/VCI for each IP address in the network. The IP-Handler checks if the VPI/VCI exists for the destination IP address B. If it does, the packets can be transmitted using the VPI/VCI over the ATM -network to the desired destination. If the

VPI/VCI does not exist, the workstation generates an ARP request for that particular IP address B, and this request is broadcasted to all the devices. The device with this IP address responds to the ARP request with the switch number (S) and port number (P) for that IP address. Only the device with IP address A receives the response. The device with IP address A then set ups the ATM connection with the destined (S,P).

6.3.2 THE FORE MIB TREE

The FORE-ATM system has its managed objects arranged in two major groups, the ATM-Adapter and the ATM-Switch groups. The MIB-tree is shown in **Figure** 6.15 [For the full MIB tree of the Fore system, please refer to the documents: FORE Systems, fore_adapter.mib, v 1.85, 1993-1994, and FORE Systems, fore_switch.mib, v 1.320, 1993-1994]. In this section, not all of the groups and managed objects in the FORE-MIB tree are presented, only those that are considered in the implementation of the traffic monitor.

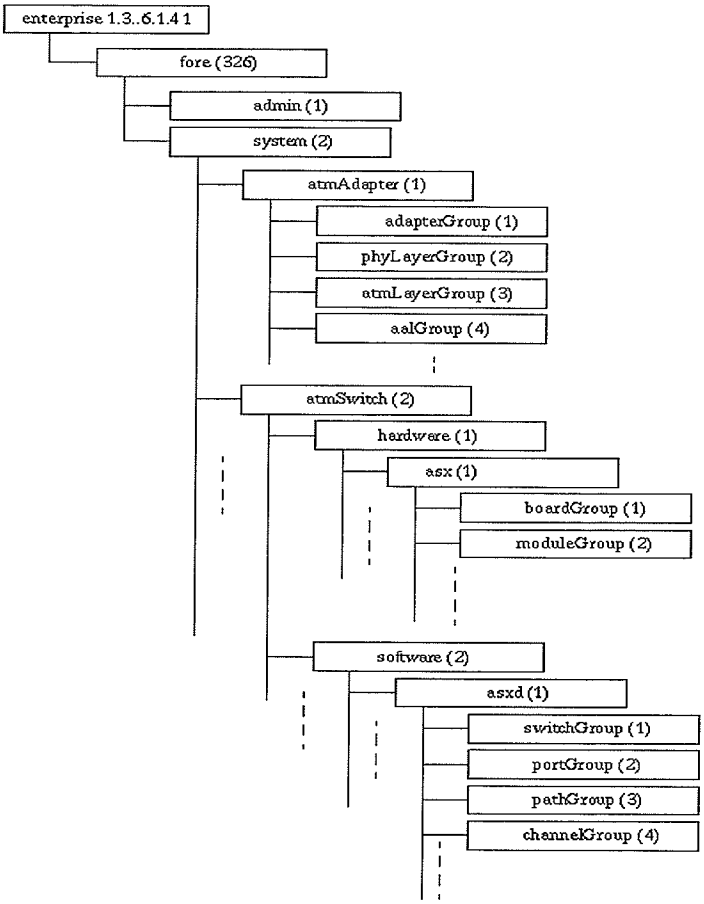


Figure 6.15 MIB tree for the FORE System.

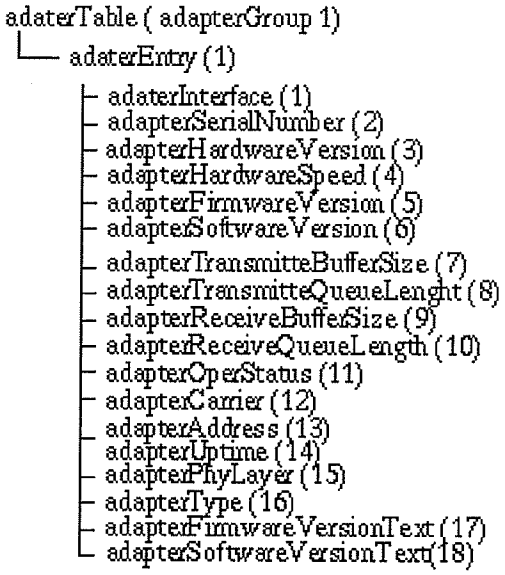
6.3.2.1 ATM Adapter Group

This group contains many sub-groups and each group contains managed objects concerning a layer of the ATM protocol.

Adapter Layer Group (adapterGroup)

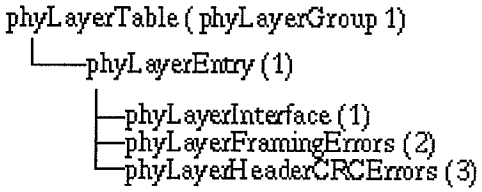
This group contains a table of managed objects concerning the information on the physical hardware description of the adapter. The managed objects contained in the adapter layer

tables are:



Physical Layer Group (phyLayerGroup)

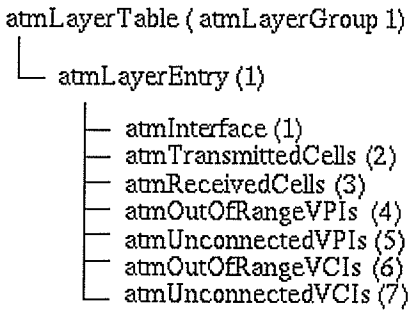
This group contains a table of managed objects with the number of errors encountered at the physical layer. These managed objects are:



ATM Layer Group (atmLayerGroup)

This group contains a table of managed objects concerning the ATM cells passing through the ATM layer as well as information about cells with errors detected at this layer. The

managed objects included in this table are



At the ATM layer, the received ATM cells are differentiated into type, such as AAL4 cell type, AAL5 cell type and AAL0 cell type (user-defined type), and the cells are routed to the appropriate ATM Adaptation Layer. Also at this layer, cells are checked for errors and are rejected if an error is found. The cell transmission on this layer and how the managed objects for this table are defined from the receiving view is depicted as in **Figure** 6.16.

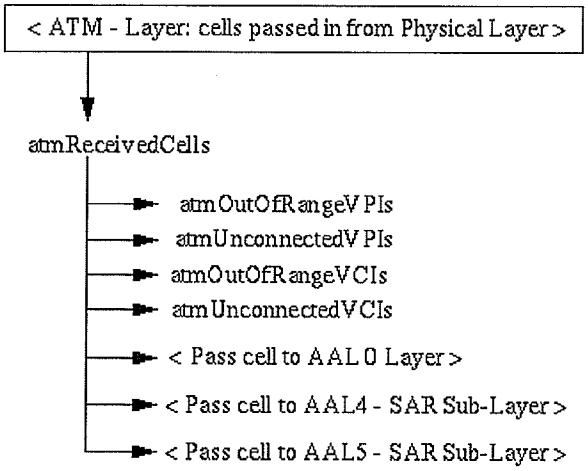


Figure 6.16 ATM cells at ATM Layer

ATM Adaptation Layer (aalGroup)

This group is sub-divided into three tables, the "aal5Table", "aal4Table" and "aal0Table". Each table contains managed objects with the information for each type of ATM cell. The AAL is divided into two sub-layers as shown in **Figure** 6.17. These sublayers are the Convergent Sub-layer (CS) and the Segmentation and Reassembly sub-layer (SAR), as mentioned in Chapter Four.

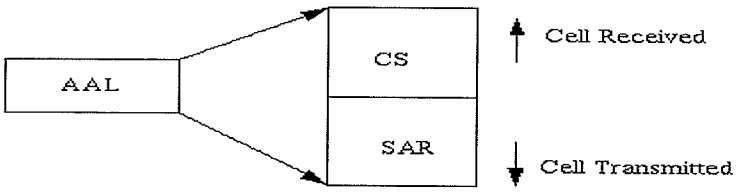
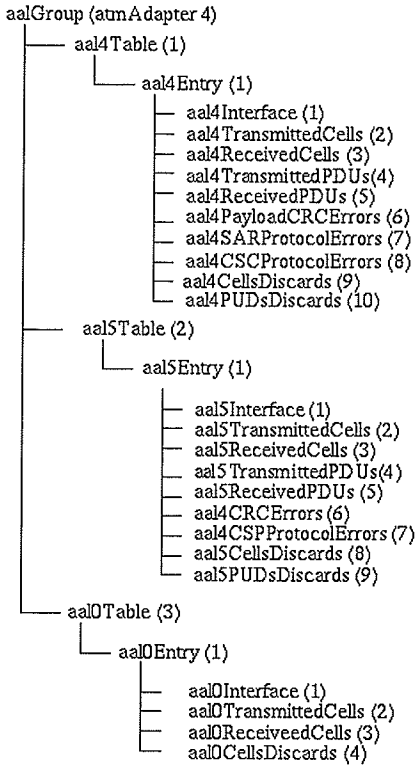


Figure 6.17 Two sub layers of AAL

The managed objects for the ATM Adaptation Group are:



AAL4 Sub-layer

The AAL4 sub-layer of the ATM Adaptation Layer supports (connection oriented and connectionless) variable bit rates. The received AAL4 cell is error-checked and passed within the AAL4 layer to the receiver, as in the diagram in **Figure** 6.18. This diagram also explains how the managed objects for the "aal4Table" are defined.

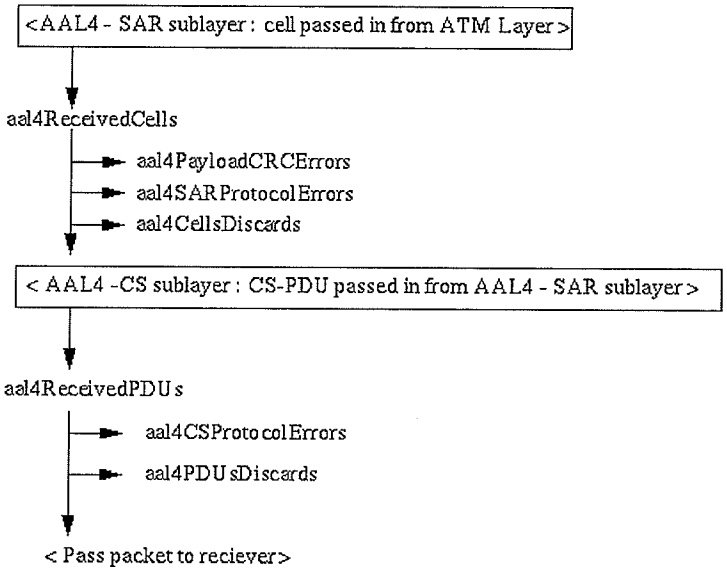


Figure 6.18 Data at the AAL 4 Layer

The ATM cells are multiplexed at the AAL4 sub-layer, as shown in **Figure** 6.19 [McDysan D. E., Spohn D. L. 95]. The ATM cells received from ATM layer by the AAL4 layer go into the SAR sub-layer, where they are passed on or rejected for errors, such as cells with payload CRC (Cyclic Redundant Check) errors, cells with SAR protocol errors, and those cells classified as discarded. The error free cells are multiplexed into SAR-PDU packets and passed to the CS sub-layer. The SAR-PDU cells enter the CS sub-layer and are rejected if they are found with errors, such as CS protocol errors, and discarded at this CS-sub-layer. The error free SAR-PDUs are then multiplexed into CS-PDUs, and the PDU Header, PAD, and Trailer are stripped at the end of the layer and the data packet is send to the receiver. The format of the SAR-PDUs and CS-PDUs are presented in **Figure** 6.19. The transmitting data is in the opposite direction of the receiving data process.

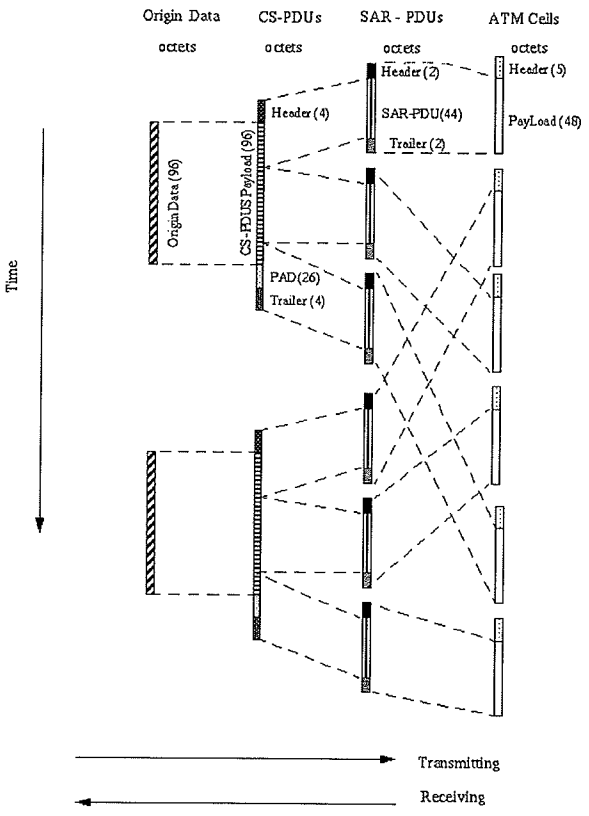
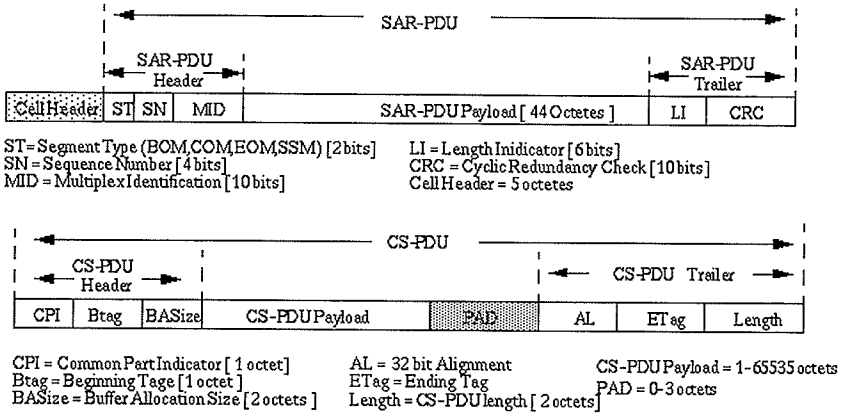


Figure 6.19 Example of cell multiplexing in AAL 4 Layers

The formats for the SAR-PDUs and CS-PDUs are as follows:



AAL5 Sub-layer

The received AAL5 cell is error-checked and passed within the AAL5 layer to the receiver as in **Figure** 6.20. This diagram also explains how the managed objects for the "aal5Table" are defined. The Common Part (CP) of AAL5 supports both connection-oriented and connectionless variable bit rate traffic.

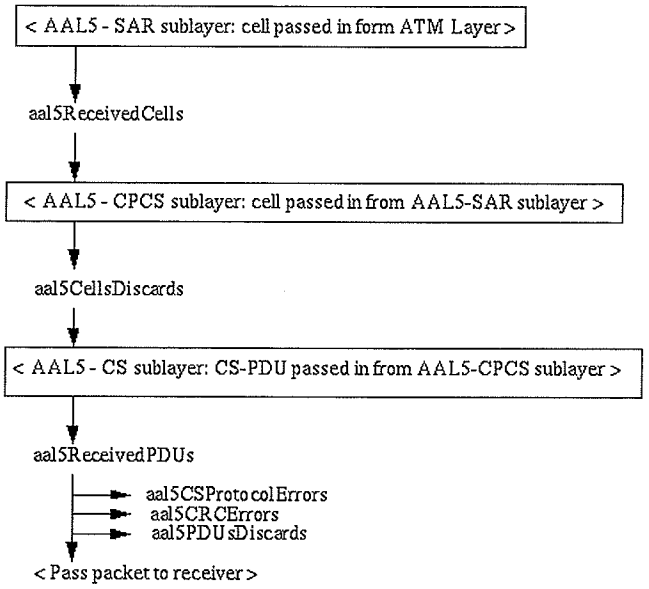


Figure 6.20 Data at the AAL 5 Layer

The AAL5 cells received at the AAL5-SAR layer from the ATM layer are multiplexed into the SAR-PDUs that are then passed into the CPCS sub-layer where the cell may have been discarded in the course of dropping partially reassembled PDUs. The correct PDUs are then passed to the AAl5-CS layer where the incorrect cells, due to CS-protocol errors and CRC errors, or those cells discarded by the AAL are dropped and the correct cells are multiplexed and reach the receiver. The AAL5 cell multiplexing is shown in **Figure** 6.21

[David E. McDysan, Derren L. Spohn].

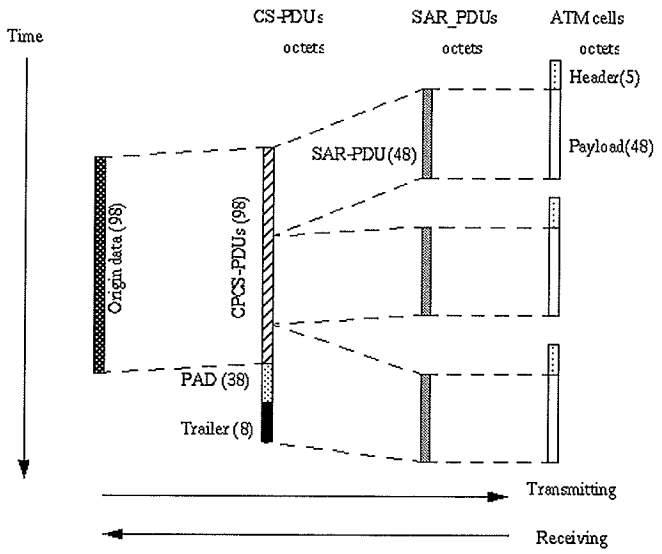
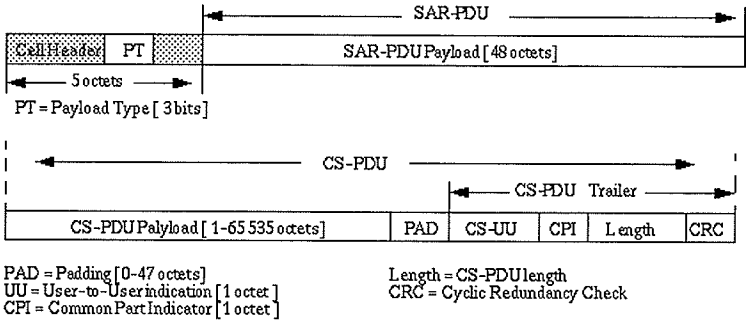


Figure 6.21 Exampled of multiplexing in AAL 5 Layer.

The formats for the SAR-PDUs and CS-PDU of the AAL5 are as follows:



AAL0 Sub-layer

This adaptation layer is not standard. It is either a user defined adaptation layer or the AAL-Null that is available in the FORE-ATM adaptation layer interface. The AAL0 cells

received from the ATM layer are checked for errors due to the dropping of partially reassembled PDUs. **Figure** 6.22 depicts the flow of the ATM cells as received from the ATM layer by the AAL0 layer and transmitted to the receiver.

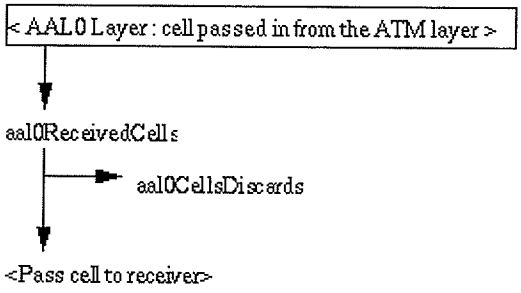


Figure 6.22 Cells in AAL0 Layer.

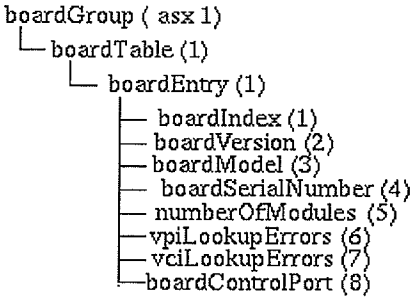
The received ATM cells are checked for errors occurring in the case of dropping partially reassembled PDUs. The error free cells are passed to the receiver.

6.3.2.2 ATM Switch Group

The "atmSwitch" MIB sub-tree of the FORE-MIB tree contains managed objects about the information of the ATM switch. The information relevant to the implementation of the network traffic monitor is the managed objects for the board group, module group, port group, path group, channel group, and the switch group of the FORE-ATM switch. The following sections briefly introduce some of the managed object groups for the FORE_ATM switch. [For the whole MIB tree of the FORE - Switch, please refer to the vendor literature.]

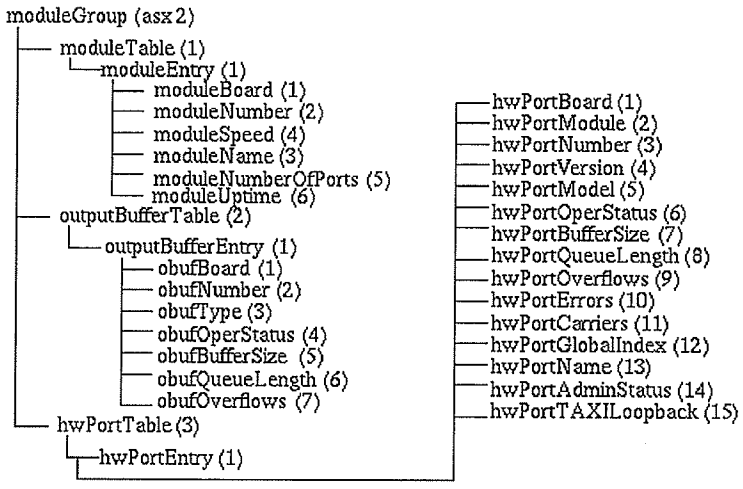
The Board Group

The board group contains a table of managed object for the information on the hardware and configuration of each board of the ATM switch. The managed objects of this group are:



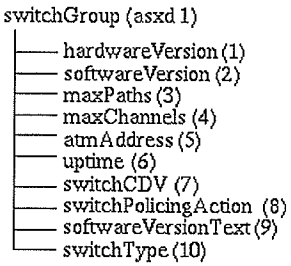
The Module Group

The Module Group contains three tables of managed objects, the "moduleTable", the "outputBufferTable" and the "hwPortTable". Each table contains managed objects for the information of the module of the switch. The managed object for this group are:



The Switch Group

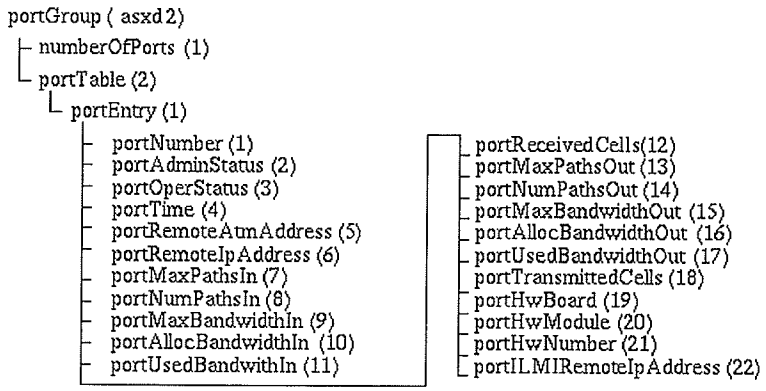
The managed objects in this group contains the information for the hardware and software versions of the switch. This group also includes the managed objects for the maximum number of paths and channels, as well as the ATM address of the switch. The managed objects in this groups are:



The Port Group

This group of managed objects contains the information on the number of ports that are available on the switch and a table of information for each of the ports [information at the port level].

For each port, the major information is the status of the port, the number of maximum and current paths in and out of the ports, and the maximum and in-use in and out bandwidths for that port. The table also includes the number of ATM cells that are received and transmitted from the port. The managed objects in the Port table are:



Path Group

This group contains three tables of managed objects for information on the virtual path for the switch [information at path level].

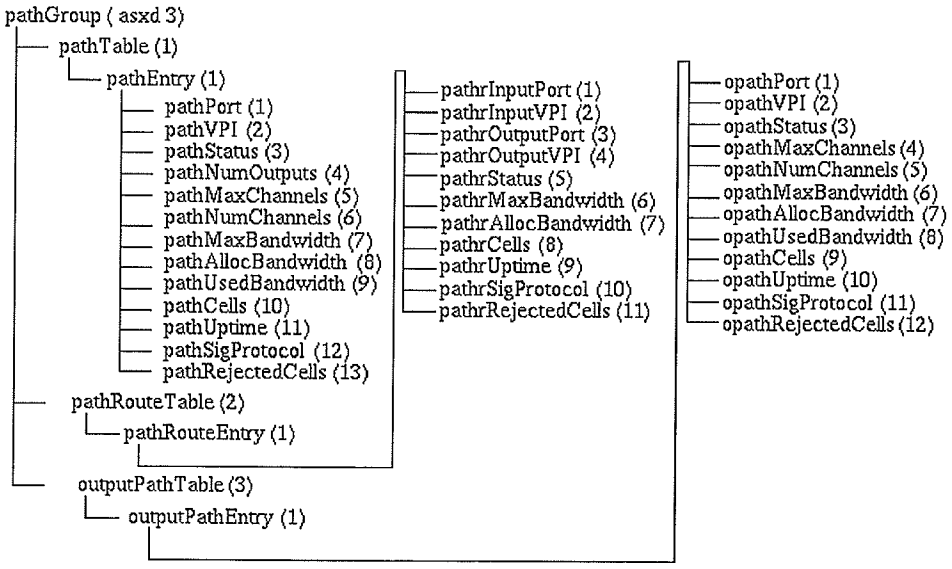
The "Path" table contains information on the virtual paths passing through the switch. Each path is indexed by the port that contains the path and the path identifier. The table includes the path operational and administrative status, the port that the path is routed to, the maximum number and current number of channels that can be allocated to this path, the maximum and current bandwidths (number of cell transferred per second), and the number of cells transferred and rejected due to error or traffic congestion on the path.

The "Path Route" table is the path routing table. Each path route is indexed by the input port, the input virtual path identifier, the output port and the output virtual path identifier. The table contains the main information on the number of cells transferred by this route, and the number of rejected cells by this route due to traffic violation, route status, and the maximum and current bandwidths of this route.

The "Output Path" table contains information about the virtual paths originating at the switch. Each output path is indexed by the port number and the virtual path identifier. The

main information contained in this table is the maximum and current numbers of channels within this output path, the number of cells transferred and rejected due to traffic violation by this path, maximum and current bandwidth of the output path.

The managed objects for the three tables of the Path Groups are:



Channel Group

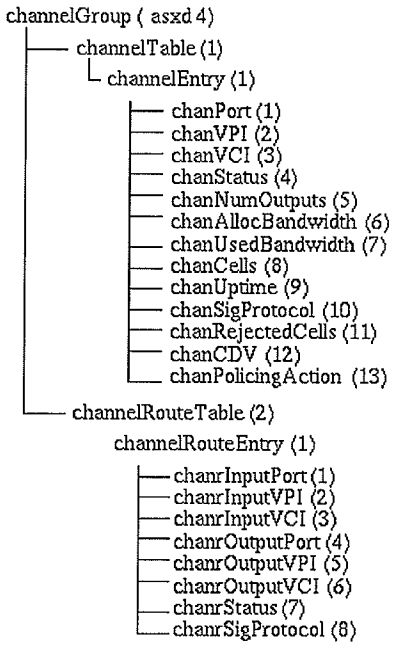
This group is concerned with the data traffic at the channel passing through the switch. The group consists of two tables, the "Channel" and the "Channel Route".

The "Channel" table contains information about each of the channels going through the switch. Each channel is indexed by the port number and virtual path identifier of the path that contains the channel and the virtual channel identifier of the channel itself. For each channel, the table has managed objects for the status of the channel, the number of output ports that the channel is routed to, the maximum and current bandwidths of the channel, and the numbers of cells transferred and rejected due to traffic violation by the channel.

The "Channel Route" table contains the information of the routing method for the channel through the switch. Each route is indexed by the input port, input VPI, input VCI, output port, output VPI and output VCI. The table also contains the status of the route for each route.

For more information about the above tables of managed objects regarding the FORE-ATM switch, please refer to the vendor literature [FORE Systems, fore_switch.mib, v 1.320, 1993-1994]

The managed objects for the Channel groups are:



6.4 INTRODUCTION TO THE MIB SUB-TREE "MIB-2"

The sub-tree "mib-2" in the MIB tree contains the information of the TIP/IP network.

Since the simulation of the network in this thesis also includes the TCP/IP sub-network interface with the ATM switch, this section will briefly introduce the groups that are contained in this sub-tree. The managed objects for these groups can be found in literature [Stalling W. 93].

System Group

This group contains the general information about the devices in the system.

Interface Group

This group contains information about the physical interface of the device. Its managed objects include the configuration, and the statistics on the data traffic and events occurring at each interface of the device.

IP, ICMP, TCP, UDP,EGP Groups

These groups contain information that is required and necessary for each protocol. This information includes the status, the traffic in and out, and the errors encountered during operation.

Address Translation Group

This group contains a table with three managed objects for each of the interfaces of the devices.The managed objects are representations of the address conversions of the device from the network address to the physical address for each interface.

Simple Network Management Protocol (SNMP) Group

This group of managed object contains information on the SNMP messages accessing the managed entities. It includes the number of each kind of SNMP messages requested and the number of each response type. It also includes the number of errors in SNMP messages due to a wrong version of SNMP, for example, the managed object requested does not exist, etc.

6.5 SNMP MESSAGE FORMAT

In SNMP, the request for information is carried by the SNMP messages. The SNMP messages carry information from the manager entity to the client entity to obtain the requested data, and from the client entity to the manger entity to response to the request. The SNMP messages are different for different kind of requests. The format for the SNMP messages are shown in **Figure** 6.23 [Stalling W. 93].

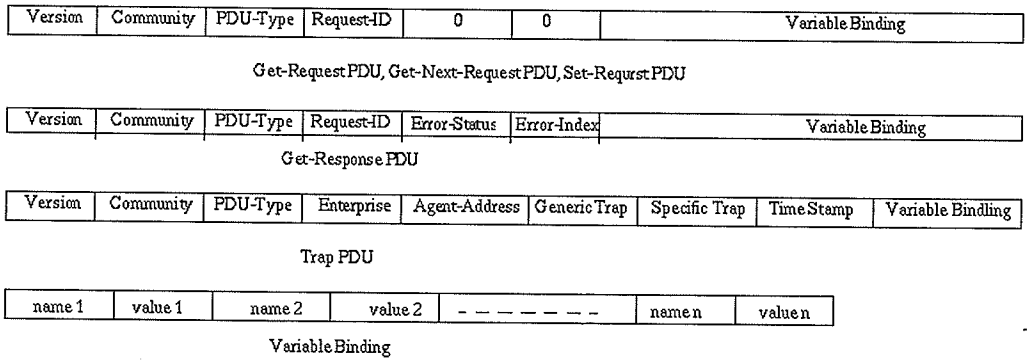


Figure 6.23 SNMP message formats.

Each message contains a version of the SNMP, a community name acting as a password to authenticate the SNMP message.

Request ID: The unique ID for each request to distinguish among outstanding requests.

Error-Status: This is in the Get-Response PDU to indicate an error is encountered in processing the request. The values of this field are: NoError (0), tooBig (1), noSuchName (2), badValue (3), readOnly (4), gerError (5).

Error-Index: This field has a non-zero value when the error status is non-zero to indicate additional information on what variable in the variable binding list is causing the exception.

Variable-Binding: The list of variable names and their corresponding values [the variable is an instance of the managed object]. For the Get-Request and Get-Next-Request, these values are set to NULL.

Enterprise: The type of object generating the trap.

Agent-Address: The address of the object generating the trap.

Generic-Trap: This indicates the trap type. Its value can be cold-start(0), warm-start (1), link-down (2), link-up (3), authentication-failure (4), egpNeighbor-loss (5), enterprise-specific (6)

Specific-Trap: The specific trap code.

Time-Stamp: This indicates the time that has elapsed between the last reinitialization of the managed entity and the generation of the trap.

[For more information on the processing of SNMP PDU, please refer to the literature Stalling W. 93]

6.6 CONCLUSION

This chapter introduced the MIB tree for ATM switches from a number of different sources, including ATM-Forum, RFC 1695, and FORE. The MIB trees for ATM of ATM-Forum and RFC 1695 were briefly discussed. The FORE-system MIB tree was discussed in some detail. Not all of the MIB tree for the FORE-system were presented, only those groups and managed objects that were considered in the implementation of the developed software. The sub-tree "mib-2" was also briefly mentioned in this chapter. This chapter also introduced the structures of SNMP messages.

In the next chapter, the design of the traffic monitor software of a network using SNMP will be discussed. The design of this software is based on the OMT methodology discussed in Chapter Two.

CHAPTER 7
OMT MODELS FOR NETWORK MANAGEMENT

7.1 INTRODUCTION TO NETWORK

As mentioned earlier, the network considered for the development of a network traffic monitor is located at **T**elecommunication **R**esearch **Lab**oratories (**TRLab**s) in Winnipeg. Only a portion of the network was considered at the time of the software implementation.

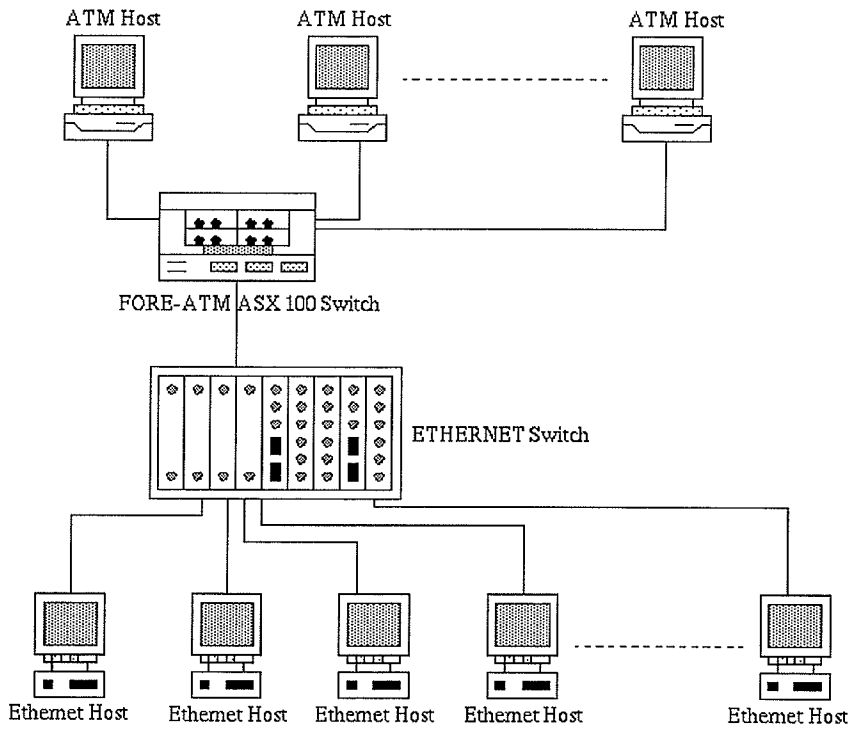


Figure 7.1 The TRLabs Sub Network.

The network at **TRLabs** is an ATM-based LAN interfaced with an ETHERNET switch. The portion of the network considered consists of a FORE-ATM ASX 100 switch, and ATM hosts (SUN workstations). This part of the network is interfaced to an ETHERNET switch and connected to many DEC hosts. The network is shown in **Figure** 7.1.

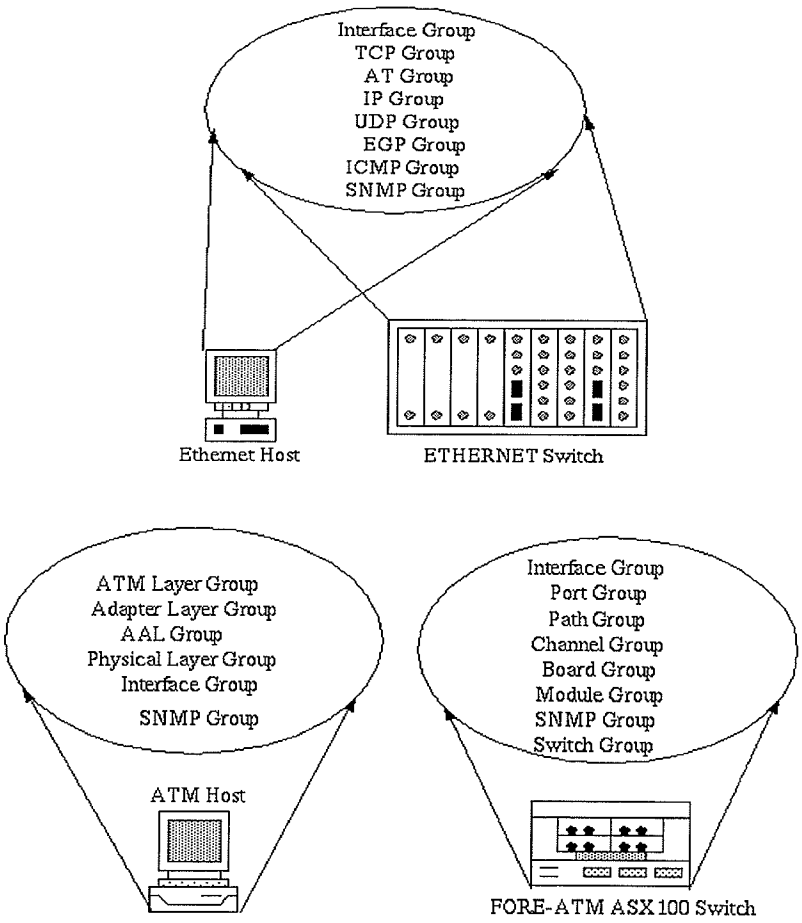


Figure 7.2 Managed Objects For each device.

In the network of **Figure** 7.1, one of the ATM hosts is selected as the system manager. The software application is run by the ATM host and the information of the system is displayed on this host. The manager software also monitors the traffic on the manager host.

As mentioned in the previous chapter, each device (host/switch) of the network considers only the set of managed objects that are appropriate for that type of device. In the future, more managed objects can be manually added to the software as desired. **Figure** 7.2 presents the number of managed object groups considered for each type of devices.

Each group is subdivided into two parts, the table and the simple managed objects. Each group of managed objects may contain either tables of managed objects or simple managed objects or both. The IP group, for example, has both table and simple managed objects, whereas the AAL group contains only tables and the SNMP group contains only simple managed objects. In the following sections of this chapter, the system described above is modeled using the OMT approach. Each model of the OMT, the Object Model, the Dynamic Model and the Functional Model, is presented and discussed.

7.2 OMT OBJECT MODEL

Figure 7.1 depicts the network, consisting of hosts and switches, which was considered for developing the traffic monitoring software. The network management system consists of the manager entity(ies) and the client entities that are managed by the manager entity. Thus, the two major object classes in the object model of the traffic monitor are the "Manager Agent" representing the manager entity, and the "Client Agent", representing the client entity. Each device in the system is then generalized to have its own characteristics.

The network management system manages the network by observing device information and setting the characteristic values of the devices in the system through the managed objects. The managed object refers to the logical and physical resources of interest for the modeling of a network management system. It is defined as a class with attributes reflecting these resources.

In the design of the object model, the managed objects are arranged in two groups, which are the "Table of managed objects" and the "Group of managed objects". The "Table of

managed objects" is the data structure for a table of managed objects in the MIB tree. The "Group of managed objects" consists of the simple managed objects which are under the same group in the MIB tree. The operations of setting and retrieving information from the network devices are carried out by the class of utility or group operations. The object model of the designed system is shown in **Figure** 7.3.

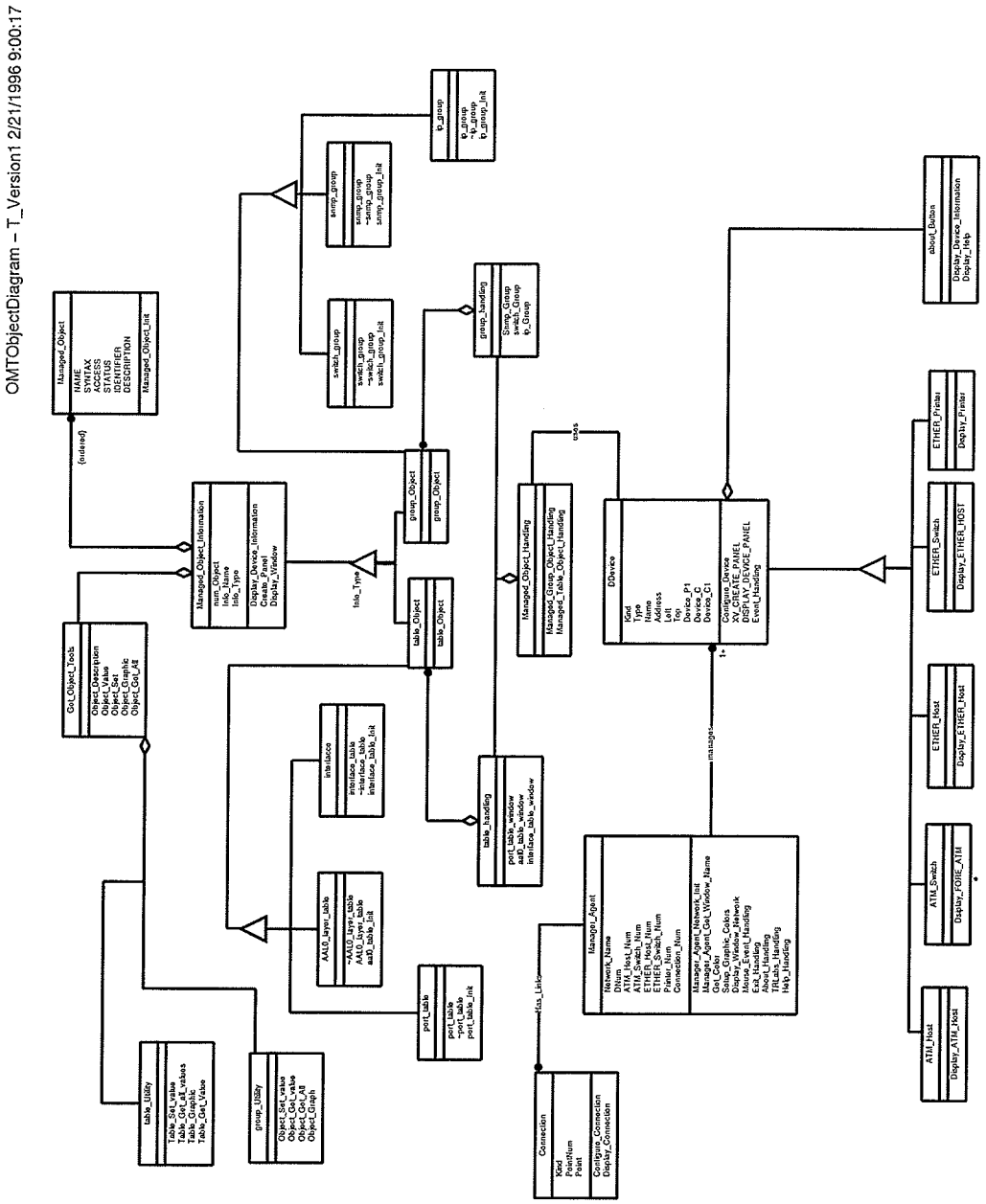


Figure 7.3 Object Model For Network Management System

In the following section, each of the classes are discussed along with its structures and its functions in the object model.

CLASS MANAGER_AGENT

This class models the network system manager. The class "Manager_Agent" has attributes that describe the configuration of the network, such as the number of devices being managed, the number of each of the types of devices, such as ATM hosts and ATM switches. In this simulation, the number of devices and number of device types are limited. The types of devices are limited to ATM host, ATM switch, Ethernet switch, Ethernet host and printers. Connections between the devices are physically present in the network, but are not physically managed by the manager. The manager agent only observes the traffic in and out of the interfaces, ports of each of the devices. The operations of this class include initialization of the network configuration, graphic setup for the main window, and event handling on the main window, such as the handling of menus, the handling of mouse clicks on the devices.

This class, "Manager_Agent", associates with two other classes, class "DDevice" and class "Connection". Thus, the implementation of class "Manager_Agent" has classes "DDevice" and "Connection" as its attributes. Class "Manager_Agent" sends messages to class "DDevice" upon mouse events. The association between the class "Manager_Agent" and class "DDevice" is one-to-many, since the manager agent manages at least one device, which is the host that is chosen as a manager host.

CLASS DDEVICE

A device (host, switch or printer) is modelled as a base class "DDevice". This class contains the general information on the device, such as the name of the device (e.g., host with the name of "truffle"), the kind of device (e.g. host or switch), the type of device (e.g.,

ATM or Ethernet technology device) and the IP address of the device within the network (e.g., the FORE-ATM switch at TRLabs has the IP address of 192.168.1.185). This information is provided by the network configuration file. The attributes of this class also include the information on the location (x, y co-ordinates) of the device on the main window.

Each specific device is inherited from this class, for instance the ATM host is modelled as class "ATM_Host" which is a sub-class of the class "DDevice". In the object model, only five sub-classes are derived from the class "DDevice", and these five classes represent the five types of devices that are considered in the simulation. The object model can be extended for a network with more types of devices by creating them as inherited from the class "DDevice", and with more attributes and operations as necessary.

The class "DDevice" has operations including the initialization of the device configuration, the handling of the events passed from the class "Manager_Agent", and the creation of a sub-window for device information when the device is selected from the main window.

The class "DDevice" has class a "about_Button" as its component. The class "about_Button" is a class of utilities that is used to handle some of the events occurring on the class "DDevice", such as the selection of the menu on the sub-window of the device.

The class "DDevice" associates with class "Managed_Object_Handling". "Managed_Object_Handling" is modelled as a group of operations that are handling the event occurring in the class "DDevice", such as the selection of "Group-Information" or "Table-Information" menus.

CLASS MANAGED_OBJECT

The managed object is modelled as an object instance of the class Managed_Object. The class contains attributes that describe all the necessary information on a managed object. These attributes are as follows:

NAME: is the textual representation of the object and gives the meaning or role of the information that the managed object holds. For example, the managed object for the number of cells that are transmitted over a port of the ATM switch has the NAME "port-TransmittedCells"

SYNTAX: is the syntax of the managed object, that is the type of information the managed object holds. For example, the managed object "portTransmittedCells" has the syntax "Counter".

ACCESS: is the access level of the managed object. For example the managed object "portTransmittedCells" has an access level of "Read-Only".

STATUS: is the status of the managed object. For example, such as the managed object "portTransmittedCells" has a status of "Mandatory".

IDENTIFIER: is the identification (series of numbers) of the managed object within the MIB tree; it distinguishes one managed object from another. (the NAME distinguishes between managed objects, but the IDENTIFIER also represents the location of the managed object in the MIB tree). For example, the managed object "portTransmittedCells" has the IDENTIFIER of 1.3.6.1.4.1.326.2.2.2.1.2.2.1.18.

DESCRIPTION: is the textual description of the role of the managed object. It describes what kind of information and the meaning of the information value that the object holds. For example, the DESCRIPTION of "portTransmittedCells" is "*The number of cells*

transmitted over this port".

This class has only one operation, which is to initialize the values for attributes of the managed object. The class "Managed_Object" is a component of the class "Managed_Object_Information". The managed object initialization is activated by messages from sub-classes of "Managed_Object_Information". The messages for managed object initialization include all the values for the attributes of the class.

CLASS MANAGED_OBJECT_INFORMATION

This class defines a group of managed objects. Its attributes include the number of objects, and what type of information and the name of information they hold (these two attributes are used for the further refinement of classes derived from this class).

The operations for this class include creating the sub-window, panel to display the information for each group or table. This class has classes "Get_Object_Tool" and "Managed_Object" as its components.

The aggregation relationship between the classes "Managed_Object_Information" and "Managed_Class" is a one-to-many with order, since the class "Managed_Object_Information" is a group of many managed objects and these managed objects are arranged in order of their object identifier.

CLASSES TABLE_OBJECT AND GROUP_OBJECT

These two classes are sub-classes inherited from the class "Managed_Object_Information". These classes define the a group of objects as table or group. discriminating by the value of the attribute "info_Type" from the super class,

which is either "TABLE" or "GROUP".

Each of table or group of managed objects are defined as sub-classes of the class "table_Object" and "group_Object", respectively. The object model in **Figure** 7.3, presents some of the tables and groups that are implemented in the simulation.

Each of refined classes from the classes "table_Object" and "group_Object" have a constructor and a destructor. The constructors initialize the number of managed objects in the tables/groups and the name of these table/group. They also perform the table initialization of its contents. This initialization methods includes passing the values of object name, object syntax, object access, object status, object identifier and object description to the class "Managed_Object" to initialize each of the managed objects.

CLASS TABLE_HANDLING AND CLASS GROUP_HANDLING

The classes "table_handling" and "group_handling" define groups of operations and are components of the class "Managed_Object_Handling".

They have classes "table_Object" and "group_Object" as components and their aggregation relationships are one-to-many, since the class "table_handling" can have many tables and the class "group_handling" can have many groups. These classes contains operations to display the sub-window for each of table or group selected.

CLASS GET_OBJECT_TOOL

This class defines a group of operations that responds to requests from the menu of the sub-window for each table or group selected to retrieve and set values of particular managed object. It has two classes, "table_Utility" and "group_Utility" modelled as groups of

operations

The above paragraphs have described the classes and their relationships in the object model shown in **Figure** 7.3 The network management system is modelled with classes which present components of the system, which operates by passing messages/information from one class to others. For instances, the "Manager_Agent" passes messages to class "DDevice" to initialize the configuration of each of the device in the network. In the next section of the chapter, the dynamic models for each of the classes are presented with discussion.

7.3 OMT DYNAMIC MODELS

CLASS MANAGER_AGENT

The class "Manager_Agent" has the dynamic behaviour as illustrated in **Figure** 7.4.

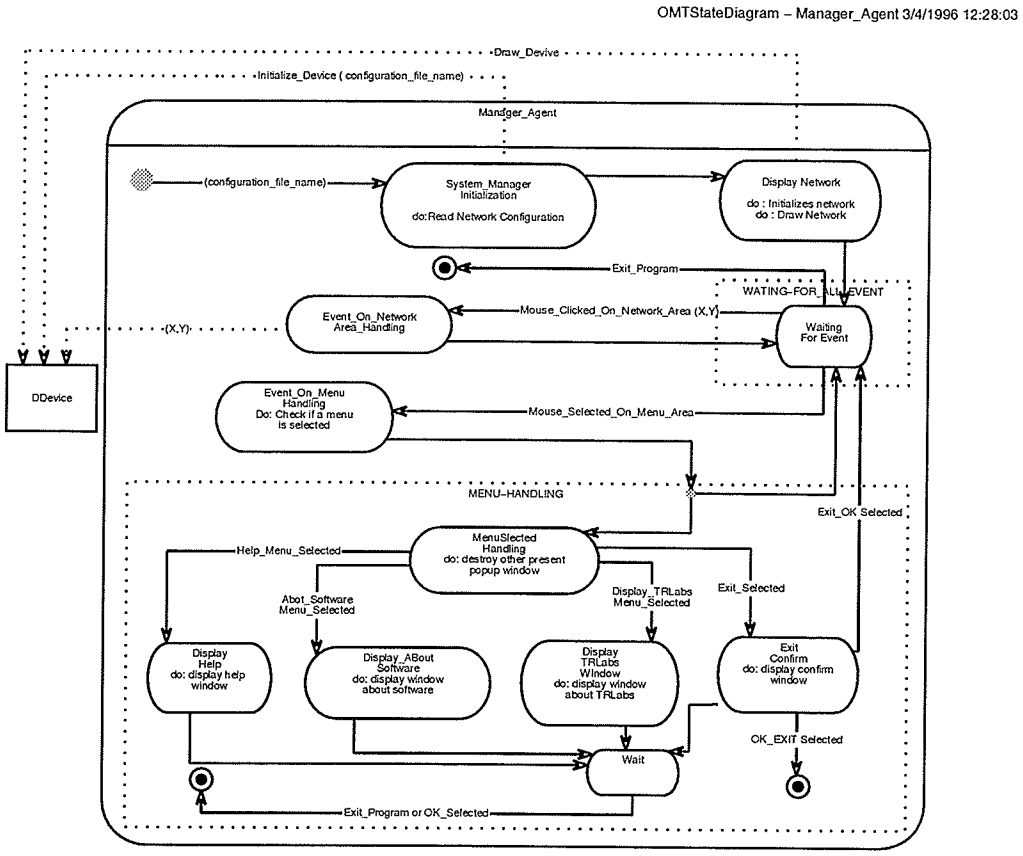


Figure 7.4 Dynamic Model for class Manager_Agent

The "Manager_Agent" starts with an initial state. The initial state makes a transition to the first state of the class, the "System_Manager_Initialization" with the name of the network configuration file. In this state, the class reads the name of the network and the configuration of the network, such as the number of devices and the number of each type of devices. Also in this state, the class sends an Initialize_Device event to the class "DDevice" with the name of the network configuration, so that the devices of the network can be configured. When all the activities in this state is accomplished, the class makes an automatic transition to next state, which is Display_Network. The class initializes the network and

draws the network on the main window by sending event Draw_Device to each of the devices to draw on the main window. After the network is drawn on the window, the class enters the state Waiting_For_Event [concurrent with the state WAITING_FOR_ALL_EVENT]. In this state, the class analyzes the event. If the mouse is clicked on the network area, the class sends the event to the class DDevice with the coordinates of the clicked point so that the class DDevice determines if the device is selected, and the class returns to Waiting_For_Event. If the mouse is clicked on the menu, the class determines what menu is selected and splits into two concurrent states, WAITING_FOR_ALL_EVENT and MENU_HANDLING. The concurrent state MENU_HANDLING occurs simultaneously with the concurrent state WAITING_FOR_ALL_EVENT. Within the state MENU_HANDLING, the event of menu selection is analysed and the class displays the appropriate window. When the popup window is closed, the MENU_HANDLING state exits. If the program exits, the class Manager_Agent is destroyed or exits as well.

CLASS DDEVICE

The dynamic model for class DDevice is shown in **Figure** 7.5. The class starts with the initial state. The configuration file name is passed from class Manager_Agent to class DDevice to read the device configuration in the next state DDevice_Init. When the device is initialized, the class changes to the state which displays the graphical presentation of the device on the network window. The class enters state Waiting_For_Event after displaying its graphical presentation on the network area. In this state, the class does not make any transition to another state until there is an event sent from the class Manager_Agent with

the co-ordinates of the point that is mouse-clicked on the net work area.

When the Manager_Agent sends an event to the class DDevice, the class DDevice determines if the device is selected. If the device is not selected, it returns to its Waiting_For_Event state. If the is device is selected, it displays the Device_Main_Window in state Display.

If one of the menus of the Device_Main_Window is selected, the class determines what menu is selected and sends an event to an appropriate class with corresponding information, as shown in the **Figure** 7.5. It then returns to state to Waiting_For_Event. The class DDevice is exited when the program ends.

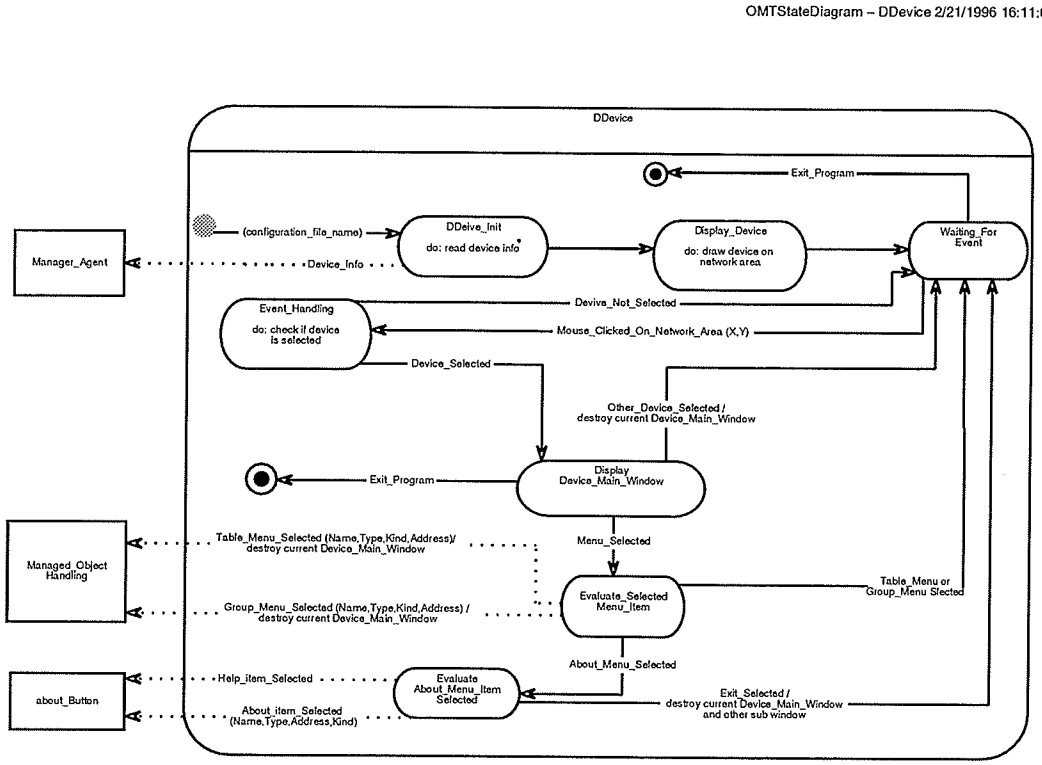


Figure 7.5 Dynamic Model for class DDevice

CLASS MANAGED_OBJECT

The dynamic model for the class Managed_Object is shown as in **Figure** 7.6 below. The dynamic model for this class is very simple. Each managed object is created when either the table or group of managed objects are created. Each managed object is started with an initial state and goes into state Managed_Object_Inititalization. The managed object is initialized in this state with the information of NAME, SYNTAX, ACCESS, STATUS, IDENTIFIER, DESCRIPTION passed from the sub-class of classes table_Object (such as classes port_table, AAL0_Layer_table and interface in the object model) or group_Object (such as classes switch_group, snmp_group, ip_group).

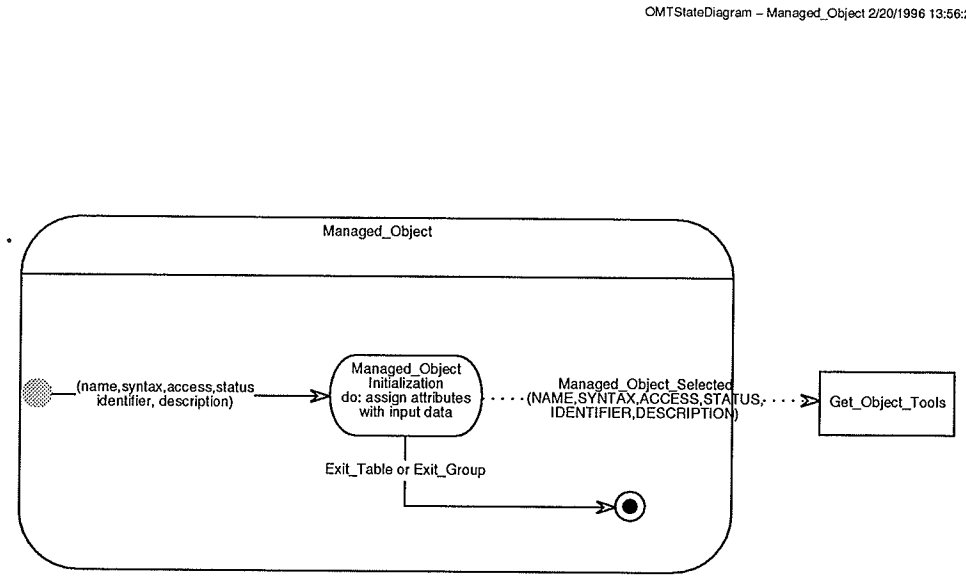


Figure 7.6 Dynamic Model for class Managed_Object

The managed object exits when the sub-class of table_Object or group_Object exits. The

class passes its information (NAME, SYNTAX, ACCESS, STATUS, IDENTIFIER, DESCRIPTION) to the class Get_Object_Tool when the managed object is selected in the table or group window.

CLASS GROUP_HANDLING AND TABLE_HANDLING

These two class have their dynamic models as shown in **Figure** 7.7 (a) and **Figure** 7.7 (b). The classes group_handling and table_handling are created when a group and table, respectively, are selected from the device window by the event passed by the class Managed_Object_Handling with the device information and the name of the table or group. When the class group_handling or table_handling is created, they pass an event Group_Initialization (or Table_Initialization) with parameters of name of the group (or table), and the information of the selected device to the class group_Object or group_Object. Then the classes exits.

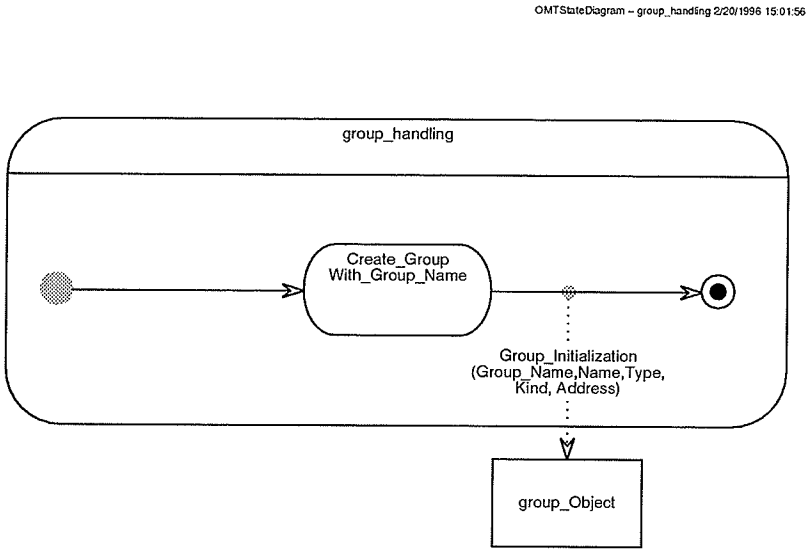


Figure 7.7 (a) Dynamic Model for class group_handling

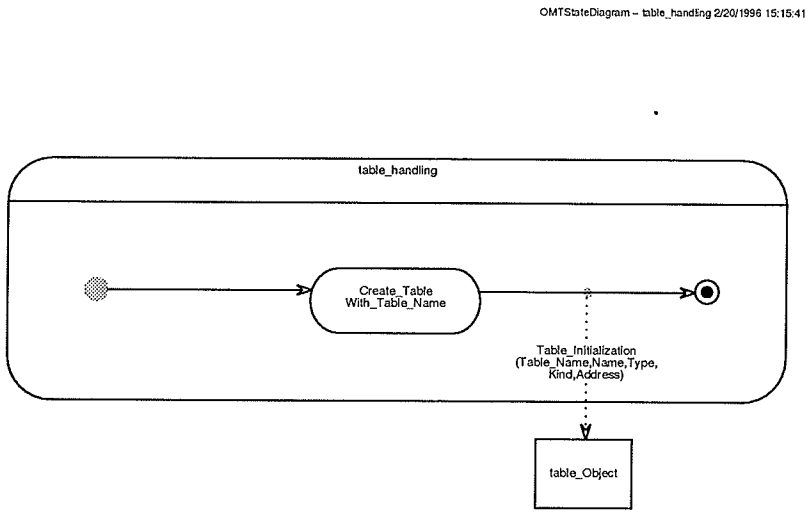


Figure 7.7 (b) Dynamic Model for class table_handling

CLASS TABLE_OBJECT AND GROUP_OBJECT

These two classes are very similar. They are inherited from the class Managed_Object_Information. The behaviour of the super-class and its sub-classes are the same. The dynamic models for the classes table_Object and group_Object are the same for the dynamic models for each individual table class (e.g. port_table) and group class (e.g switch_group), respectively.

The dynamic models for the table_Object and group_Object are shown in **Figure** 7.6 (a) and **Figure** 7.6 (b), respectively.

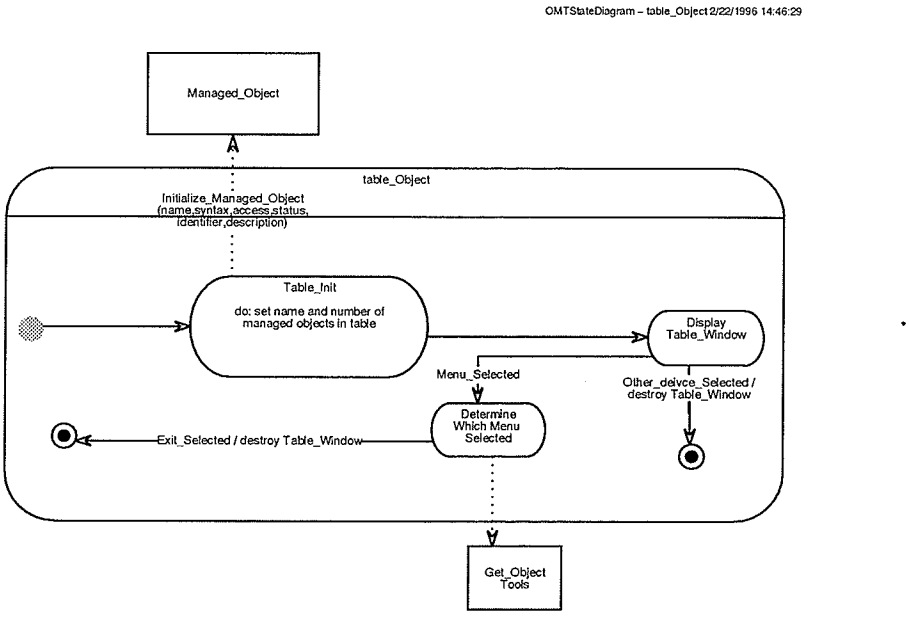


Figure 7.8 (a) Dynamic Model for class table_Object

Each of the group or table class starts with an initial state.When a group or table is selected from the device window and the group/table name, it enters the state Group_Init (or Table_Init) and sets the number of managed objects allocated in this group/table. It

then sends event Initialized_Managed_Object to the class Managed_Object with the necessary information for each managed object (name, syntax, access, identifier and description).

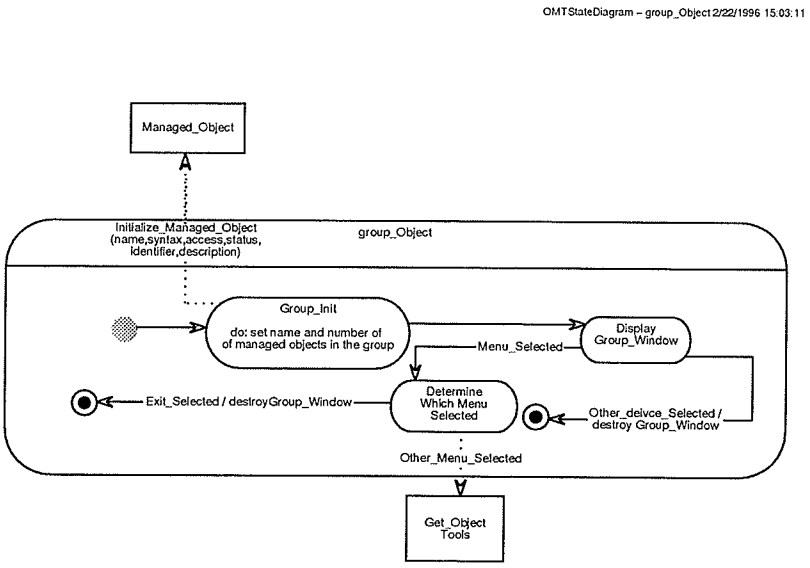


Figure 7.8 (b) Dynamic Model for class group_Object

When the initialization state is complete, the class enters the state Display_Group_Window (Display_Table_Window) to display the window and operations that can be performed for each group (table) and to wait for a further event. The group_Object (table_Object) class is exited when another device on the network window is selected. Selecting the exit menu on the window also causes the class to be exited.

CLASS MANAGED_OBJECT_HANDLING

This class is a set of operations created when a menu is selected on the device information

window. It displays a menu (Table Information or Group Information) from the device information window and sends the event to the corresponding class, table_handling or group_handling. The dynamic model for this class is illustrated in **Figure** 7.9.

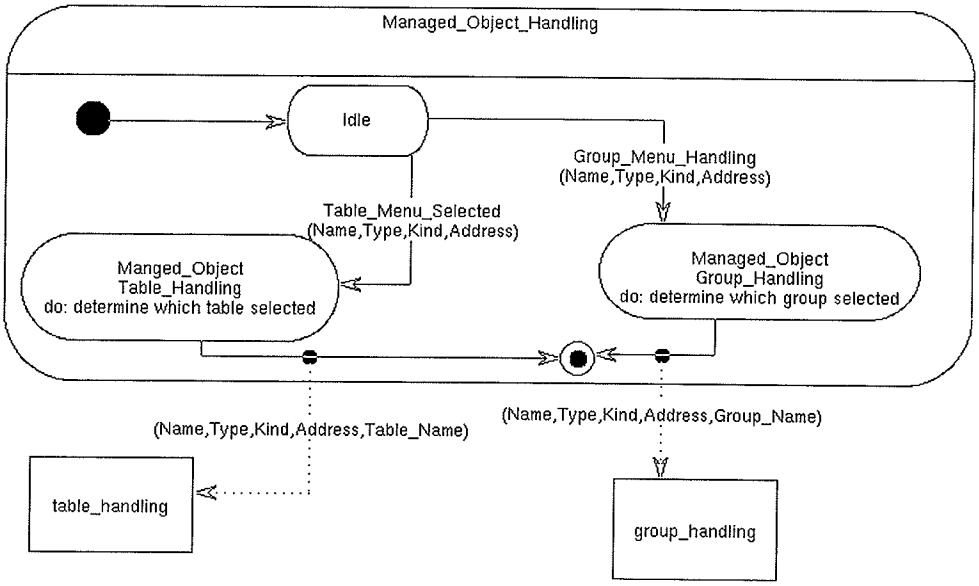


Figure 7.9 Dynamic Model for class Managed_Object_Handling

CLASS GET_OBJECT_TOOLS

This class is also group a of operations. It starts with an initial state and enters the state Idle. When there is an event sent from the class table_Object or group_Object, it evaluates the event and performs the appropriate actions shown in the dynamic model. The class determines if the event is from a table or group of managed objects and also classifies what kind of information has been requested, such as managed object description, managed object single value, whole table or group of managed objects, or graph the managed object value. When the event is a managed object description request, the class sends the

event Get_Managed_Objects to either class table_Utility or class group_Utility depending on whether the event has been sent from table_Object or group_Object, respectively. The dynamic model for this class is illustrated in **Figure** 7.10

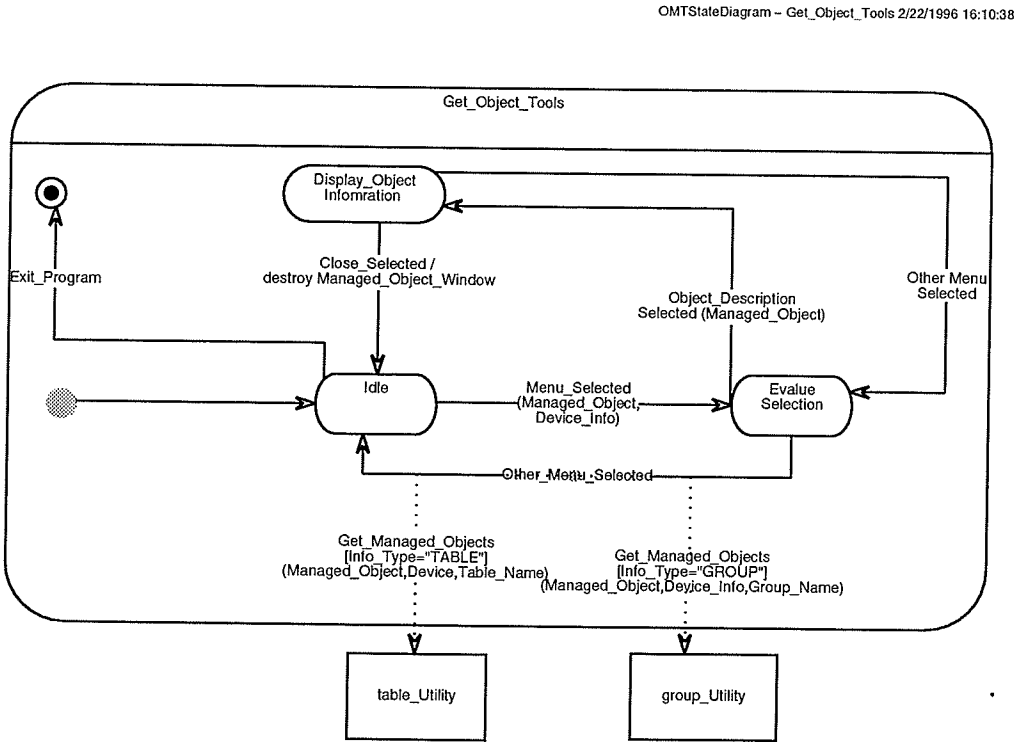


Figure 7.10 Dynamic Model for class Get_Object_Tools

CLASS GROUP_UTILITY AND TABLE_UTILITY

These two classes are groups of operations. The dynamic models for these two classes, group_Utility and table_Utility, are shown in **Figure** 7.11 (a) and **Figure** 7.11 (b), respectively.

Each class starts in an initial state and makes a transition to state Idle. When an event from class Get_Object_Tools is sent to one of these classes, the next state of the class depends

on the information from this event. The dynamic behaviour of these classes is illustrated in the **Figure** 7.11(a) and **Figure** 7.11 (b).

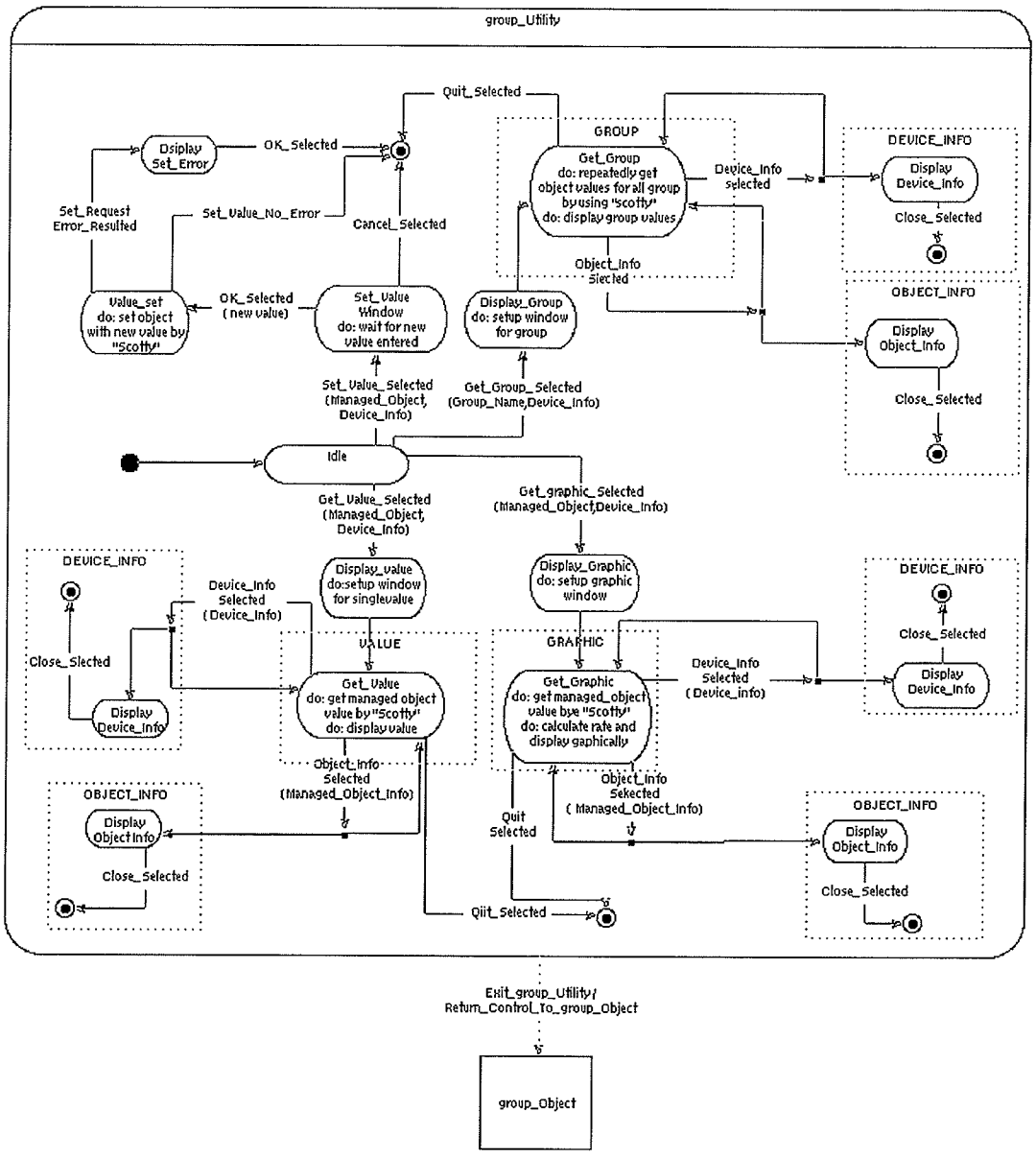


Figure 7.11 (a) Dynamic Model for class group_Utility

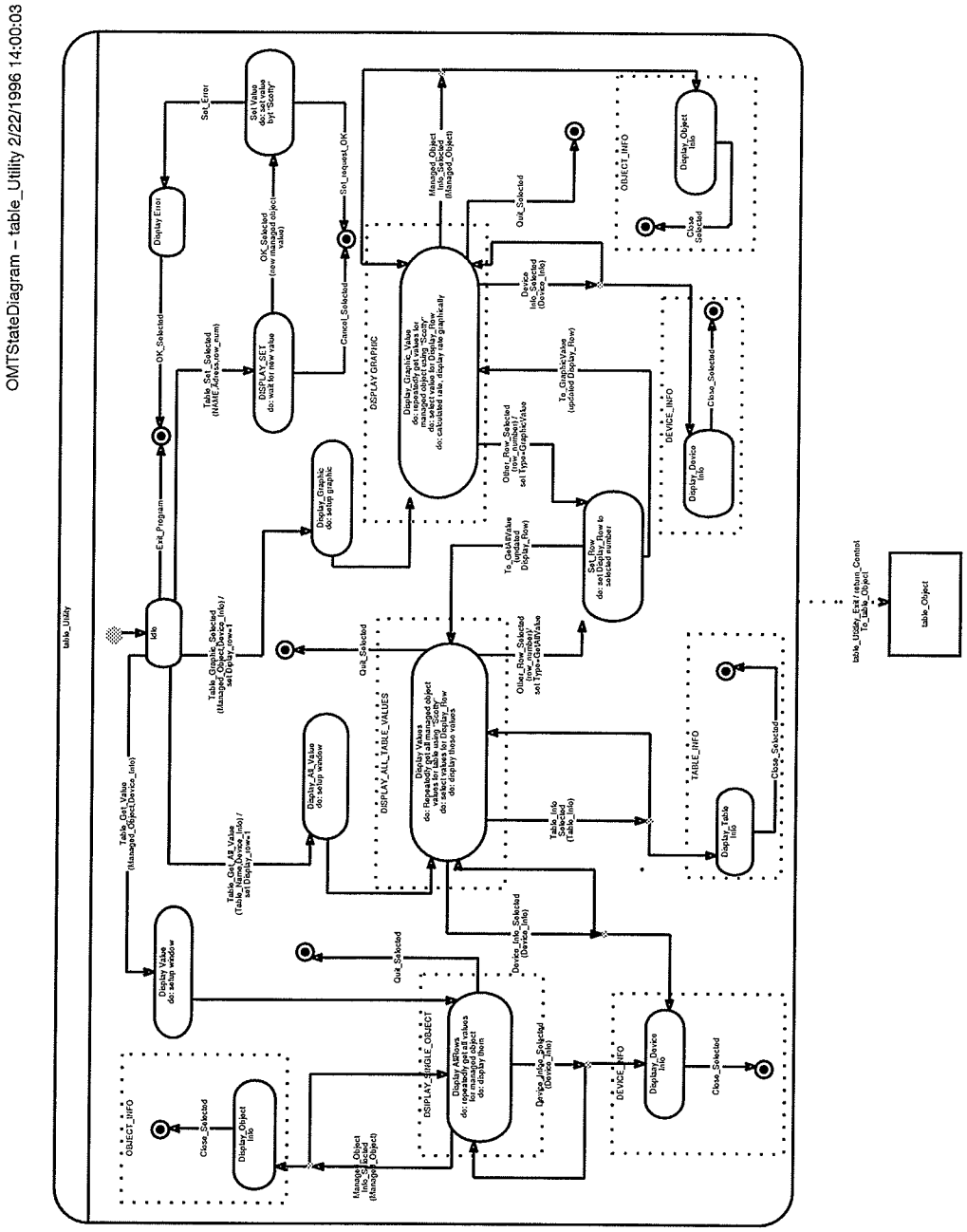


Figure 7.11 (b) Dynamic Model for class table_Utility

CLASS ABOUT_BUTTON

This class serves as a pop-up window for the device information window. The dynamic

behaviour of this class is as in **Figure** 7.12.

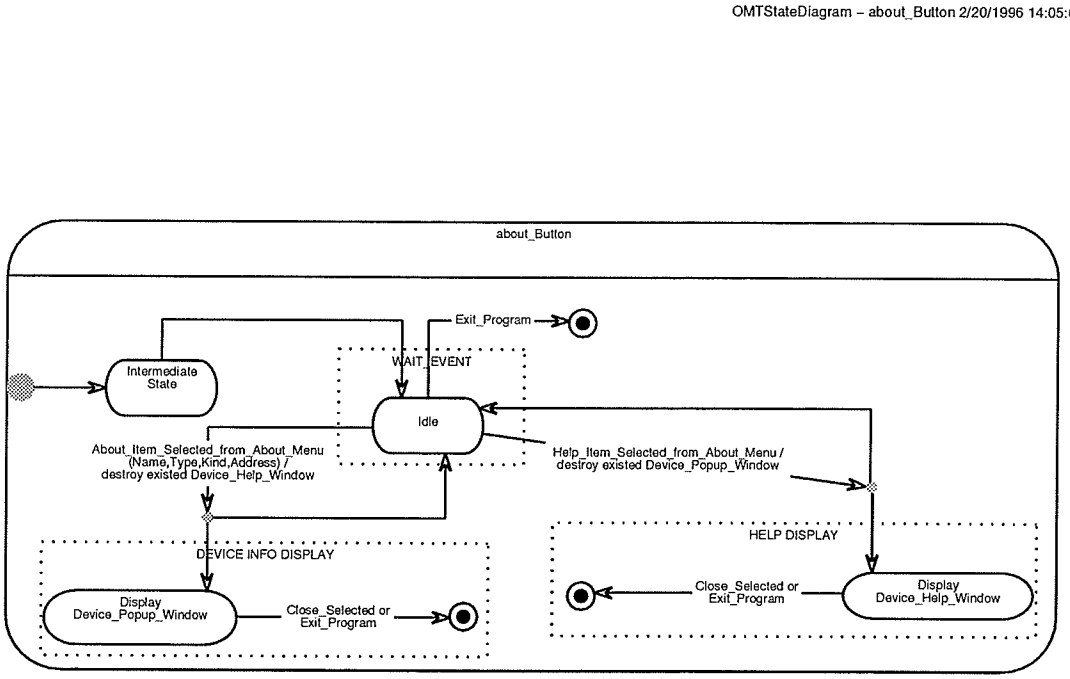


Figure 7.12. Dynamic Model for the class about_Button

7.4 OMT FUNCTIONAL MODELS

As mentioned in an earlier chapter, all functional models for the system have not necessarily been presented. Those function(s)/procedure(s) that are considerably simple have been omitted with the functional models.

Most of the function(s)/method(s)/procedure(s) of the classes defined in the object model are fairly simple and straightforward. The following figures shows some of the functional

models for the system.

Figure 7.13 displays the functional model for the operation of network initialization from the class Manager_Agent.

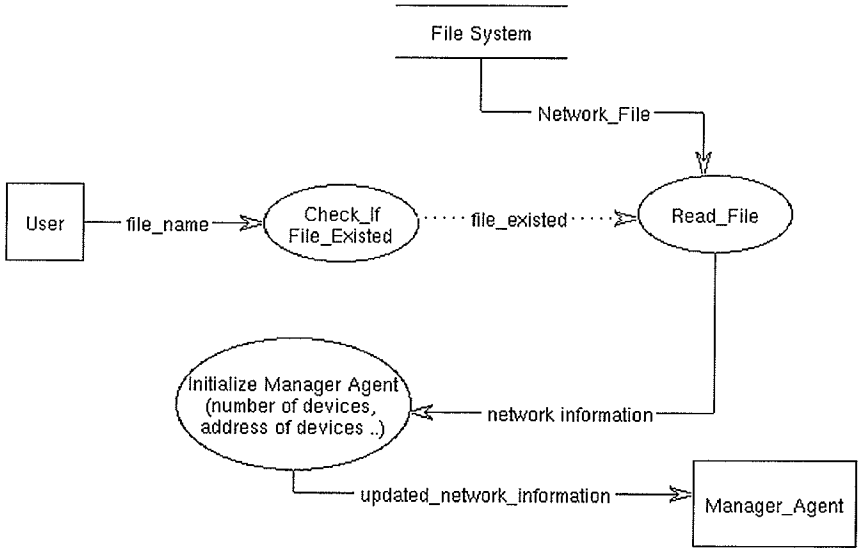


Figure 7.13 Network Initialization

The function checks if the file for the network configuration exists or not. If the file exists, the program reads the file and initializes the characteristics of the network for the network manager, such as the number of devices in the network, what type of devices, etc.

Each of the device is initialized as shown in **Figure** 7.14.

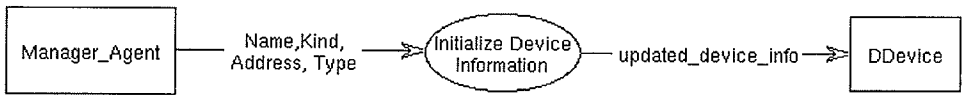


Figure 7.14 Device Initialization

The Manager_Agent passes the name, the kind, the IP address and the type of each device to the class DDevice to initialize its characteristics.

Other functional models are shown in **Figure** 7.14. These functional models presents some of the operations of the system. Other operations are omitted due to their simplicity.

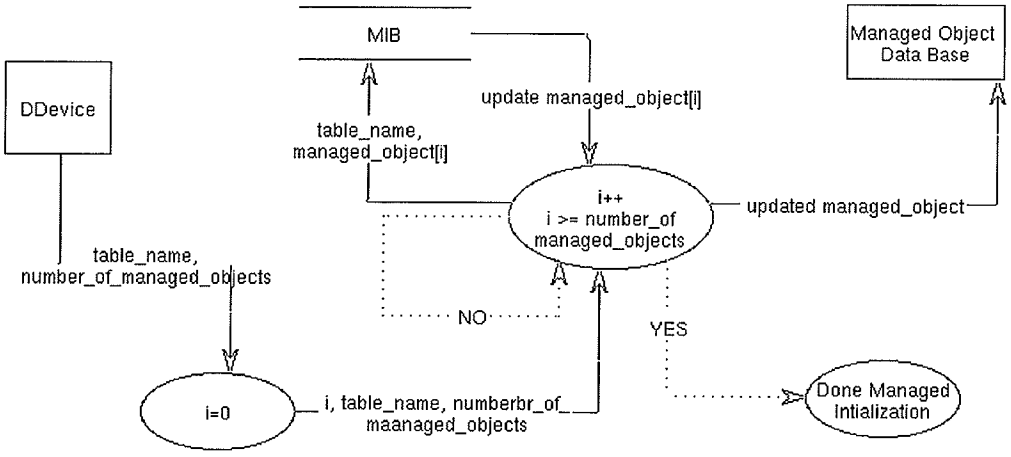


Figure 7.15 [a] Table & Group of Managed Object Initialization

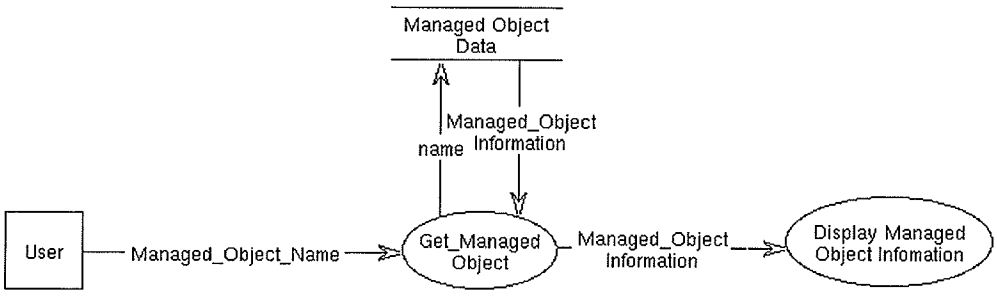


Figure 7.5 [b] Display Managed Object Information.

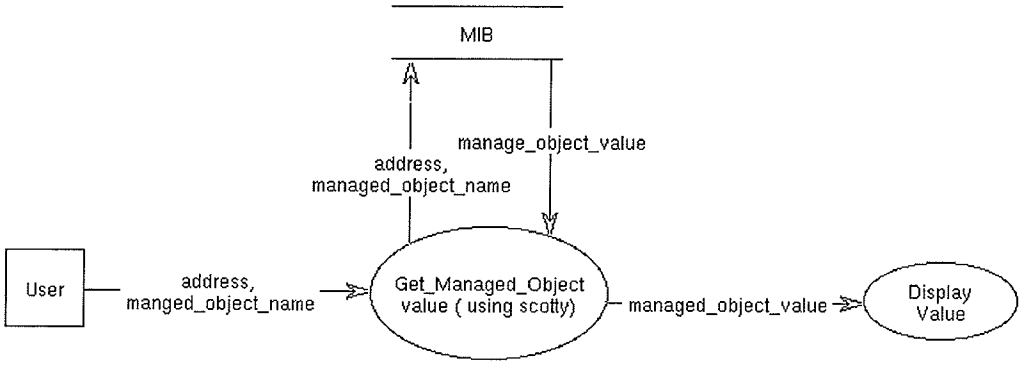


Figure 7.15 [c] Get Managed Object Value

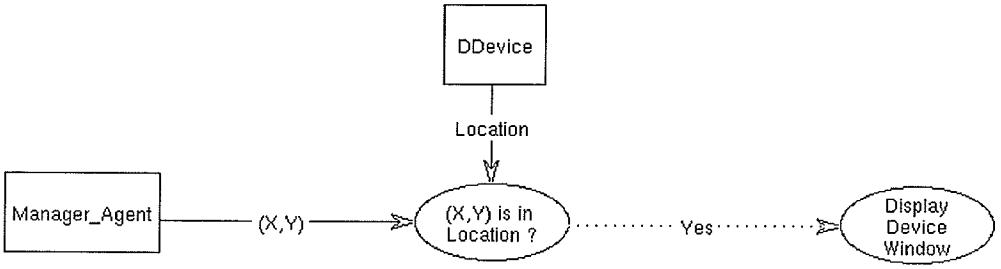


Figure 7.15 [d] Device Handling Of Mouse Events.

7.5 IMPLEMENTATION OF THE NETWORK TRAFFIC MONITOR

OMT is a concept for designing and analysing a system considering it as a group of concrete objects that pass messages among themselves to perform the system operations. It is not necessary that the implementation language system being designed be an object oriented language.

In the present instance, the language that has been chosen for implementation of the network traffic monitor is a combination of C++ and Tcl-Tk (Tool Command Language -

Tool Kit).

The implementation of the system is not pure object oriented. Most of the classes defined in the OMT object model of the system are defined as classes in the implementation. Some of the classes, such as Managed_Object_Handling, Get_Object_Tools, table_Utility, and group_Utility, which are groups of operations, are implemented as immediate functions for passing the messages between classes. The code for the designed system can be found in Appendix-B. The classes Get_Object_Tools, table_Utility, group_Utility are implemented using Tcl-Tk which performs the graphical display and retrieves the managed object values. The Tcl function "scotty", a Tcl intepreter with extensions to obtain various status information about the network, is used extensively in the implementation.

7.6 CONCLUSION

This chapter introduced the network at **TRL**abs that was considered for developing a management system. The network was limited to consist of a small number of ATM hosts and Ethernet hosts, as well as an ATM switch and an Ethernet switch. The network can be expanded to consist of other devices such as bridges and routers. The network manager system was analysed and developed with models according to the OMT approach for software development. The object model of the system consists only classes for the devices presented in the network. If the network were expanded through the addition of new devices, new class representations for these new devices could be added to the object model. The dynamic model for each class was also presented to describe the behaviour of the class within the system. Only a number of functional models were presented, since the

other functions of the classes are either simple or straightforward.

In the next chapter, the implementation of the designed system will be presented. The developed software is discussed in the following chapter.

CHAPTER 8
NETWORK TRAFFIC MONITOR

The network traffic monitor is developed incorporating a graphical user interface. The configuration of the network is provided in a file so that the program will read the file and initialize the network.The following **Figure** 8.1 shows the organization of the software.

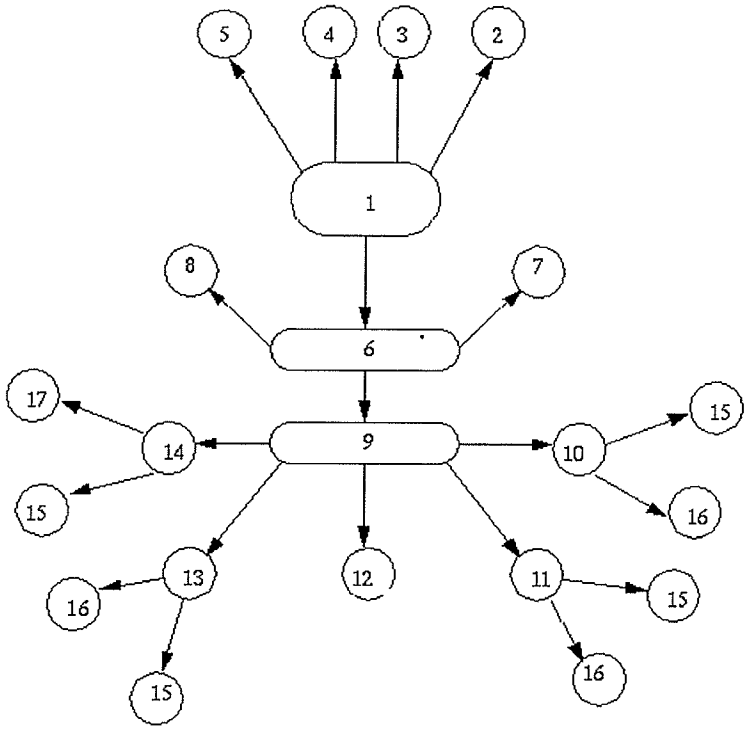


Figure 8.1 Behaviour of Traffic Monitor

In the above **Figure** 8.1, each element represents a window which will be described later in this chapter. Each window is displayed by the event occurring in its parent window and described according to its transition from the parent. The details of these event will also be

described later.

8.1 MAIN WINDOW - INDICATED BY ELEMENT 1

The main window of the software displays the layout of the network that it manages. The main window for the network at **TRL**abs is shown in Figure 8.2.

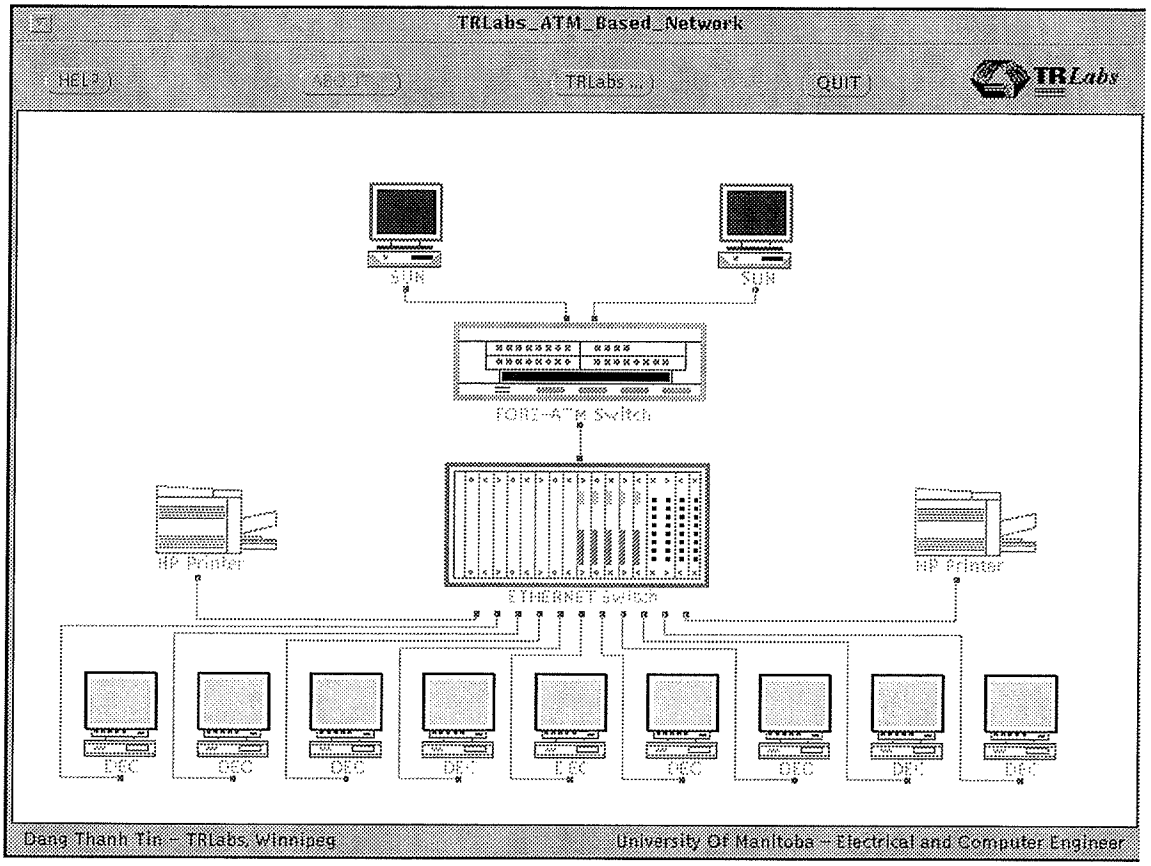


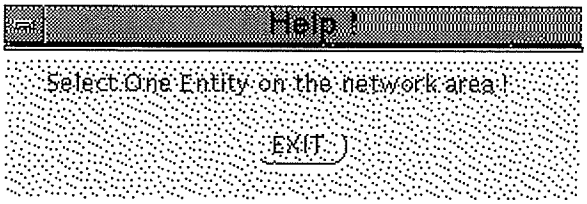
Figure 8.2 Main Window

The menu of the main window consists of four functions, the "QUIT" button which displays the notification to confirm before exiting the program, the "TRLabs" button which

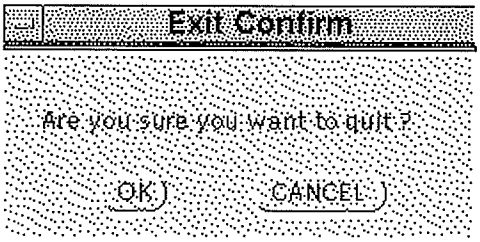
displays the icon of the sponsor of the project, the "ABOUT" button which displays the author of the system and the "HELP" button which display the help commands available for the user.

The pop-up windows for the above functions are represented as circle with numbers (2), (3), (4) and (5) as in **Figure** 8.1 and illustrated in **Figure** 8.3. These pop-up windows are displayed when the button is selected from the main window and closed when the appropriate action occurs.

Circle #2



Circle #3



Circle #4

Figure 8.3 Pop-up windows from main window

Circle #5

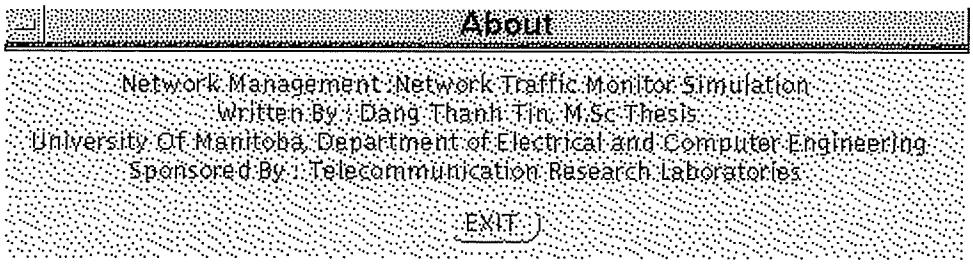


Figure 8.3 Pop-up windows from main window (cont.)

When a device in the network area is selected from the main window, a sub-window, as shown in **Figure** 8.4, describes the device. The sub-window is presented as element #6 in **Figure** 8.1.

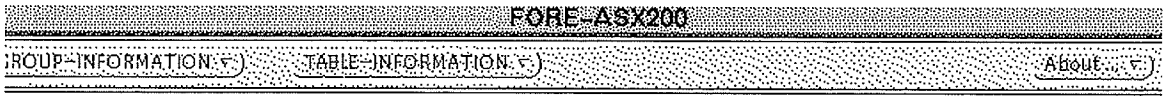


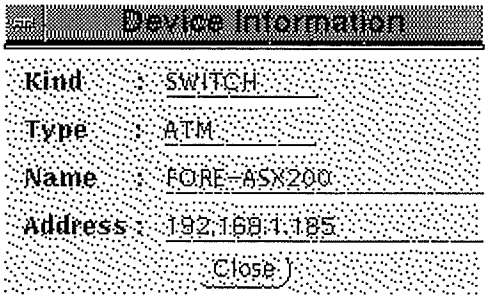
Figure 8.4 Device Window

8.2 DEVICE WINDOW - INDICATED BY ELEMENT 6

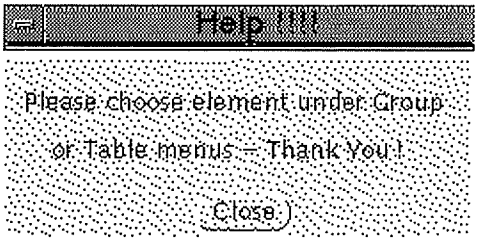
The device window contains 3 main functions:

"About" Button which includes help functions from the device window, information on the selected device and exit from the window. The "Help" and "About" pop-up windows represented as circle #7 and #8 in **Figure** 8.1, are shown in **Figures** 8.5.

Circle #7



Circle #8



Figures 8.5 Pop-up windows from Device window.

When an item is selected from either the "TABLE-INFORMATION" or the "GROUP-INFORMATION" menu, the device window is extended with more information on the selected table or group of managed objects and a new window represented as element #9 in **Figure** 8.1 is shown and illustrated in **Figure** 8.6.

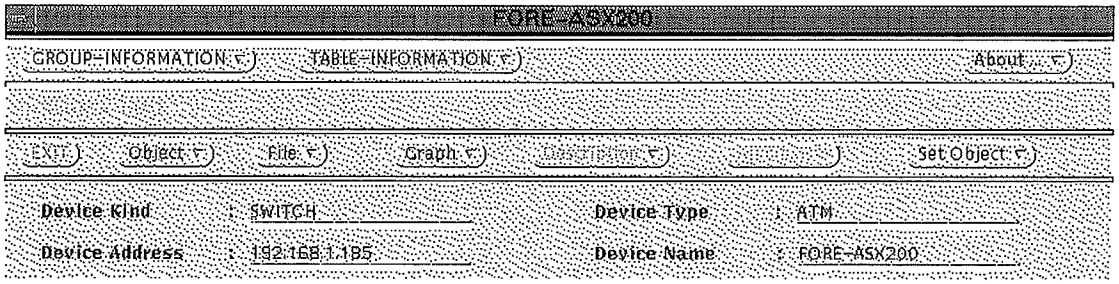
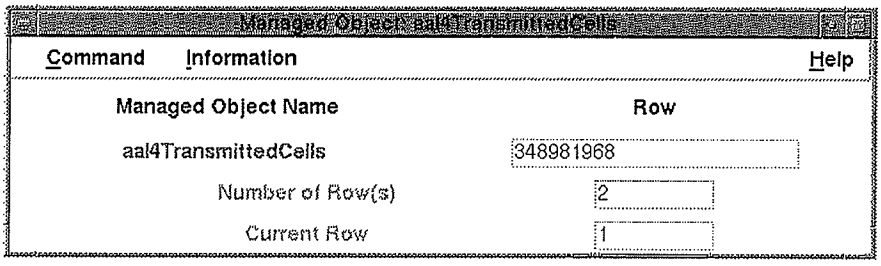


Figure 8.6 Table/Group of Managed Object Window

8.3 TABLE/GROUP WINDOWS - INDICATED BY ELEMENT 9

This window contains many functions to retrieve managed object information, including: retrieving a single object value (circle #10), retrieving a whole table or group (circle #14),retrieving a single managed object value and displaying it graphically (circle #11),retrieving description of the managed object (circle #12), setting a value for managed object (circle #13). The pop-up windows are created appropriately when the item is selected from the menu. These pop-up windows are as in the following **Figure** 8.7.

Circle #10



Circle #11

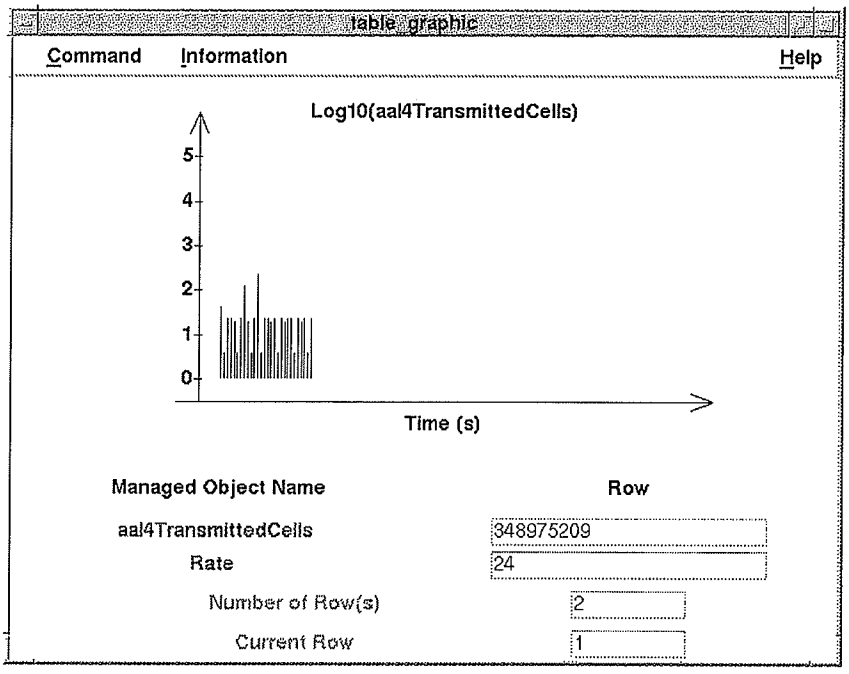
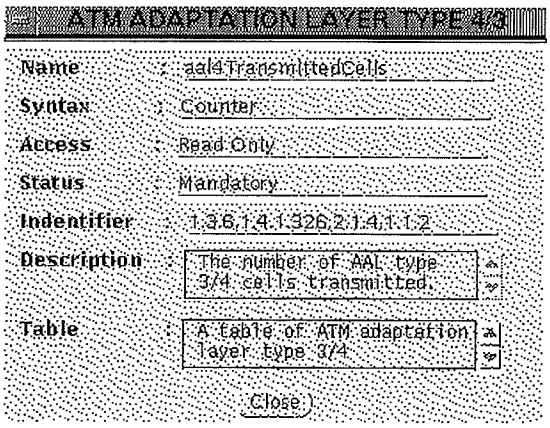
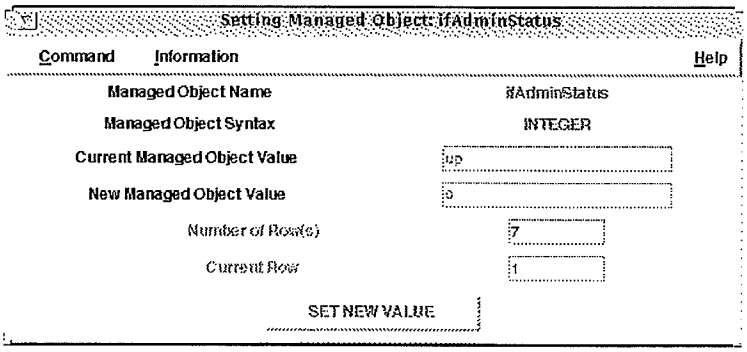


Figure 8.7 Pop-up windows from Table/Group of Managed Object window.

Circle #12



Circle #13



Circle #14

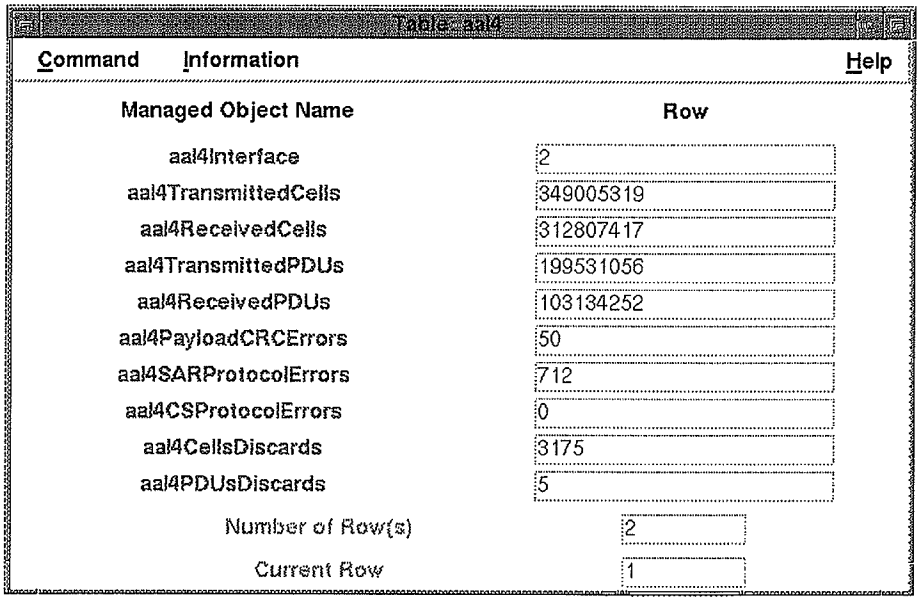
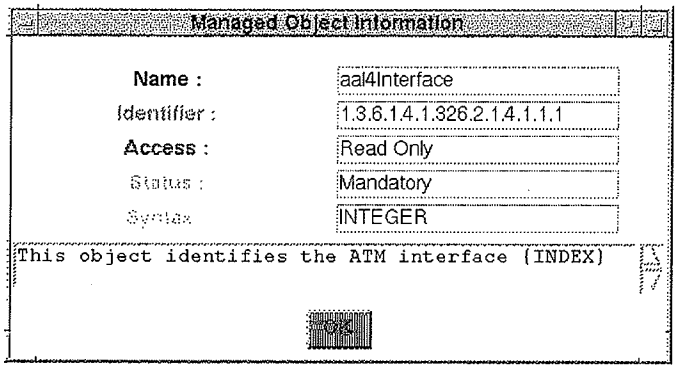


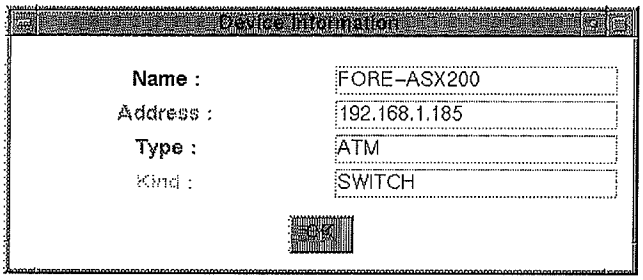
Figure 8.7 Pop-up windows from Table/Group of Managed Object window (cont.).

In the above pop-up windows, information about the individual managed object, the information for group or table of managed objects and information of the current selected device can be retrieved. These type of information are shown in **Figure** 8.1 as elements 15, 16 and 17, respectively and are illustrated in **Figure** 8.8

Circle #16



Circle #15



Circle #17

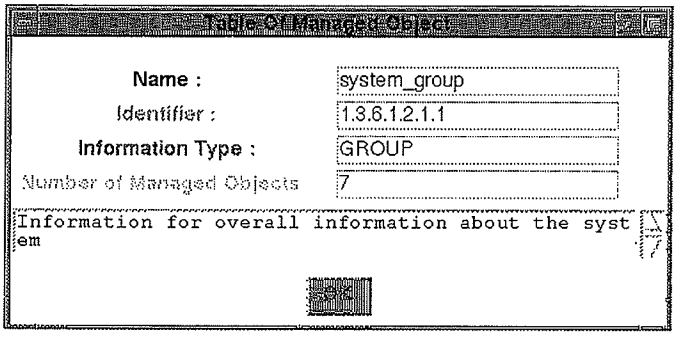


Figure 8.8 Managed Object/Device/Group information pup-up windows

8.4 FEATURES OF THE SOFTWARE

The network traffic monitoring software consists of the following features which are completely implemented and several that will be implemented [some of the features are not fully operating at the time of writing report]. The features include:

- A friendly graphical user interface.
- Capabilities of retrieving managed object information in the form of:
    * Table of managed objects
    * Group of managed objects
    * Individual managed objects
- Managed object values can be displayed in both graphical form and textual form. Only those managed objects with syntax "Counter" can have their value displayed graphically.
- Managed object with access "read-write" can have their values be set to desired values by authorized persons, such as the system administrator/manager.
- The system is designed using an object oriented approach OMT, as such system modifications can be made with little effort.
- The system can be expanded with the addition of new devices.
- Managed object values can be polled at different time intervals, set manually by the user.
- The values of managed objects can be retrieved and stored in text files for further analysis.

8.5 CONCLUSION

This chapter has introduced the implementation of the system described in Chapter 7. It presents the organization of the software and windows for each step of the execution process.

The software is very dynamic; many kinds of information of the device can be observed simultaneously. Only one of the devices within the network can be monitored at a time. However, the software could be extended so that multiple devices could be monitored simultaneously. The network can be changed by modifying the network configuration file. In the next and final chapter, it includes the conclusion of the project, final thoughts about the project and suggestions for further study.

CHAPTER 9
CONCLUSION

In the first part of the thesis, it has given a general introduction to the topics covered in the thesis. This starts with the introduction to one methodology of designing and analysing a software system using concrete objects to describe the system. The methodology discussed is the Object Oriented Technology, OOT. The selected object oriented technology is the Object Modeling Technique, OMT, which is studied and discussed in detail. Advantages of OOT are also addressed. OMT consists of 3 models and each of which describes an aspect of the system. The object model of OMT describe the static structure of the objects and the relationships between the class in the system. The dynamic model of OMT describes the dynamic behaviour of each class during the operation of the system. The last OMT model is the functional model, which describes the computation/evaluation of the processes of the system. Each of the models is discussed and demonstrated with examples. The thesis next introduces the software package, Software through Picture/Object Modeling Technique StP/OMT, which is a graphical tool used to develop a software system using the OMT approach. StP/OMT does syntax, consistency and completeness checking for the designed system. It also provides a C++ generator for the designed system based on information from the model object of the system. The C++ generated is a template for the classes and their relationships defined in the object model. An example of the object model and how the C++ code is generated using the software StP/OMT software packages and manually is demonstrated.

The second topic covered in this thesis is the introduction to the newly developed protocol for telecommunication, the Asynchronous Transfer Mode (ATM). ATM provides an efficient medium for various bandwidth and multimedia information transportation. The basic concepts of ATM are briefly addressed.

Lastly, the concepts of network management systems are discussed. This provides a back ground of the types of management system, the architecture of management systems, and the role of network management in today networking. One of the protocols for system management studied in more detail is the Simple Network Management Protocol (SNMP). The core data structure in the SNMP is the Management Information Base (MIB) which contains the information about each of the devices in the network. The information concerning the device can be read and stored for further studies or they can be controlled by an authorized person.

After all the above topics were discussed or review, I have proposed a design and implementation of an network management system for an ATM -Based LAN. The network selected is located at **T**elecommunication **R**esearch **Lab**oratories (**TRLab**s) in Winnipeg, Manitoba. The design and analysis for the network management are accomplished using the OMT methodology. The developed OMT object models, OMT dynamic models and OMT functional models of the system were discussed and presented. The implementation of the design was successfully completed.

In conclusion of the thesis, OMT is an efficient and advantageous approach not only for software system development but it is also an appropriate approach for different types of system development, such as hardware design and implementation. OMT provides an effective methodology for implementation and modification of code. The developed man-

agement system can be used to study the traffic within the network combining existing information collection tools with a usable interface on a object model platform. These tools are particularly important for newly developed technology like ATM, as its behaviour needs more study to improve its performance and provide better services.

The software developed provides for only a traffic monitoring. It does not provides other functions of an network management system, such as network security management, network account management, or network behaviour analysis. I recommend the study of ATM network management can be further explored with the continuation of development in object oriented technology. These studies can improve and complete a more sophisticated ATM management system not only at local level but global level.

REFERENCES

[1] Alles A., "ATM Networking", Cisco System Inc., May 1995.

[2] ATM Forum, "ATM User Network Interface Specification", Version 3.1, Sept. 1994.

[3] Aidarous S., Plevyak T., "Telecommunication Network Management In The 21th Century, Techniques, Standards, Technologies And Application", Institute Of Electrical And Electronics Engineer Inc., 1994.

[4] Armitage G. J., "The Application Of Asynchronous Transfer Mode To Multimedia And Local Area Networks", Ph.D Thesis, Jan. 1994, Department Of Electrical And Electronics Engineering, University of Melbourne, Australia.

[5] Bapat S., "Object Oriented Networks, Models For Architecture, Operations And Management", P.T.R Prentice Hall Inc., 1994.

[6] Batubana M., McGregor A. J., "An Introduction To B-ISDN And ATM", Sept. 93, Department Of Robotic And Digital Technology, Faculty Of Computing And Information Technology, Monash University, Australia.

[7] Black U., "Network Management Standards, The OSI, SNMP And CMOL Protocols", McGraw Hill, 1992.

[8] Booch G., "Object Oriented Analysis And Design With Applications", Benjamin/ Cummings Publishing Company Inc., 1994.

[9] Booch G., "Coming Of Age In An Object Oriented World", IEEE Software, Nov. 1994, pages 33-41.

[10] Cavanaugh J. D., Salo T. J.,"Internetworking With ATM WANs", Dec. 12 1992, Minnesota Super Computer Centre Inc.

[11] Chao H. J., Ghosal D., Saha D., Tripathi S. K., "IP On ATM Local Area Network", IEEE Communication Magazine, August 1994, pages 52-59.

[12] Dybeman D., Goguen M., "ATM-Forum 94-0471R12, PNNI Draft Specification", ATM-Forum, 1994.

[13] Elzekker Th. Van, "Integrating Method Objects Into The Object Oriented Life Cycle Approach", M.Sc Thesis, Sept. 1994, Software Engineering Group, Department Of Com-

puter Science, Faculty Of Mathematics & Natural Science, Leiden University, The NetherLands.

[14] Farnandez J., Winkler K., "Using SMI To Model SNA Network", IEEE Communication Magazine, May 1993, pages 60-67.

[15] Farkouh S. C., "Managing ATM-Based Broadband Networks", IEEE Communication Magazine, May 1993, pages 86-98.

[16] Fischer W., Wallmeier E., Worster T., Davis S. P., Haytey A., "Data Communication Using ATM: Architectures, Protocols And Resource Management", IEEE Communication Magazine, August 1994, pages 24-33.

[16] FORE Systems, "Programmer Reference Manual For AALI Interface", Nov. 1994, Software Version 2.3.

[18] FORE Systems, fore_adapter.mib, v 1.85, 1993-1994.

[19] FORE Systems, fore_switch.mib, v 1.320, 1993-1994.

[20] Fowler H. J., "TNM-Based Broadband ATM Network Management", IEEE Communication Magazine, March 95, pages 74-79.

[21] Kim J. B., Suda T., Yoshimura M., "International Standardization Of B-ISDN", Computer Network And ISDN System, v.27, no.1, Oct. 1994.

[22] Klerer S. M., "System Management Information Modeling", IEEE Communication Magazine, May 1993, pages 38-44.

[23] Kumar S., Aylor J. H., Johnson B. W. and Wulf W. A., "Object Oriented In Hardware Design", IEEE Computer, June 1994, pages 64-70.

[24] Ly F., Smith A., Noto M., Terink K., "Definitions Of Supplemental Managed Objects For ATM Management", June 1995.

[25] Marchisio L., Ronco E., Saracco R., "Modeling The User Interface", IEEE Communication Magazine, May 1993, pages 68-74.

[26] May M. D., Thompson P. W., Welch P. H., "Network, Routers And Transputer: Functions, Performance And Application", INMOS Limited, 1993.

[27] McDysan D. E., Spohn D. L., "ATM: Theory And Application", McGraw Hill, 1995.

[28] Mellquist E. P., "SNMP++, An Object Oriented Approach For Network Management Programming Using C++", Revision 2.1, Hewlett Packard, 1994.

[29] Meyer B., "Object Oriented Software Construction", Prentice Hall International (UK) Ltd., 1988.

[30] Minoli D., Vitella M., "ATM And Cell Relay Service For Corporate Environments", McGraw Hill Inc., 1994.

[31] Newman P., "ATM Local Area Networks". IEEE Communication Magazine, March 1994, pages 86-98.

[32] Newman P., "Traffic Management For ATM Local Area Networks", IEEE Communication Magazine, August 1994, pages 41-50.

[33] Ousterhout J. K., "Tcl And The Tk Tool Kit", Addison Wesley Publishing Company, 1994.

[34] Perkins D. T., "Understanding SNMP MIBs", Revision 1.1.7, Sept 1993.

[35] Pras A., "Network Management Architecture", Ph.D Thesis 1995, The University Of Twente, The Netherlands.

[36] Request For Comment RFC 1155, McCloghrie K., Rose M., Networking Group, May 1990

[37] Request For Comment RFC 1156, McCloghrie K., Rose M., Networking Group, May 1990

[38] Request For Comment RFC 1157, Schoffstall M., Fedor M., Davin J., Case J., Networking Group, May 1990.

[39] Request For Comment RFC 1213, McCloghrie K., Rose M., Networking Group, March 1991.

[40] Request For Comment RFC 1695, Ahmed M., Tesink K., Networking Group, August 1994.

[41] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., "Object Oriented Modeling And Design", Prentice Hall Inc., 1991.

[42] Scheckkerman E. J., "An Analysis Of Simple Network Management Protocol Version 2", M.Sc Thesis, June 1993, University Of Twente, The Netherlands.

[43] Serre J., Lewis P., Rosenfeld K., "Implementing OSI-Based Interface For Network Management", IEEE Communication Magazine, May 1993, pages 76-81

[44] Software Through Pictures, Object Modeling Techniques, Creating Models Manual Release 1, 1993-1994, Interactive Development Environment (IDE).

[45] Stallings W., "SNMP, SNMP v2 and CMIP, The Practical Guide To Network Management Standards", Addison Wesley Publishing Company, 1993.

[46] Stallings W., "ISDN And Broadband ISDN With Frame Relay And ATM", Prentice Hall Inc., 1995.

[47] Stevenson D. W., "Network Management, What It Is And What It Isn't".

[48] Terplan K., Atre S., "Effective Management Of Local Area Networks, Functions, Instrumental And People", McGraw Hill Inc., 1992.

[49] Terplan S., "Communications Networks Management",P.T.R Prentice Hall Inc., 1992.

[50] Theobalt D., "Tcl/Tk In A Nutshell", FZI Report, July 1993.

[51] Togtema J. W.,"Specification And Implementation Of Network Management Service Based On SNMP v2", M.Sc Thesis Jan. 1993, University Of Twente, The NetherLands.

[52] Yemini Y., "The OSI Network Management Model", IEEE Communication Magazine, May 1993, pages 20-29.

[53] Williams M. J., "ATM - What Does It Mean?", University Of QueenSland Prentice Centre, Australia.