Managing Heterogeneous Networks - Integrator-based Approach -

Jose Neuman de Souza[1]

MASI Laboratory UPMC,
45, Av. des Etats-Unis,
78000 - Versailles, France.

Ahmed Patel[2] and **Bharat Bhushan**[3]

Computer Networks and Distributed
Systems Research Group,
Department of Computer Science,
University College Dublin,
Belfield, Dublin 4, Ireland.

Abstract

This paper discusses an object oriented approach for network management. The goal of this paper is to explain, in brief, a real example of an Integrated Network Management (INM) system. One of the major requirements when looking at information transfer between the managed network and the management system is to mask the heterogeneity of the underlying resources. As an example of the unification of heterogeneous networks, a system called Integrator has been built and is operational. The Integrator is a mechanism that provides an object oriented interface to the user (human or network management application programs) in order to offer a homogeneous view of a world (set of heterogeneous domains) through a model (depicting a formal information view). The Integrator uses two agents to communicate with underlying network elements: an SNMP[4] agent accessing TCP/IP parameters for an Ethernet network through a SNMP daemon and an X.25 agent[5] doing the same for X.25 parameters through proprietary management software. The concepts of the Integrator have been applied in the EC project PEMMON.

Keywords

Integrator, Capsules, Controller, Semantic knowledge, Syntactic knowledge, Attributes, Model, Instance, Domain, Manufacturer Access Points (MAPs).

1. Introduction

Network Management can be defined as the co-ordination, monitoring and control of the distributed resources in a network. Various issues must be addressed when connecting different networks with different topologies, protocols, and networking models.

Interconnection of different protocol results in a heterogeneous network. The heterogeneity problem manifests itself when networks are connected to form an infrastructure for communication. Sources of heterogeneity include[1]:

---

[1] desouza@masi.ibp.fr [2] apatel@ccvax.ucd.ie [3] bharat@ccvax.ucd.ie

[4] Simple Network Management Protocol [5] A user written application running on an X.25 node

- Different Data Representations;
- Different Data Communication Networks;
- Different Protocols for Interaction; and
- Different Naming Domains.

In this paper we discuss some of the basic aspects, related with networks management, such as:

- how to mask the heterogeneity in managing a heterogeneous network?
- how to present the components of a heterogeneous network in an object oriented way?
- what methods should be adapted for the functional design of a Network Management system?
- how to specify the user and functional requirements for the design and development of an Integrated Network Management System; and
- how to access heterogeneous networks without getting involved into their complexity.

As an answer to above questions, we present an Integrated Network Management system called Integrator. An Integrated Network Management (INM) system is an application system which allows end-users to integrate, control, and manage heterogeneous networks involving a multiplicity of vendor processing and communication products[9]. The purpose of INM is to provide a single set of tools for administering all the network resources. For the functional design of an Integrated Network Management system, mainly three kinds of approach are considered.

- **Translator-based approach**

One of the proprietary systems is taken as reference. It offers standardized interfaces to which other management systems may be connected. In this case, the integration of the management systems is done within every sub-system that needs to adapt to the referential proprietary systems. This approach is the most widespread among manufacturers.

- **Standards-based approach**

In this case, all the network elements and management systems should use the same language and one common set of functions. This approach unifies the two previous ones. However, it is still far being completely established.

- **Integrator-based approach**

This uses a super-system which integrates the different underlying network management systems. It is the most common approach among users and service providers. In the paper we discuss about this approach. The Integrator is based on this approach.

The Integrator accesses two different networks, namely a X.25 network and a TCP/IP network. The Integrator integrates the two networks, accesses statistical information, and gives the information obtained to the user. The user may be an end-user or a set of performance management applications. The Integrator hides the heterogeneity of the networks and gives a single view to the user. The Integrator is based on a methodology called IDEA (Intelligence, Diagnostic, Expertise, Administration)[8]. IDEA is a method in which the different command syntaxes of many manufacturers are translated into a single view. We also discuss the tasks we have performed to implement the Integrator, i.e. from requirements gathering to SNMP and X.25 agent support. The Integrator is already operational and accesses statistical parameters from a TCP/IP network and an X.25 node.

2. Problems involved in managing networks

2.1. Context of management problems

The number of local, national, and world-wide communications networks is continually expanding. The facilities and complexity of these networks increase with technological progress. The user is confronted with problems in managing a network environment composed of these different networks and different services. Moreover, because the network is the nervous system of the communication structure, it is very important to plan, operate, control access to and maintain the networking system with dedicated management tools. Present management systems tend to operate only with the network equipment of a single vendor. Because of the variety of vendors, there are different, parallel, and mostly incompatible management systems in heterogeneous networks.

2.2. Problems due to heterogeneity

Problems due to heterogeneity arise in several specific areas.

- Interconnection: How should heterogeneous systems communicate with each other? How can systems and languages with different data representations be accommodated?
- Naming: How is naming provided in a heterogeneous environment? What objects can be named across systems? How are they named? How does the environment evolve as new systems and naming approaches are incorporated? These questions suggest the need for a Management Information Base (MIB).
- User interface: What kind of user interface should be provided? How would the user interface solve the problems? How would the interface represent the heterogeneous network (preferably with object class and attributes)?

2.3. Reasons for solving the problems

Resources are distributed.

The distribution of the network resources and software requires local as well as remote management.

The trend has changed.

In the early days of managing networks, vendors looked at network management in very narrow terms: diagnostics, testing, or control. The current situation is that network management does not simply mean management of computer networks by modems, multiplexers, PBXs for diagnostics, testing, and control. Network management involves accounting, performance, security, fault, and configuration management (as described in the OSI NM model)[5].

NM systems should not be protocol or vendor dependent.

Most networks and network services currently in use are managed by vendor-supplied management systems. These systems provide a large number of network management functions within the vendor domain. The systems are based on proprietary management protocols and management information.

A NM system should provide diverse services.

Users require a system that is able to provide diverse services. For example, such a system can greatly reduce the expertise needed to administer the diversity of networks. Some additional benefits that a system may provide are:

- a single user interface for network administration;
- a single set of common management functions for all networks;
- automatic translation between managed object definitions in different networks, reducing the expertise needed for network administration; and
- automatic maintenance of the relationship between managed objects.

3. Aspects of Performance Management

The term performance management may be interpreted in many ways. The simplest interpretation is to collect statistical and monitoring information about a network and assess it for measurement of performance.

3.1. Statistical information

This could be interpreted as general accumulated information about a network. The accumulation continues until the whole network is reset. During the time of accumulation the information is accessed and used for day-to-day or long term planning. Typical statistical information is packets received by a network node since some date.

3.2. Monitoring information

Monitoring information presents the status of a network. It includes alarms, network usage, and other similar information. Typical monitoring information would the number of calls in progress.

3.3. Performance Management - OSI view

OSI system management standards provide mechanisms for monitoring, controlling, and co-ordinating all the managed objects within open systems. The OSI NM Forum specifies performance management in the following manner[5].

- **Performance Management (ISO/DIS 7498/4-N/2673)**

Performance management provides the set of facilities needed to evaluate the behaviour of OSI resources and the effectiveness of communication activities. Performance management provides facilities to:

(1) gather statistical data; and
(2) maintain and examine logs of state histories.

- **Common Management Information Service (ISO/DIS 9595)**

Common Management Information Service (CMIS) is an Application Service Element (ASE) which is used by an application process to exchange information and commands for the purpose of system management. Basically, this standard defines a set of 10 service primitives that constitute the ASE, as well as attributes passed in each primitive[6]. The following 10 CMIS primitives form the basis for virtually all OSI management activities.

(1) Confirm-Event-Report
(2) Event-Report
(3) Confirmed-Get
(4) Confirmed-Set
(5) Set
(6) Confirmation-Action
(7) Action
(8) Linked-Reply

(9) Confirmed-Create
(10) Confirmed-Delete

4. Requirements for the Integrator

We define the requirements from the point of view of performance management of a network. The user of accessed statistical information may be either an end-user or a performance management application. The main requirements are as follows.

- The ability to access all the subnetworks and statistical information in the network, regardless of the OSI protocol suite used.
- The ability to access information from a wide range of network resources, from low-level devices (e.g. repeaters, modems) to intermediate systems. (e.g. bridges, gateways) to end systems (i.e. systems with full protocol stacks)
- The ability to maintain error log reports about a network or a component of a network.
- The ability to give traffic control information and routing information about a network or a component of a network.

The Integrator not only gives us facilities to monitor the performance but also gives facilities for network administration in an object oriented way. The administration of a network requires detailed knowledge of the managed network and the tools that help in maintaining the resources. The Integrator classifies the managed network into object classes. So, in addition to the requirements defined above, we have the following requirements.

- The ability to create and modify object classes in specified domains (e.g. the X25 and the IP domains)
- The ability to create and modify instances of an object class
- The ability to create and modify the knowledge part in the model for the object classes
- The agents must support the application. The SNMP agent and X.25 agent must access statistical parameters from a X.25 and a TCP/IP network
- The network management tools should decrease the network administrator's work load, providing a single set of tools for all networks to be managed.

5. The IDEA architecture

IDEA is an innovative architecture that proposes a methodological approach for handling heterogeneity when accessing manufacturer-dependent network management systems. The IDEA Integrator is a technique that offers means of interfacing to heterogeneous management functions and entities. It provides semantic and syntactic unification, thus achieving a single user interface[8]. Figure 1 shows the main functional components of the IDEA architecture. They are defined below.

5.1. Model

The Model is an abstract representation of the world under consideration. It provides the formal view of user information. The first step towards building the prototype is to build the model of the world.

5.2. World

The World is a heterogeneous collection of domains. It is a set of real, potentially, heterogeneous, elements to be handled. The two domains in the Integrator are X25 and Internet.

5.3. Domain

The Domain is the unit of heterogeneity. Within a particular domain all the information is homogeneous. Within a particular domain internal information is named and accessed in a unique manner.

5.4. External Attributes

These are the world's information entities as represented by the model, i.e. the external attributes enable access to the internal information of a domain without knowledge of their specific characteristic such as localisation and access methods.

5.5. Internal attributes

These are the domain's information entities as provided by a given MAP (Manufacturer Access Point), i.e. the field in the internal information that can be named and attached to the mechanism that knows how to access them.

5.6. Semantic module and semantic unification

A Semantic module is responsible for the semantic unification (unification of meanings of different commands). Semantic unification is a mechanism that performs the translation between a formal view of the information and a real view of the information. The semantic module unifies the different semantics (commands that are accepted by different management systems) to provide a single homogeneous view at the application level. In semantic unification a translation of the different internal attributes to one external attribute is done. A mapping between them is provided by the semantic knowledge which covers one of three situations for a given domain.

1. An external attribute exists as an internal attribute, i.e., there is a one to one relationship between them.
2. An external attribute corresponds to several internal attributes whose values are combined before in order to provide its value, i.e., it is a function of several internal attributes.
3. There is no mapping between an external attribute and any internal attribute

5.7. Syntactic module and syntactic unification

A Syntactic module translates from the particular syntax used to access each real domain to a single and homogeneous syntax to be used by the semantic module. This unification process generates real commands from user's request to drive or control a real network resource or to perform an operation on a network resource. At the same time, through syntactic unification, real responses are carried out to the semantic unification which in turn give the responses to the application level. In syntactic unification mapping of an internal attribute to a field of the internal information is done. The mapping between them is provided by syntactic knowledge. In syntactic unification, two possibilities may occur.

1. An internal attribute is directly mapped to internal information from a real access method, i.e., there is no need to parse the results from a command.
2. An internal attribute is part of internal information resulting from a real access method, i.e., the results from an access method must be parsed to mask something and provide only the field that corresponds to the internal attribute, based on the syntactic knowledge.

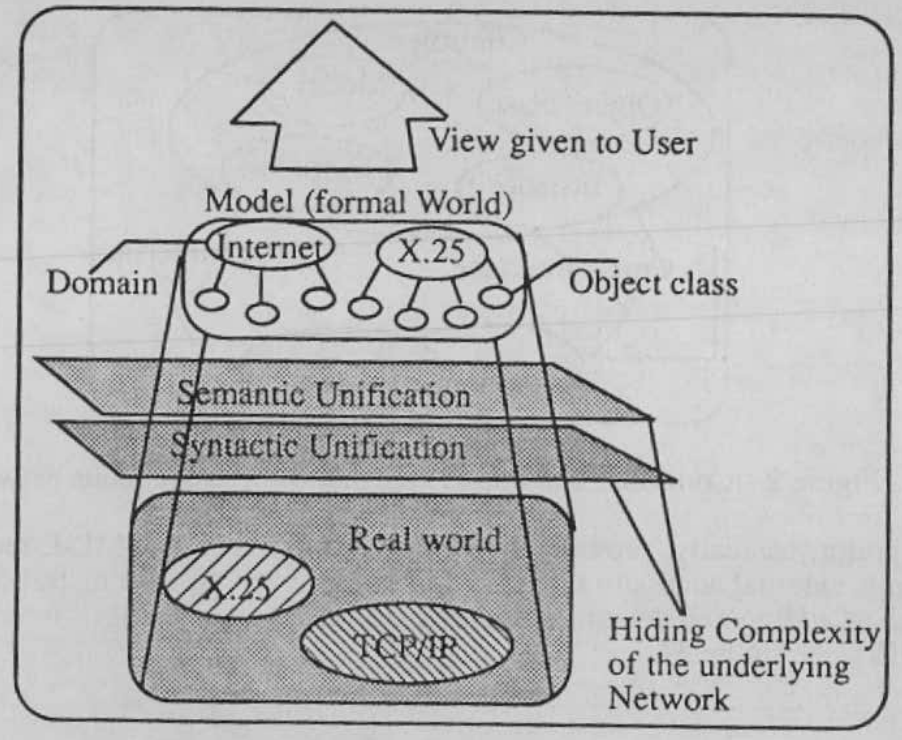

Figure 1 - Functional Components of IDEA Architecture

To validate the concept and software structure, the first prototype Integrator has been built. This prototype permits homogeneous access to statistical parameters of a heterogeneous network, through the integration of the respective agents. The heterogeneous network is composed of TCP/IP and X.25 elements. A SNMP agent and a X.25 agent allow access to the parameters.

In this example we see that the world is made of a heterogeneous network running over TCP/IP and X.25 protocols and X25 and IP are the domains that correspond to a X.25 and TCP/IP network respectively. The Integrator accepts get_requests in the form and syntax defined in the CMIS (without scoping and filtering for now)[6].

According to the chosen example, a model is created that describes the real world and provides the formal view of the user's information. This model is defined using the T1M1.5 Modelling Guidelines, adapted to our modelling requirements. The prototype has been implemented using the ANSA (Advanced Networked Systems Architecture)[3] architecture module and testbench facilities for the intercommunication. As a result, a module is implemented as a set of ANSA capsules. The information required from this model concerns object classes, their instances, and attributes. This information is accessed by two agents running over an X.25 switch and a TCP/IP network.

A controller module, shown in figure 2, manages a set of object instances. The controller, when handling a request, validates the request, finds the class of the referred instance and invokes the respective operation in the appropriate class. The set of supported instances is saved in persistent memory and loaded into a hash table during execution. There is an independent ANSA capsule for each class in each domain of heterogeneity. This module contains knowledge of how to perform operations in a homogeneous environment at a specific (semantic or syntactic) level.

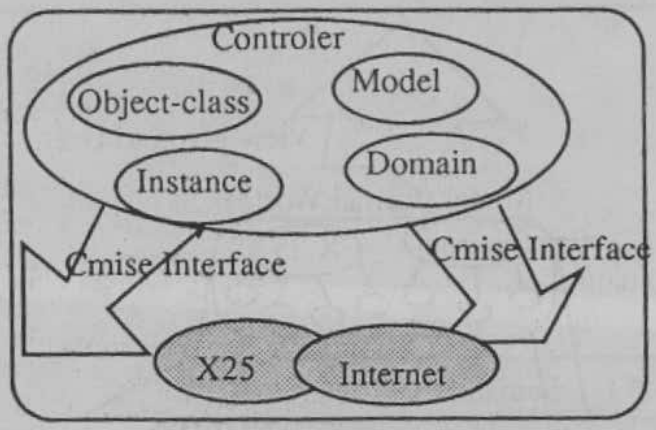

Figure 2 - Controller and knowledge part of heterogeneous network

The Integrator, basically, provides two kinds of interfaces: a CMISE request interface that permits external access to the managed objects; and an administration interface for tasks such as adding and deleting instances. However, other kinds of interfaces may also be provided (see figure 3).

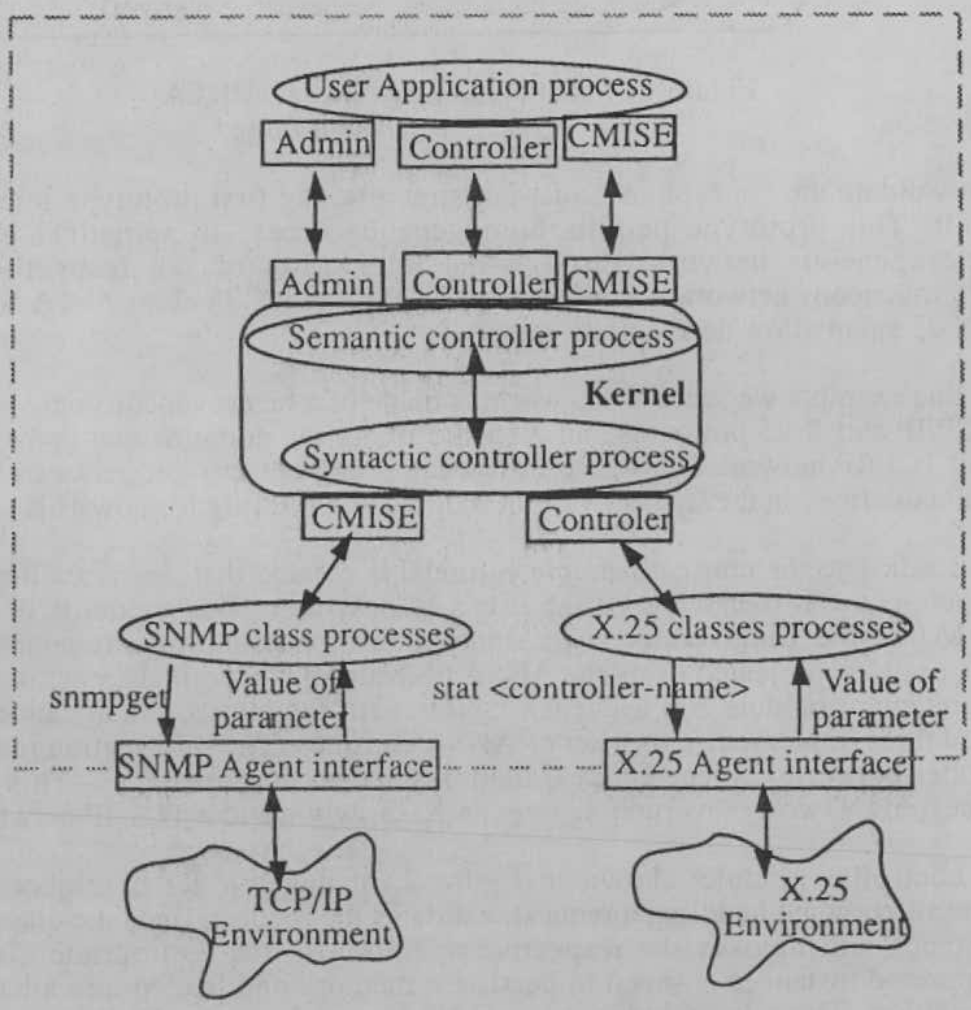

Figure 3 - Interfaces provided by the Integrator

6. Functional design of the Integrator

The design of the Integrator has the following functional requirements. Functional components of the Integrator are illustrated in figure 7.

- Definition and selection of semantic and syntactic level attributes
- Definition and selection of the attributes which determine the behaviour of respective object classes
- Selection of MAPs and their properties (since the properties selected are the attributes of managed object classes)
- Type of correspondence between syntactic level and semantic level attributes, i.e. the correspondence may be one-to-one or one-to-many
- Correspondence between object classes' attributes as defined in the MIB, and MAPs selection and parameter definition

6.1. Specification of the object classes and instances

When building the Integrator, the first thing to do is to specify the system model composed of a set of classes according to T1M1 templates. This specification will be used to construct the semantic and syntactic ANSA capsules. These capsules represent the implementation of the objects. Afterwards, we need to list the semantic attributes. In the X25 domain there are four object classes. They are listed below.

```
{ X25_HDLC001 IS INSTANCE OF X25_HDLC_LINE
        DOMAIN X25
        SUPERIOR root
}
{ IPprotocol001 IS INSTANCE OF IPprotocol
        DOMAIN INTERNET
        SUPERIOR root
}
```

Figure 4 - A T1M1 template defining the instances

- **X25_HDLC_Line object class**

The **X25_HDLC_Line** object class describes the packet level of a link between two X.25 nodes inside the X.25 network. It may contain one or more HDLC lines. The X25_HDLC_Line object class contains an instance named X25_HDLC_Line001.

- **X25_Network object class**

The **X25_Network** object class describes the packet level of the X.25 wide area network. The X25_Network object class contains an instance named X25_Network001.

- **X25_Link object class**

The **X25_Link** object describes the packet level of a link between an X.25 node and a DTE on the X.25 network. X25_Link object class contains an instance named X25_Link001.

- **X25_Node object class**

The **X25_Node** object class describes the nodes inside the X.25 network. The X25_Node object class contains an instance named X25_Node001.

The way in which we define the object classes for an X.25 network in the existing scenario is given in figure 15. In the Internet domain there are three object classes. They are listed below.

- **IPprotocol Object class**

The **IPprotocol** object class describes the Internet Protocol. The IPprotocol object class contains two instances named IPprotocol001 and IPprotocol002.

- **TCPprotocol Object class**

The **TCPprotocol** object class describes the Transmission Control Protocol. The TCPprotocol object class contains two instances named TCPprotocol001 and TCPprotocol002.

- **UDPprotocol Object class**

The **UDPprotocol** object class describes the User Datagram Protocol. The UDPprotocol object class contains two instances named UDPprotocol001 and UDPprotocol002.

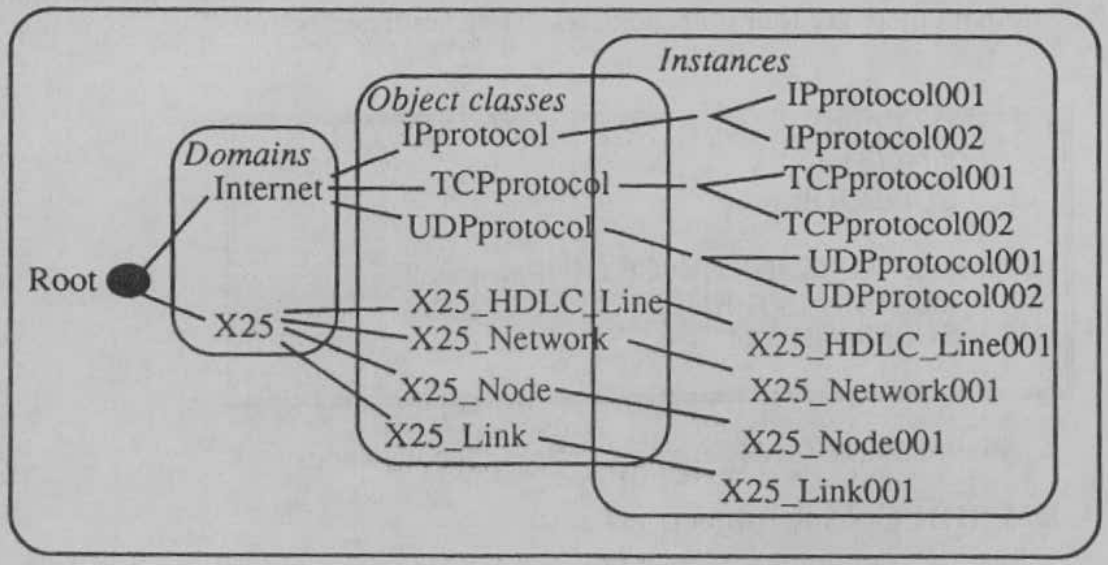

Figure 5 - Relationship between Domains, Object classes, and Instances

6.2. Attributes and their syntax

The term "syntax" is applied to the type of value an attribute can have. In the Integrator an attribute can have either type "string" or type "integer". This means that at the time of realising an attribute, the process that realises the attribute returns a type string value or type integer value. Two attributes are explained below.

```
{
X25_HDLC_Line OBJECT CLASS
        MUST CONTAIN { Line_speed }
Line_speed FUNCTIONAL_ATTRIBUTE
        WITH ATTRIBUTE SYNTAX STRING
}
{
IPprotocol OBJECT CLASS
        MUST CONTAIN { ipInReceives }
ipInReceives FUNCTIONAL_ATTRIBUTE
        WITH ATTRIBUTE SYNTAX INTEGER
}
```

Figure 6 - A T1M1 template defining attributes and their syntax

- **Line_Speed**

The attribute "Line_Speed" is one of the attributes of the **X25_HDLC_Line** class. It has type string. On execution, process **lsp()** (see the section "**X25 agent**") realises this attribute and accesses the value of the line speed of a HDLC line. It returns the line speed (a string) to the syntactic level.

- **ipInReceives**

The attribute "ipInReceives" is one of the attributes of the **IPprotocol** class. It has type integer. On execution, the agent SNMP (see the section "**SNMP agent**") realises this attribute and accesses the total number of packets received by the TCP protocol. This value (an integer) is returned to the syntactic level.

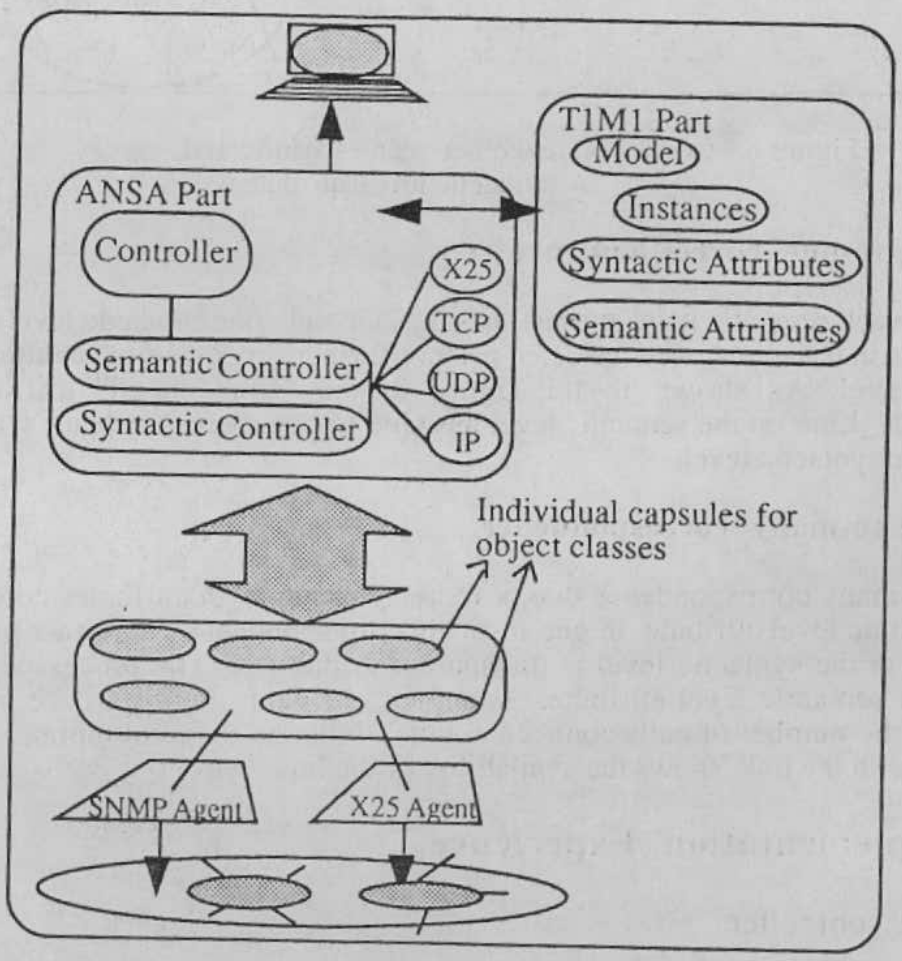

Figure 7 - Functional overview of the Integrator

6.3. Correspondence between semantic level and syntactic level attributes.

Figure 8 shows the correspondence between a semantic level attribute and a syntactic level attribute. The correspondence may be either One-to-one or One-to-many.

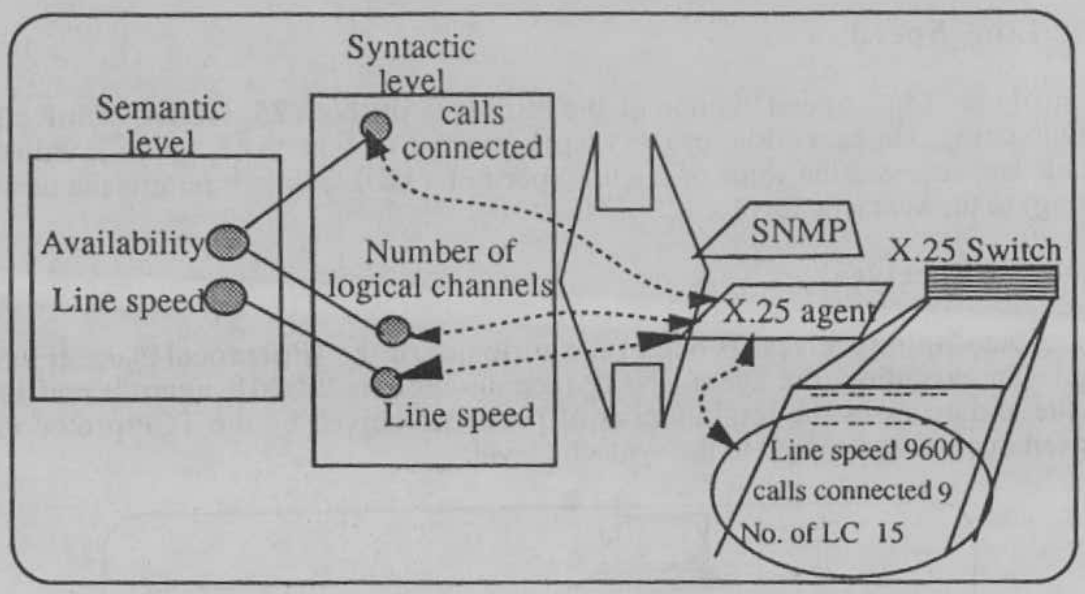

Figure 8 - Correspondence between semantic and syntactic level attribute

- **One-to-one correspondence**

In this case each semantic level attribute has one and only one syntactic level counterpart. This implies that the attributes selected are not further processed or manipulated at the syntactic level. As shown in the figure 8, the "Line speed" attribute of the "X25_HDLC_Line" at the semantic level has One-to-one correspondence with the "Line speed" at the syntactic level.

- **One-to-many correspondence**

In a one-to-many correspondence two or more syntactic level attributes correspond to a single semantic level attribute. In one-to-many correspondence the further processing of parameters at the syntactic level is transparent to the user. The processed parameters constitute a semantic level attribute. As shown in figure, the difference between two values, i.e. the number of calls connected with a link and the total number of channels associated with the link, shows the availability of the link.

7. Implementation Experience

7.1. The controller

The controller manages the set of instances. The semantic module and the syntactic module assist the controller to provide a homogeneous view of the network. As the controller comes into existence, it sublets its tasks to the semantic module and the syntactic module. These two modules appear in the form of two ANSA capsules: semantic controller and syntactic controller. For managing the heterogeneous environment, these modules require necessary information on domains, object classes, instances, attributes, and agents. This information is given as a knowledge to the semantic and syntactic modules. We divide the knowledge in two parts. The part of knowledge, which is used by semantic module, is known as the semantic knowledge. Similarly the part of knowledge, which is used by syntactic module, is known as syntactic knowledge.

7.2. The semantic knowledge

The semantic knowledge is a collection of processes that realise various attributes at the semantic level. The semantic knowledge is used for semantic unification. Each process in the semantic knowledge contains information about a semantic attribute. It, in turn, calls corresponding syntactic level process. Two examples of the semantic level processes are given in figures 9 and 10.

The semantic controller obtains the following information from the process se_X_Line_speed (see figure 9).

- Domain X25
- Object class X25_HDLC_Line
- Instance X25_HDLC_Line001
- Attribute Line speed
- Type String

```
int
Line_speed_process(instance, result)
{
BasicType = STR;
return get_syntax(instance, "Line_speed", type, result)
}
```

Figure 9 - Process se_X_Line_speed

Another example of semantic level process is given in figure 10. The information obtained from se_i_ipInReceive process is given below.

- Domain IP
- Object class IPprotocol
- Instance IPprotocol001
- Attribute ipInReceive
- Type Integer

```
int
ipInReceive_process(instance, result)
{
BasicType = INT;
return get_syntax(instance, "ipInReceive", type, result)
}
```

Figure 10 - Process ipInReceive at semantic level

7.3. The syntactic knowledge

The syntactic knowledge is a collection of processes that realise various attributes at the syntactic level. The syntactic knowledge is used for syntactic unification. Each process in the syntactic knowledge contains information about a syntactic attribute. It, in turn, calls corresponding agents. Two examples of the syntactic level processes are given in figures 11 and 12.

The syntactic controller obtains the following information from the process sy_X_Line_speed (see figure 11).

- Domain X25
- Object class X25_HDLC_Line
- Instance X25_HDLC_Line001
- Attribute Line speed

- Type String

```
int
Line_speed_process(instance, result)
{
        result = lsp();
}
```

Figure 11 - Process sy_X_Line_speed

The lsp() process is the lowest level process and realises the "Line speed" attribute. It connects the Integrator to the X.25 switch, accesses the "Line speed" attribute, and returns the value of the HDLC line speed from the X.25 switch. The functional diagram, given in the section "**X.25 Agent**", shows how the lsp() process works.

The information obtained from sy_i_ipInReceive (see figure 12) process is given below.

- Domain IP
- Object class IPprotocol
- Instance IPprotocol001
- Attribute ipInReceive
- Type Integer

```
int
ipInReceive_process(instance, result)
{
        snmpget(instance, attrname, buffer);
        result = mask(buffer);
}
```

Figure 12 - Process ipInReceive

This process calls the SNMP agent using the snmpget() command and accesses the parameters from the TCP/IP network. The output is parsed and the required attribute is chosen.

7.4. Capsules

Capsules are the managing processes. They realise the object oriented behaviour of the Integrator in a real environment. To access an attribute, ANSA instanciates the following capsules to manage various jobs.

- **Controller semantic**
- **Controller syntactic**
- **PERF/semantic/X25/X25_HDLC_Line/X25_HDLC_Line**

  This capsule runs for the **PERF** agent which is the name of the agent for the performance management applications. It manages the **semantic** level processes for the **X25** domain. It is assigned to manage the **X25_HDLC_Line001** instance of the **X25_HDLC_Line** object class.

This capsule calls the corresponding syntactic level process and wait until the value of "Line speed" is returned from the syntactic level.

- **PERF/syntactic/Internet/IPprotocol/ipInReceive**

  This capsule runs for the **PERF** agent which is the name of the agent for the performance management applications. It manages the **syntactic** level processes for the **IP** domain. It is assigned to manage the **IPprotocol001** instance of the **IPprotocol** object class.

The capsules at the syntactic level call the SNMP and the X.25 agents and wait until they get the value of the attributes from the network. After receiving the values of the attributes, capsules at the syntactic level return them to the corresponding capsules at semantic level.

8. Interfacing with the Agents

There are two agents associated with the Integrator: an SNMP agent and an X.25 agent. The SNMP agent is a proprietary agent while the X.25 agent is an application which runs on an X.25 node. The agents are the parts of the Integrator that manage the managed object within its local system environment. The agents perform management operations, i.e. Cmise_get, as a consequence of the management operation communicated from the Integrator.

The agents act as the lowest level functional component of the Integrator and provides access to the TCP/IP network and the X.25 switch. The TCP/IP network consists of Sun workstations and a gateway. The X.25 network consists of Sun workstations and an X.25 switch. The model permits access to the TCP/IP network and the X.25 switch components by using the SNMP and the X.25 agent.

The statistical information can also be obtained from the TCP/IP network and the X.25 switch. The X.25 switch has its own command language, which can be used to obtained output containing statistical information about the X.25 node. The X.25 agent hides this complexity from the user. The X.25 agent has been implemented using SunOS socket based communications. As shown in figure 13, it uses the SunOS socket facility to communicate with the X.25 switch. The main reason for the use of SunNet X.25 is that it enables us to open a socket in the X.25 address domain.

8.1. The X.25 Agent

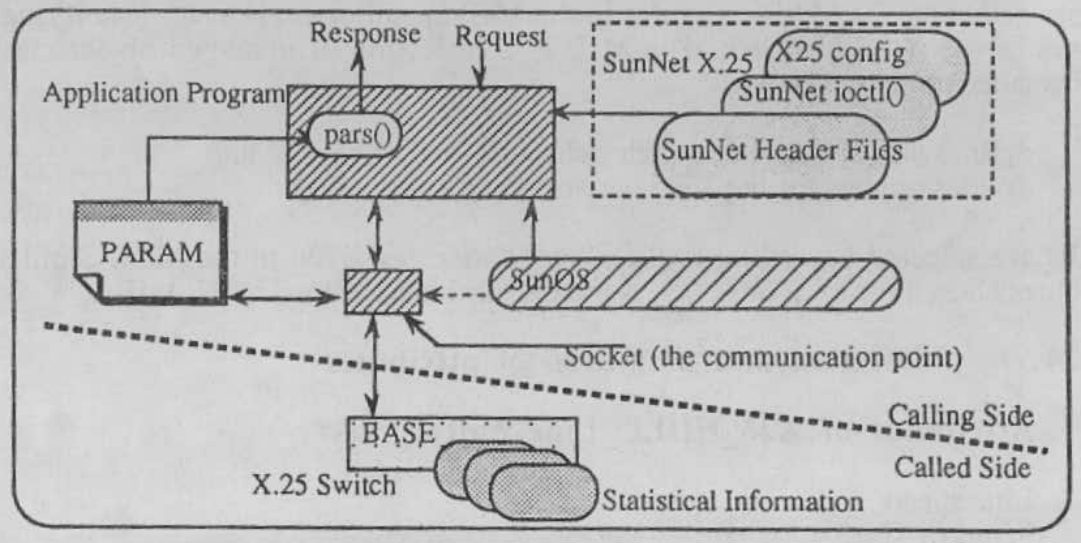

Figure 13 - Functional components of X.25 Agent

8.1.1. Distinguishing between parameters and attributes

By definition the terms parameter and attribute differ from each other but we use them interchangeably, depending on the context. For instance, if we discuss in the context of object classes and refer to X25_HDLC_Line object class, we use the term **attribute** for "Line Speed" (which defines the behaviour of the object class) but the same term "Line Speed" is refereed to as one of the **parameters** of the HDLC line (which also defines the behaviour of HDLC line). More explicitly, the value of **parameter** "Line Speed" of the HDLC line is accessed by and determines the **attribute** "Line Speed" of the X25_HDLC_Line object class.

8.1.2. Manufacturer Access Points (MAPs)

MAPs are the memory blocks associated with physical components of the X.25 switch. These memory blocks control the behaviour of associated components and give us statistical information about the components. As shown in figure 15, PS02 is the memory block that controls a HDLC line. The X.25 agent accesses MAPs and passes the information to the syntactic module. SunNet X.25 software provides functions that enable access to a remote X.25 system by a user program.

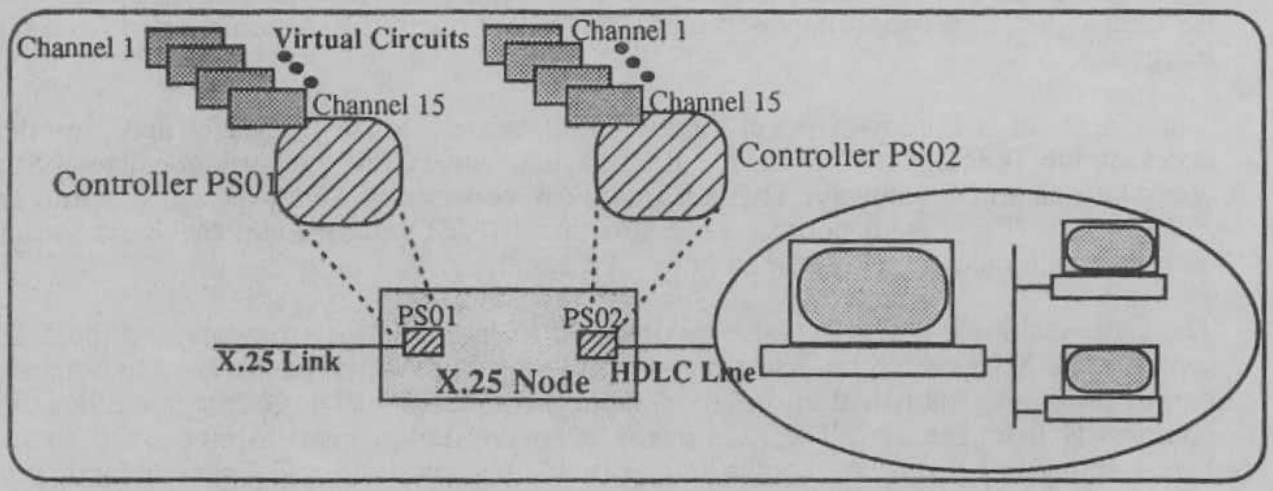

Figure 15 - Manufacturer Access Points from X.25 Switch

8.1.3. Management Information Base (MIB)

Before selecting the MAPs we refer to the MIB specification[16] for specifying object classes of the X.25 Network. The MIB is a collection of managed object classes for performance models. The MIB:

- defines all object classes, their definition, and attributes; and
- divides objects for the X.25 network into 5 classes;

MAPs are selected according to the object classes as given in the MIB. Similarly, the attributes chosen for each object class are the attributes defined in the MIB.

8.1.4. Selection and definition of attributes

- **Attributes of X25_HDLC_Line object class**

- Line speed
- Window size
- Cyclic redundancy check errors
- Retransmission
- Characters received

These are controller level parameters. For HDLC line parameters, controller PS02 is accessed. The commands **stat** and **exam** are used to obtain the parameters.

- **Attributes of X25_Network object class**

- Data frame received
- Data frame transmitted
- Total calls set up on the network
- Retransmission

For the X.25 network parameters, we access the virtual port BASE and send the **stat** command to it.

- **Attributes of X25_Link object class**

- Link type
- Packet size
- Cyclic redundancy check errors
- Retransmission
- Characters transmitted

For X.25 network parameters we access the virtual port BASE and send the **stat** command to it.

- **Attributes of X25_Node object class**

- Administrative status
- Data frames received
- Data frames transmitted
- Retransmission
- Current calls

The X.25 node parameters are accessed from virtual port BASE, since we have only one node, i.e. Switch, and so the parameters from the BASE port will be the parameters of the X.25 node.

8.1.5. Queries which the switch understands

Whatever information we seek from the switch is requested in the command language. We send commands to the switch and output of the commands is stored in a text file. The text file is parsed for the required attribute. The attribute is chosen and sent back to the syntactic module. We use the following commands to obtain the parameters.

- **Stat**: The Stat command is used to display the statistics and status information of a controller or channel.
- **Exam**: The Exam command is used to examine the configuration of a file, controller, or a channel.

```
STAT <controller PS01>
Type/ X25      CRC Errors/ 0   Retransmissions/ 300      Data In/ 0435

-------------------------------------------------------------------------
EXAM <controller PS01>
Name/ PS01     Speed/ 9600     Channels/ 15     K-Level-3/ 2
```

Figure 16 - Example of commands and their output

8.1.6. An algorithm for accessing parameters

We adopt the following algorithmic approach for accessing parameters. SunNet X.25 software provides logical access to the X.25 node.

- Open a connection (socket) in the X.25 address domain.
- Set the process group id for socket and signal handler
- Set the local address (host running the X.25 agent) and the remote address (the X.25 switch).
- Assign the address of remote host (as name) to the socket.
- Pass the address of remote host to the socket.
- Bind the address of remote host (as name) to the socket
- Set the Q-bit for communication (The Q-bit is the qualifier bit and qualifies the data in data packets.)
- Go to the databases (files) of the X.25 switch.
- Send a query (command(s) of the X.25 switch).
- Receive the output.
- Parse the output and chose the required parameter
- Close the connection

8.2. The SNMP agent

Concerning the SNMP software, there are three public-domain packages available:

- the CMU SNMP distribution;
- the MIT SNMP development kit;
- the 4BSD/ISODE SNMP package.

In our prototype, we use either the MIT or the ISODE package. Figure 17 illustrates our laboratory example for the SNMP case study. The MIT development kit is a set of functions organised as a library to handle SNMP messages, SMI (Structure of Management Information) and MIB (Management Information Base) objects addresses. Using this library we can generate the following executable codes:

- **snmpd** - This command starts the SNMP agent that manages a sub-set of the MIB-II objects;
- **snmptrapd** - This command starts up a Unix daemon that is responsible for waiting for unsolicited messages (traps);
- **snmpget, snmpset** and **snmpgetnext** - These commands are used to send a standard SNMP message to a precise agent. Functionally the action, create and delete operations of the OSI CMIS can be performed by the single snmpset operation; and
- **snmptrap** - This command enables the sending process of an unsolicited message.

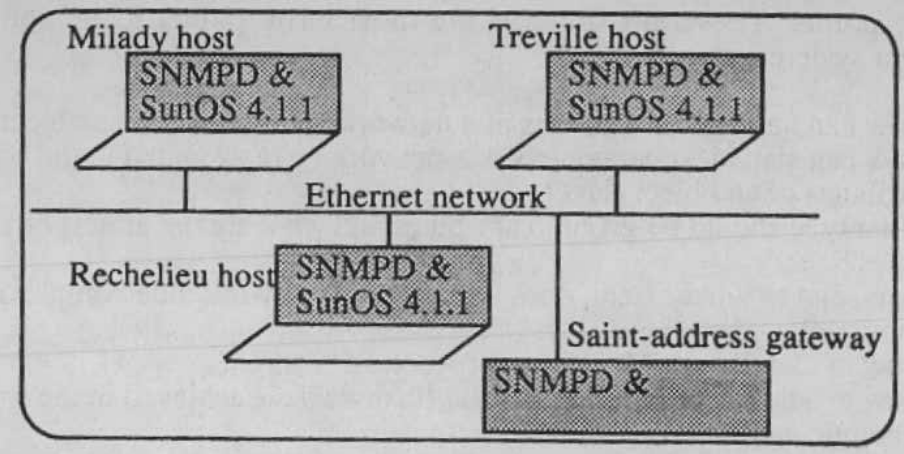

Figure 17 - Laboratory example of TCP/IP network and SNMP agent

Using only the SNMP software, to get the attribute value of the MIB SysDescr variable on the treville host from milady, we have to do the following:

snmpget -h treville.ibp.fr 1.3.6.1.2.1.1.1.0

The output of this command is:

Request Id: 0 Error: no Error Index: 0 Count: 1

Name: 1.3.6.1.2.1.1.1.0
Kind: OctetString
smxValueToText: Kind 2 Len 22
Value: "BSD Unix host treville"

We can see from the results of the snmpget execution that the field of interest is "**Value: "BSD Unix host treville**". This means that there is a need to mask everything else and return to the user only this line. In order to do that, using the integrator, we proceed in the following way:

- Firstly, we construct a model defining classes, instances and semantic and syntactic attributes. For instance, for the case above we have:

  class **Host** instance **Treville**
  semantic Attribute **System Descriptor** syntactic attribute **SysDescr**

- Secondly, we need to code at the syntactic level the real call to the SNMP agent and pass to it the necessary parameters. At this point we are implementing the real access. With the integrator running, the value of the SysDescr can be picked up by

  get(treville, System Descriptor)

After that the user will see only the last line of the snmpget output.

9. Conclusions

We have implemented a system which is based on the object oriented paradigm. In this system, components of a network are divided into object classes and the parameters defining behaviour of components determine attributes of object classes.

By analysing our experience we can give guidelines for a Network Management System which is based on object oriented design. We can summarise these guidelines in the

following points. These are some of the most basic points to be considered while designing a system.

- How can various components of a network be represented as object classes?
- How can statistical parameters of a network be represented in the context of attributes of an object class?
- What type should be given to attributes and what are the effects of specifying types?
- How, and to what extent, does an agent of a network hide complexity of the network?
- How to identify the Manufacturer Access Points?
- How to achieve level of abstraction (that we have achieved in the form of semantic and syntactic level)
- how to divide the requirements of performance management in the form of attributes?
- What information from the underlying network should be masked?

10. Acknowledgements

The authors would like to thank Ms. Olga Havel, Mr. Séamus Ó Ciardhuáin (from Department of Computer Science, University College Dublin, Dublin-4, Ireland) and Mr. Mauro de Oliveira and Mr. Serge Lalanne (from MASI Laboratory UPMC, Versailles, France) for their co-operation at various stages.

References

[1] Derok, H., "Information Transfer Within Advanced Logical Architecture", Roke M. Research Internal Document, June 1991.

[2] CCITT Recommendation X.25, "Interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminal Operating in the Packet Mode", 1984.

[3] ANSA Reference Manuals, Architecture Project Management Limited, Cambridge, UK, Jan. 1991.

[4] Rose Marshall T., "The simple Book - An Introduction to Management of TCP/IP based Internet", Printice-Hall New Jersey, 1991.

[5] ISO DIS 7498/4, "Information Processing Systems - Open System Interconnection - Basic Reference Model Part 4 - OSI Management Framework", ISO, Geneva, 1988.

[6] ISO DIS 9595-2, "Information Processing Systems - Open System Interconnection - Management Information Service Definition - Part 2: Common Management Information Service", ISO, Geneva, Dec.1988.

[7] ISO DIS 9596-2, "Information Processing Systems - Open Systems Interconnection - Management Information Service Definition - Part 2: Common Management Information Protocol", ISO, Geneva, Dec. 1988.

[8] Claude, JP et al "Management of Open Networks in a Heterogeneous Context", Laboratoire MASI, 78000 Versailles, France, 1991.

[9] Patel, A., "Comprehensive Network Management-Issues, Requirements and a Model" Proc IFIP TC6/WG 6.4a International Symposium on Management of Local Communication Systems, Canterbury, UK (18-19 Sept.. 1990).

[10] Kolas, F., "Management of Different Architectures and Environments under IBM's SNA", MIROR System Pty (Ltd).

[11] Paulisch, S., "Configuration and Performance management of LANs", Computer Science Department, University of Karlsruhe, D-7500, Karlsruhe 1.

[12] Data Communication, "The Simple Network Management Protocol", Feb. 1990.

[13] Datapro Report On PC Communication, "Network Management Concepts", March 1990.

[14] Datapro Report On PC Communication, "OSI-Based Network Management", March 1990.

[15] Notkin, D., et al, "Heterogeneous Computing Environment: Reports on the ACM SIGOP workshop on accommodating heterogeneity", CACM, Feb. 1987 Vol. 30, No. 2.

[16] ESPRIT II Project 5371, Performance Management and Monitoring of Open Network, "Specification of a Performance data model.