MASTER'S THESIS

Database Models and Architectures for Hybrid Network Management

by J. Valluri
Advisor: J. Baras

CSHCN M.S. 96-1
(ISR M.S. 96-2)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Abstract

Title of Thesis: Database Models and Architectures for Hybrid Network Management

Name of degree candidate: Jaibharat Valluri

Degree and year: Master of Science, 1996

Thesis directed by: Professor John Baras
Department of Electrical Engineering

As the complexity of communication networks increases, there should be commensurate advancements in the tools used to manage these networks. A Management Information Base (MIB) is a repository for information on all the network resources, services and customers and forms the heart of any Network Management System. Unfortunately the existing data models for networks are not powerful enough to represent all the heterogeneous resources available today, and at the same time capture all the network management functionality in a unified fashion. A related problem is that no model exists for storing sensor data collected from the network. Hence even though large amounts of data are collected from the network today, they aren't used to improve network performance. In this work we have developed data models for both of the above mentioned problems. Another issue that is of growing importance is the absence

of an architecture for the interoperability of heterogeneous database systems. Several companies would like to protect their investments in independent network management applications built on diverse platforms, and at the same time be able to integrate them. In this work we present an architecture for database integration. By developing a Configuration Management application on such a testbed, we demonstrate the ability of this architecture to perform the role of a "middleware".

Database Models and Architectures for Hybrid Network Management

by

Jaibharat Valluri

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1996

Advisory Committee:

Professor John Baras, Chairman/Advisor
Professor Michael Ball
Professor Nick Roussopoulos

Acknowledgments

I would like to express my deep regard for Dr.John Baras to whom I am extremely grateful. He has been a source of inspiration. He allowed me the liberty of exploring my areas of interest and his continued faith in my work granted me this excellent opportunity of being a part of a challenging research environment. His thrust on Systems Integration broadened my focus and exposed me to research problems in a wide variety of areas from Manufacturing to Speech and Image processing.

I would also like to thank Dr.Michael Ball and Dr.Nick Roussopoulos for always being available to discuss research issues and for consenting to be on my thesis defense committee.

Dr.Ramesh Karne, Steve Kelley and Sandeep Gupta were always approachable to discuss day to day problems and I would like to thank them for that. Kap Jang was extremely helpful with any problems in Graphical User Interface development.

This material is based on work supported by the NASA Center for Satellite and Hybrid Communications Networks under grant no. NAGW-27775, Hughes Network Systems/MIPS under grant no. MIPS 1122.23 and Loral/MIPS under grant no. MIPS 1316.25. I am also grateful to Mr. Richard Stanley of GTE Laboratories for offering me an internship for the summer of 1995. This intership was valuable industry experience.

Most of all I would like to thank my family without whom this work would not have been possible. They have been a source of constant support and encouragement. Finally I would like to thank all my friends.

Table of Contents

List of Tables

List of Figures

Database Models and Architectures for
Hybrid Network Management

Jaibharat Valluri

February 17, 1996

This comment page is not part of the dissertation.

Typeset by LaTeX using the `dissertation` class by Pablo A. Straub, University of Maryland.

Chapter 1

Introduction

As we move into the Information Age, there is an increasing emphasis on having an efficient communications infrastructure. A wide range of products and services is now available from a slew of vendors. These products and services operate on different technologies like ATM, Cellular, PCS, Satellites etc. Many of the services are driven by the ever growing entertainment industry.

Communication networks are developed around the different technologies. ATM networks are terrestrial and operate over diverse physical media such as fiber optic cables, coaxial cables and twisted pair copper wires. ATM is used both for Wide Area Networks (WAN), in order to develop high bandwidth backbones, as well as for Local Area Networks (LAN) within campuses etc. These networks are capable of carrying multimedia traffic. The cellular and PCS industries are driven by the need for wireless voice and data communication. There are already large subscriber bases for cellular services in most developing countries, and the demand in continually growing. Satellites serve as an ideal broadcast medium and have been widely used in the entertainment industry. They have also been used for long distance telephony. Satellites also have the ability to provide com-

munication facilities to areas that don't have a good terrestrial communications infrastructure or where terrestrial communication may not be an economically viable alternative. These are only a few examples of where Satellites are used.

Each of the technologies described above is driven by different factors and will continue to grow. Hybrid networks and services that combine two or more different modes of communication are already being deployed. As the existing technologies mature and consumer demands become more complex, there will be a growth in hybrid networking. Consider the case of video conferencing, where some of the parties involved may have wireless terminals, while others may be located in remote areas without any terrestrial lines. Such an application would require a hybrid network. For many user applications such as reading news or accessing large information databases, the communication requirements are essentially asymmetric. A Terrestrial-Satellite hybrid network can be used to meet such demands, where the user requests are sent terrestrially and the reverse channel is through the satellite.

Current Network Management tools and techniques aren't capable of managing such hybrid networks. Network Management Systems are organized hierarchically in layers, as will be seen in a later chapter. One of the layers in an Element Management layer. This layer is responsible for managing a specific portion of the network. For example all multiplexers or PBXs may be managed by this. Alternately, all elements within an administrative domain could be managed by such a system. The Central Management System or the Manager of Managers System is developed at the highest layer. This manages the entire network. A Management Information Base (MIB) which is a repository for all the information about the network is central to the development of Network

Management Systems. Before any data can be stored in the database however, a model has to be developed for storing this data. Existing models are inadequate and cannot capture the growing complexities of hybrid networks.

OSI has identified five different functional areas under Network Management: Fault Management, Configuration Management, Accounting Management, Performance Management and Security Management. A large number of Network Management Systems that exist today do not address all these functional areas. Even if the different areas are covered, they are typically managed by independent applications that may not interact. The different applications may not even have the same model for the network.

A first step to developing an Integrated Network Management System, encompassing all the NM functional areas is, interoperability between the different applications. All the management applications must use the same model for the network. This is especially important in the case of hybrid networks. The Satellite and ATM sub components may be managed separately at a lower management layer but at a higher layer, a Manager of Manager System is developed that has to manage the entire network. Hence, there must be a unified data model that addresses all the NM functional areas. This model is implemented on a central or distributed database system and all the different management applications should interact with this same database. This database then maintains all the information about the network, the services and the customers. In this work we attempt to develop such a unified data model.

Performance Management is an area where the focus has been very limited. Performance statistics are collected from various network elements in the network. These vast amounts of data pour into the network control center, but

currently very little is done with this data. Quite often this data isn't even stored in a database and is stored in flat files instead. Many network managers don't know what to do with the data, since much of the data may not be in a meaningful form and even if it is, they can't associate it with the appropriate network elements. Hence it is essential to store this data in such a way that they can easily be associated to their corresponding network elements. Performance data is potentially huge and it is very important to provide an efficient storage model for this data, so that little time is wasted in navigating through irrelevant data and operator queries can be answered as quickly as possible. It may also be necessary to map the incoming data to something that is more meaningful to the operator. No model exists for this currently. We address this issue in this work. To improve the management efficiency, it is important that the storage model be efficient and at the same time be integrated with the Configuration Management model.

An important consideration while developing the Integrated Network Management System is, the interoperability between the lower level management systems. It is possible that the network management systems for the terrestrial and Satellite sub components of a hybrid network were developed on two different database platforms. Hence the interoperability of databases becomes an important issue. This is also an issue for plain terrestrial networks as well. This problem is of interest to several companies today. They have developed different Network Management applications for their networks on diverse database platforms. Now they would like to develop an Integrated Network Management System for their network. But this is a difficult problem since the database platforms on which the applications were built aren't interoperable, hence the

different applications cannot interact. The naive solution is to have all the data transferred to one database. This is however not acceptable since the amounts of data are vast and it isn't a scalable solution. There is no existing model or architecture that can achieve this interoperability efficiently.

To summarize, two of the main problems being faced while trying to develop Integrated Network Management Systems are:

- the absence of a unified data model encompassing all the network management functionality
- the issue of interoperability between various commercial database systems

We have addressed both these problems in this work, and attempt to provide solutions for them. In addition to the above problems we have also developed a data model for the sensor data collected from the network.

This thesis is organized as follows: Chapter 2 discusses the benefits of the object oriented approach to modeling, gives a brief introduction to the Network Management functional areas, outlines the NM modeling requirements and develops an object-oriented data model for hybrid networks. Chapter 3 proposes two different Performance Management models and compares their relative merit. Chapter 4 presents an architecture for the integration of heterogeneous database platforms and an implementation of the architecture. Chapter 5 develops a Configuration Management Application on the Integrated Database platform and exhibits the usefulness of such an architecture. Chapter 6 presents the conclusions and opportunities for future work.

Chapter 2

Object-oriented modeling for Network Management

The Object oriented concept evolved from the programming language domain where it has led to significant efficiency in software development. It is now being widely adopted in other disciplines like databases and distributed computing.

Object orientation provides a methodology to abstract elements from the problem domain and model them as objects. Most often each object has a direct correspondence with a physical real world entity.

As mentioned earlier the networks of today are large and heterogeneous, with equipment from a large number of vendors. The object oriented modeling technique can prove to be very useful in reducing the exploding complexity of these systems.

Before discussing the various aspects of object oriented modeling it is useful to consider the various modeling paradigms, as defined by Wegner [15]:

1. Object based modeling: This paradigm requires that every element of the problem domain be an abstracted and encapsulated object. Each object

has a state, reflected by a set of attributes and a behavior, all together in one module. In addition there should be a well defined interface through which the rest of the system can interact with this object.

2. Class based modeling: In this paradigm all elements are modeled as objects, and in addition, all objects that are considered "similar" are said to belong to a certain **class** or **type**. Every object must belong to a certain class. A class can be considered a template for objects that are similar. All objects are instances of a particular class with specific values for their attributes. Object belonging to the same class share common properties.

3. Object oriented modeling: This is the richest paradigm and includes all features of the above two paradigms. In this paradigm, in addition to defining classes of different types, an inheritance is defined between classes. In other words, a new class called the **child** class can be created by inheriting the attributes and behavior of another class, which is called the **parent** class.

   Hence given a group of classes, we could extract the common properties from the different classes and create a new class, called **superclass** from which all the given classes could inherit. This process is called **Generalization** and is used while using a bottom-up approach to modeling. The reverse process called **Specialization** is also possible, where child classes are created by inheriting the properties of a parent class and adding new properties of their own. To see how this could be of use to us we consider an example. Consider a generic **ATM_Switch** class. We also need to model ATM switches from specific vendors, for which we might

create **Fore_Systems_ATM_Switch** and **Cisco_ATM_Switch** classes. These two switch classes have many attributes in common with the generic ATM_Switch. In fact, each has all the properties of the ATM_Switch and adds on some special properties of its own. Hence instead of replicating all the information in the two vendor specific switches we could just inherit that from the generic switch. This is an example of **specialization**. Depending on whether our approach to modeling is top-down or bottom-up we would use either specialization or generalization respectively. These would be specializations of the ATM_Switch class.

We will use the object-oriented paradigm in all our modeling.

2.1 Object Oriented Modeling concepts

- Abstraction: This is a mechanism by which, given a problem domain, only the properties of interest are extracted to form a model. Hence only the "essential" attributes and behavior are captured. For example, if we are modeling a Workstation for the purposes of Network Management, we wouldn't include as part of our model, the architectural attributes of a Workstation. e.g processor bus width or processor cache size.

- Encapsulation: This is a method of separating the internal representation and implementation of a class from the external view of the class. All interaction between this class and any other class takes place through a clean and well defined interface. This allows the internal representation of the object to be modified without affecting the rest of the system, given that the external interface of the object is kept the same. A class then

includes all static and dynamic attributes along with the operations that can be performed on and by that class.

Encapsulation is also used in the layering of networking protocols. Each layer has a well defined interface to its adjoining layers. Hence each layer forms a module that can be modified without affecting the rest of the layers.

Encapsulation is a very powerful construct, since once the external interface of a class is defined, the implementation of that class is not important.

- Relationships: Relationships can be defined between two different classes. This construct is very useful in the modeling of networks. For example, consider a Node class and a Link class. A Link in the network is either a point-to-point link or a multi drop Link. To model this we define a many-many relationship between the Node and Link classes. The relationship is many-many because each Link could be connected to several Nodes and vice versa.

  Note: The object oriented programming language used for implementation, may not support Relationships. In such cases relationships are implemented by storing embedded pointers to the conjugate class object. This results in a pair of conjugate pointers that need to be managed together i.e if one is modified the other also needs to be suitably modified.

- Inheritance: As discussed earlier, this is a method by which objects of one class can inherit attributes and behavior of another class. Inheritance is a special relationship called the **is-a** relationship. A child class is-a class of type parent. Using multiple inheritance one class can inherit properties

of many classes simultaneously. Inheritance and relationships can be used to provide a meaningful and well structured abstraction of the problem domain.

- Aggregation: This is also a special relationship and is also known as the **part-whole** relationship. A larger object is composed by **aggregating** smaller objects. This allows the creation of complex systems from simple systems. This concept is especially useful in manufacturing where complex components need to be modeled in terms of their parts.

  In networks a part-whole relationship exists between a line card inside a PBX and also between Line Interface Modules (LIM) and Data Port Clusters (DPC) in a Satellite subsystem.

2.2 Modeling for Network Management

There are various functional areas of Network Management. A more detailed description of the Network Management functional areas can be found in [11, 7].

2.2.1 Configuration Management

This is the most fundamental network management functional area. This management function is responsible for maintaining a record of all network elements in the network. It should also provide the operator with the ability to statically or dynamically configure any network element. In addition, the operator should have the capability to create new and delete existing "Managed Objects". Configuration changes may be necessary during installation of new components, resource reallocation or in the case of network faults.

The structural aspects of Network Elements are modeled as part of this functional area. For example a typical attribute for a modem card class would be the **baud rate**. The Node class would have a set of ports, location etc. The Link class would have set of nodes connected to it, capacity etc. We will see the details of these classes later.

It is evident how the notion of inheritance can prove to be very useful for modeling network elements. Specific nodes like ATM_Node and Router will inherit from the generic Node class discussed above.

2.2.2 Performance Management

Performance Management deals with Monitoring and Control of Network Elements in order to guarantee the Quality of Service (QoS) requirements of various customers.

- Monitoring: This involves gathering data from the various network elements in order to evaluate their performance. From the modeling perspective this would mean that each Network Element class must have operations that report the status of that network element, to a Management Application. This status reporting can be:
  - periodic reporting
  - reporting when polled
- Analysis: The raw data gathered from the network elements needs to be converted into a more meaningful form, so that it can be used as a measure of the quality of service. Typically a network element reports the **total**

number of packets received or the **total number of packets in error** since a fixed reference time. These numbers have to converted into quantities like **utilization** or **error rate**.

- Control: On the basis of the above analysis certain thresholds or parameters on the network elements may need to be modified in order to maintain satisfactory service. Hence certain operations must be available corresponding to each network element, that allow an operator to set or modify certain parameters for that element.

Modeling performance related properties of resources

Certain generic performance related quantities are often of interest to typical Performance Management functions. Depending on the resource to which they are applicable, they take on different names. Some of the quantities that are considered are:

- Throughput: It is essential to measure throughput both for short term as well as long term performance management. In the short term, if the operator notices that the throughput is low, then it could mean that there is a bottleneck somewhere in the network, and the operator could try to rectify the problem. The history of throughput values is stored and is used for appropriate resource allocation in the long term.

- Utilization: Utilization is also useful for short term and long term management. It is basically a measure of the work load on the various resources. It can help the operator detect overloaded conditions in the short term. Similar to throughput, the history of utilization values are also used for

long term resource allocation.

- Delay: Delay has several components, including transmission delay, buffering delay at intermediate nodes etc. From the user's perspective, end-to-end delay and connection establishment delay are important. However it would be in the interest of the service provider to determine the major component of end-to-end delay, and to keep it within tolerable limits, so that the desired **Quality of Service** may be maintained.

- Error Rate: It is imperative to monitor error rate and ensure that it is within tolerable limits. Different kinds of connections have different limits on the error they can tolerate. Voice connections can typically tolerate higher error rate as compared to data connections.

The parameters identified above are meant to be generic and can be applied to different entities. While modeling the performance of protocols like TCP/IP or X.25, the above identified quantities would translate to:

- Request rate: Most protocols have counters for **number of connection requests**, **number of successful connections** & **number of unsuccessful connections**. The request rate can be determined from these counters.

- Utilization: There is also a counter for **number of active connections**, and each protocol has a limit on the maximum number of connections it can have open at any given time. The utilization is determined using these values.

- Error rate: Many protocols monitor the **number of interrupted connections**. This is used to determine the error rate.

While modeling connections the quantities would translate to:

- Throughput: Each connection has a counter specifying the **number of bytes transmitted** or **number of packets transmitted**. The throughput is determined using this counter.

- Utilization: Utilization is usually determined for a link by dividing the traffic carried on the link by the capacity of the link. However in the case of ATM networks, where a certain amount of bandwidth may be allocated for a connection, Virtual Circuit (VC) or Virtual Path (VP), it may be useful to monitor the utilizations of these "Logical Links".

- Error rate: There are counters associated with the **number of errored packets** and the **average number of retransmissions per packet**. These are used to give a measure of the error rate.

It is worth noting that the OSI Network Management model does not specify any performance related information in the classes it specifies for the various "Managed Objects".

The existing Network Management systems using standard protocols like SNMP and CMIP monitor and store raw data in the form of 'counters' or 'gauges' mentioned above. It is left to the Performance Management applications to compute the above identified generic performance related quantities. The ultimate goal of performance management is to ensure that the QoS requirements are met. The quantities of interest for this purpose would utilization, throughput,

error rate, delay etc. The various counters and gauges have to be mapped to these quantities.

The networks in future will be very heterogeneous with multi vendor equipment and supporting a slew of protocols. There would be one Integrated Network Management Systems managing the network. The onus of computing QoS metrics from the various counters should not be on the main Network Management Application. Instead the Network Elements should include operations that return utilization, throughput, error rate etc, instead of returning just the counter values. In a true object oriented manner the network elements could just be objects returning these values. Different network elements have different counters or gauges depending on the protocols they are using. The Network Management Application should not have to keep track of the mapping between the counters and the QoS metrics for the various network elements. These operations that return QoS metrics would be "virtual" i.e the implementation would be different for each network element. In this manner the various heterogeneous NEs would present a uniform interface to the Network Management Application.

It is important to realize that quantities like utilization, throughput and error rate represent the Quality of Service for a "connection". The overall availability of the system is also a QoS measure but has a different time scale and would relate to a particular customer. It is useful to put into perspective the different QoS metrics. We can distinguish 3 time scales for which QoS may be monitored:

- per connection: This is a short period, typically of the order of a few seconds or minutes. It is easy to quantify the QoS metrics for each connection, as discussed above.

- per customer: This is what the customer perceives of a particular service. Typical examples of QoS measures are:

  1. overall availability of the system
  2. mean time to rectify faults
  3. cost to satisfaction ratio (satisfaction quantified in a particular manner)
  4. average time to talk to a customer service representative
  5. number of complaints about billing statements

  These are more important from a business perspective.

- per service: This is measured over the entire lifetime of a service and represents the overall "satisfaction" of the customers with that service. These measures are largely qualitative.

We will concentrate on modeling the QoS metrics on a per connection basis.

2.2.3 Fault Management

This management function deals with maintaining the network elements and the network as a whole, in a state of operation. In the case of a fault the following steps may be necessary:

- Fault detection: When a fault occurs many Alarms get reported to the operator. The operator has to sift through this information to determine the source of a fault. Certain intelligent techniques to correlate alarms using prior knowledge, may be used to not only reduce the amount of

information presented to the operator but also make it more meaningful. If the source of the fault isn't known the operator may perform tests on the NEs. Each NE should have operations that allow the operator to conduct tests and gather more information. The Alarms or Notifications are also generated by the NEs in response to certain network events.

- Fault isolation: Once the fault has been detected, the faulty portion of the network has to be isolated. An operator would reconfigure the network to render at-least part of the network operational.

- Fault rectification: This could be done remotely by an operator or off-line by a service representative. It may be just a matter of altering certain thresholds which the operator should be able to do from his management station.

- Restoration of service: Once the fault has been rectified the operator would restore the network to its original state.

Hence each network element should be capable of issuing 'Notifications' or 'Alarms' to the operator in response to 'Events'. The operator should also be able to invoke various operations on the NEs as described above.

An 'Event' class would model the generic attributes of any network event. Typical attributes would be:

- event name

- source of event

- notification to be issued to the operator in response to the event

Specific Events would inherit from this generic Event class. Similarly we would have a generic 'Alarm' class and specific Alarms would inherit from this class. Typical attributes of the generic Alarm would be:

- name of the alarm
- event/list of events causing the alarm
- probable set of alarms that may be triggered along with this alarm
- action to be taken or notification to be issued in response to this alarm

2.2.4 Security Management

This functional area is responsible for providing all the security related features such as:

- Authorized access: Only authorized users should be allowed access to various systems and services. Typically users have to enter some form of identification, like a password, that is verified against a password file, before being granted access to a system. Within the system itself certain files or services may be restricted to a certain set of users. For example only a superuser can modify the contents of a password file. An Access Control List (ACL) may be used to list the authorized users. In addition, each user may have a specific priority level that would determine his/her access privileges.
- Authentication: The recipient, of a message coming across the network may want to verify that the message originated at an authentic source.

Authentication is a service that the service provider may choose to offer. Digital signatures are sometimes used to provide authentication.

- Encryption: The users of a network may require secure transmission of data. Encryption is used to guarantee this. The original message is encrypted before being transmitted and decrypted at the receiver. Hence even if an unauthorized party does get access to the message, the actual contents of the message will not be compromised. There are certain algorithms like the DES algorithm that are used for this purpose. These algorithms require 'key' management which must be provided by the security management system.

2.2.5 Accounting Management

This functional area is responsible for maintaining a record of the usage of network resources by the customers. Each customer's usage must be monitored and the customer's billing record updated accordingly. Billing statements have to be periodically sent to the customers. This system should have a record of the tariffs for the different services. The tariffs for different services may vary depending on the time of day.

2.2.6 Management Layers

Orthogonal to the Network Management functional areas, there are conceptual layers into which a Network Management System is partitioned. Each layer uses the services provided by the lower layers.

1. Service Management Layer: This is the topmost layer. This layer manages the services provided by the service provider to the customers and ensures that the service agreements between the provider and customer are honored.

2. Network Management Layer: This layer is responsible for managing the network as a whole. In a typical scenario as shown in figure 2.1, the 'Manager' would be part of this layer. The details of the NEs are hidden from this layer. However relevant information may be requested from the NEs. On the side of the NE, the 'Agent' responds to the queries. This layer provides services to the Service Management layer by monitoring the QoS of the various customer connections.

3. Network Element Layer: This layer is concerned only with the management of a particular network element. The details of the NEs are managed at this layer. An Agent can only interact with a Manager and not with other Agents. Hence unlike the Network Management Layer this layer has no global functionality.

2.3 Requirements

Before doing the analysis and design of the data model it is useful to summarize all of our requirements.

- The network operator may need to configure the network. This may be necessary while adding or deleting network elements or while modifying

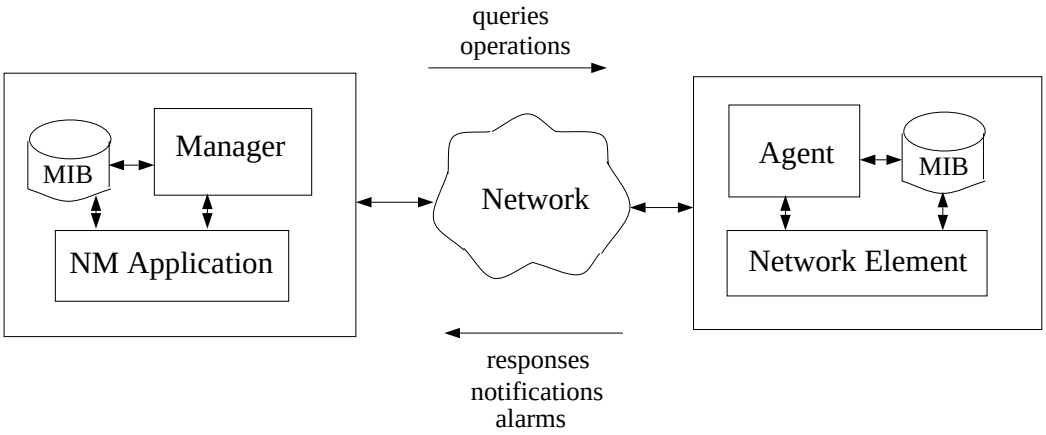

Figure 2.1: A typical Network Management System scenario

the current parameter settings on any NE. The operator may need to view the current settings of parameters on any NE.

- The operator monitors the performance of various network elements. As discussed earlier each network element should have the capability to report the generic performance related quantities identified earlier. However the operator may also wish to view the actual 'counter' or 'gauge' (SNMP parlance) values. Based on reports from various NEs, the operator may notice bottlenecks or "hot spots" in the network, which need attention. This may require some resource reallocation or changes in the configuration of the network.

- Historical records or Logs must be maintained for the performance statistics of all the network elements. Based on the historical performance statistics the operator may wish to reconfigure the network or add additional resources.

- The various 'Events' occuring in the network need to be monitored and recorded. Hence a Log is kept for all the Events that occur. Each entry in the log will have a time stamp associated with it. When a fault occurs in the network the operator may wish to see the sequence of events that could have led to the fault, for future reference. If later, the operator notices the same sequence of events he may act early and preempt a fault or potential bottleneck. This process could be automated also.

- Certain events can cause 'Alarms' that must be reported to the operator. There are numerous different kinds of alarms. The reaction of the operator to alarms is described in an earlier section.

- Once the fault has been detected the operator may wish to isolate the faulty part of the network for which appropriate operations must be provided on the NEs.

- After the fault is rectified the operator needs to restore the network to its original state.

- Customer records need to be maintained. Customers access certain services and have billing records. They may also have an access priority or level. The operator may wish to view the customer records.

- A record of the various services offered by the service provider needs to be maintained. Each service had a list of customers using that service.

Apart from the requirements described above, we find that implementation details dictate that we model certain support classes.

- The graphical representation of the network to the operator is very important and should not be underestimated. Hence the network layout needs to be modeled. Both the Graphical User Interface (GUI) and the persistent store should use the same model for the network.

- The operator may have certain "views" of the network, that combine information from different "Managed Objects". Though it is more of an implementation issue, it may be useful to model some of the typical views.

2.4 Notation

The Rumbaugh notation as described in [9], is used to depict the object model. In the Rumbaugh notation a class is depicted as shown in figure 2.2. The topmost rectangle is used for the class name. The attributes are represented in the middle rectangle and the bottom rectangle is used to show the operations that can be performed on or by the class.

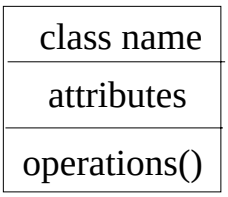

Figure 2.2: Rumbaugh notation for depicting a class

Figure 2.3 shows relationships between classes in the Rumbaugh notation. The top figure denotes a one-one relationship. The middle figure shows a one-many relationship and the figure at the bottom depicts a many-many relationship.

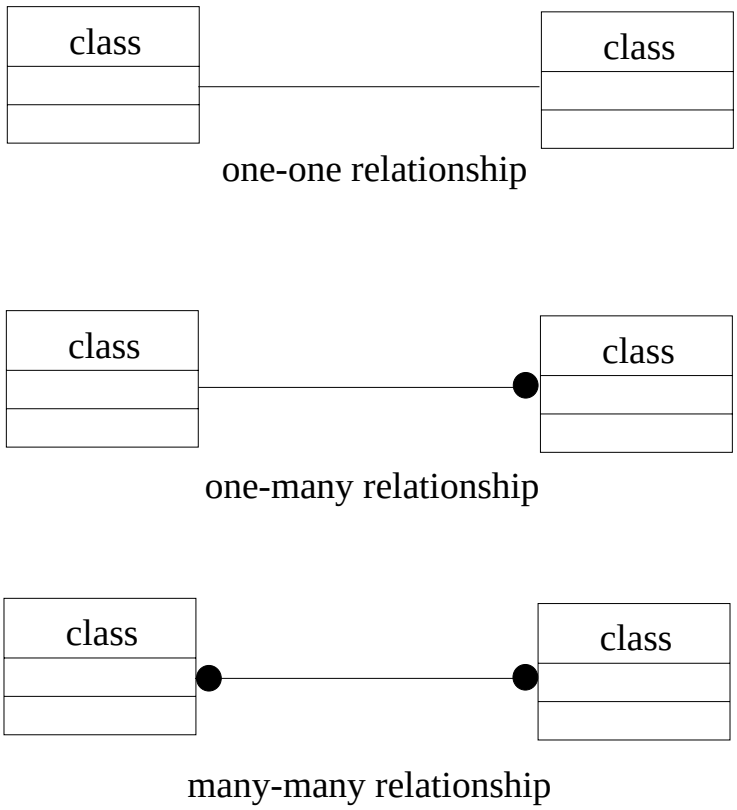

Figure 2.3: Rumbaugh notation for depicting relationships

Figure 2.4 shows the Rumbaugh notation for depicting inheritance between classes. An Aggregation or part/whole relationship is depicted in the Rumbaugh notation as shown in figure 2.5.

2.5 Data Types

SNMP supports extremely simple data types, while in CMIP, a Managed Object can contain extremely complex data types. To manage large complex networks we will need richer data types than those supported by SNMP. The different data types used in modeling the various classes are shown below:

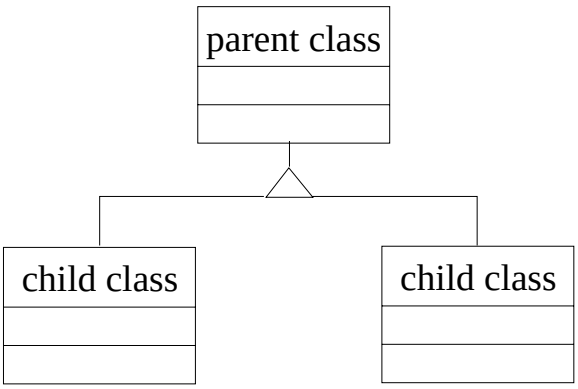

Figure 2.4: Rumbaugh notation for depicting Inheritance

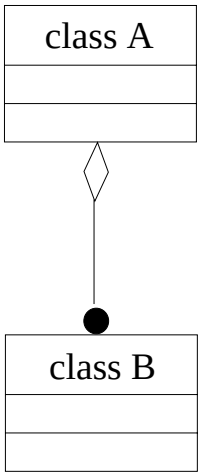

Figure 2.5: Rumbaugh notation for depicting Aggregation

- short integer: This type supports integer values in the range
  $-2^{15} <= i < 2^{15}$

- long integer: A variable of this type can take values in the range
  $-2^{31} <= i < 2^{31}$

- real: This data type represents any floating point value. The precision is determined by the language and platform on which it is implemented.

- Boolean: A variable of this type can on values **true** or **false**.

- String: This data type is a list of bytes of arbitrary length. Each byte is a character. It is usually used to represent text or names.

- Enumerated: This data type requires a list of values to be specified. A variable belonging to this type then take on one of the values specified in the list.

  e.g Host_Interface_type TokenRing, 10BaseT, 10Base2, 10Base5

  A particular node's interface will take on one of these values.

- record: This is a complex type and has multiple fields. Each field could be any of the types defined above or could itself be a record.

- List: This is a ordered collection of variables belonging to a particular data type.

- Set: This is an unordered collection of variables of a certain type.

A **class** is a combination of **attributes** and **operations**. Each attribute could belong to any of the above defined types or could itself be another class. Note: As we will see in the implementation of our model, we don't have any attributes which are themselves classes. Instead we store pointers to classes. A pointer is just a number and is of type long integer.

Every object should have a unique identifier so that any reference to an object can be resolved unambiguously. The class name and the id uniquely identify every object. For example Node:16 would refer to a Node class object with identification number 16. If we decide on having a distributed implementation of the MIB, we would build this on top of a CORBA compliant layer. This layer will have the task of determining the location of the object.

2.6 Data Model

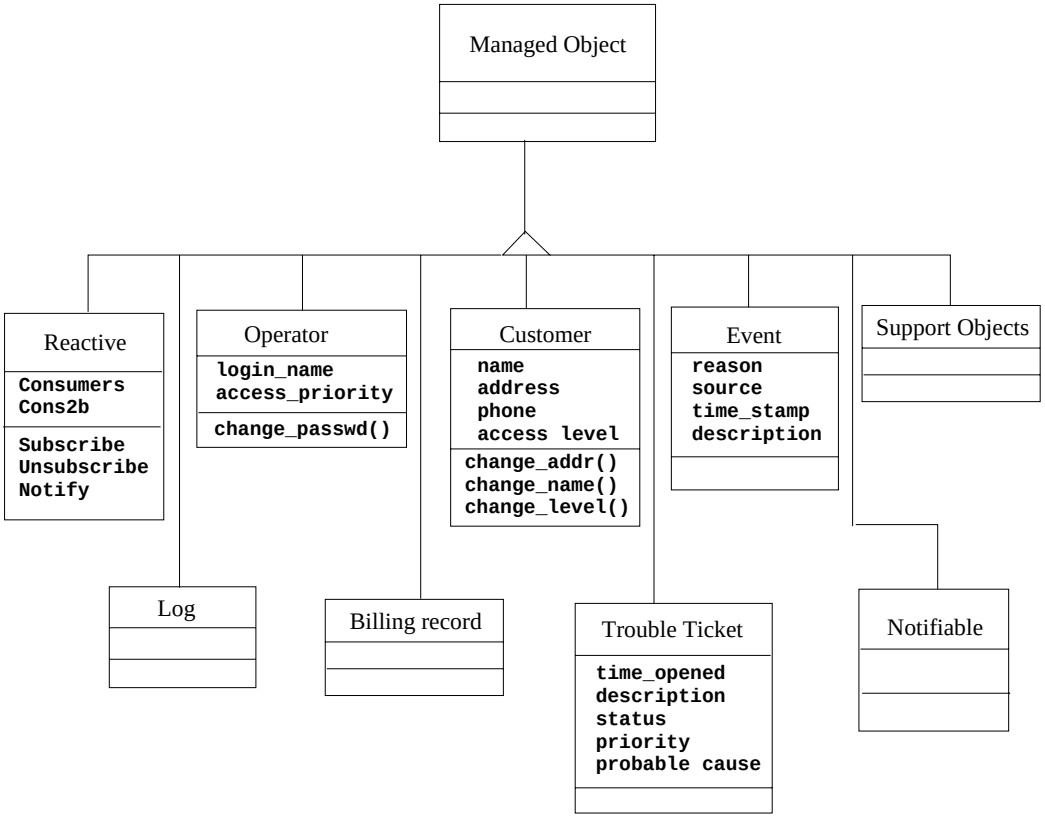

Figure 2.6: Managed Object Inheritance hierarchy

The Management Information Base (MIB) is central to the development of a Network Management system (e.g [1, 14, 13]). In this section we present an object oriented data model for the MIB. An earlier effort was made in this direction at Columbia University [16]. This work has been an extension of the data model developed in [3]. The MANDATE [4] and object oriented modeling principles [9, 2] have been followed while developing this model. The figures shown below represent the data model. Figure 2.6 shows the Managed Object Inheritance hierarchy. Figure 2.7 shows the Specialization of Events. Figure 2.8 shows the specialization of Support objects. Figure 2.9 shows the relationships

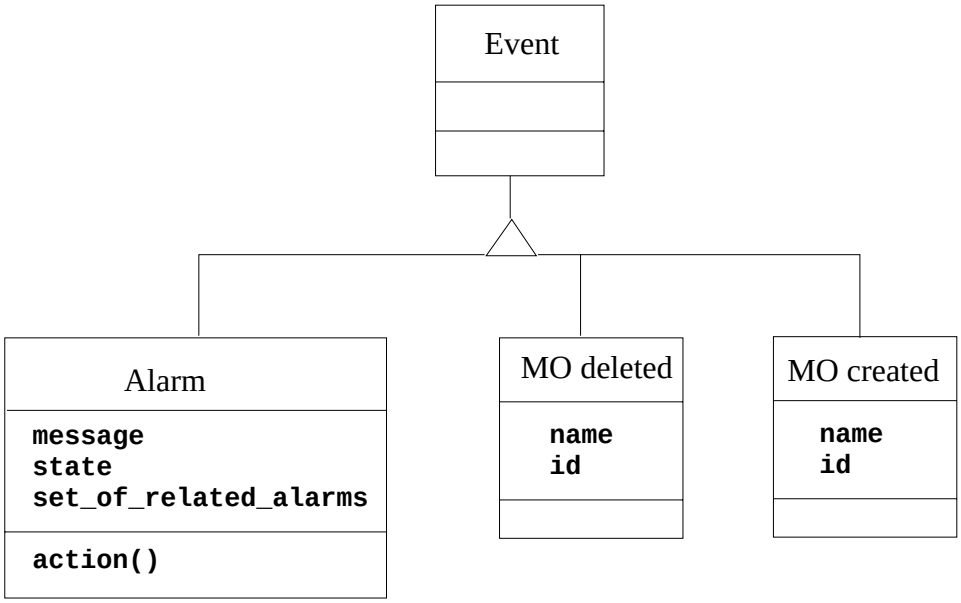

Figure 2.7: Specialization of Events

that the customer class has. Figure 2.10 shows the specialization of the Log class. Figure 2.11 shows the specialization of the Reactive class. Figure 2.12 shows the specialization of the Network Element class. Figure 2.13 shows the relationship between the Node, Link and Hardware Component and Statistics classes. Figure 2.14 shows the specialization of the Node class. Figure 2.15 shows the part/whole relationships between various types of Nodes. Figure 2.16 shows the specialization of the Link class. Figure 2.17 shows the relationship between Logical Links and Physical Links.

2.6.1 Class description

1. **Managed Object**: This class is the top of the hierarchy. It is the model for any class that can be managed in software by the Network Management System. All the other classes defined inherit from this class.

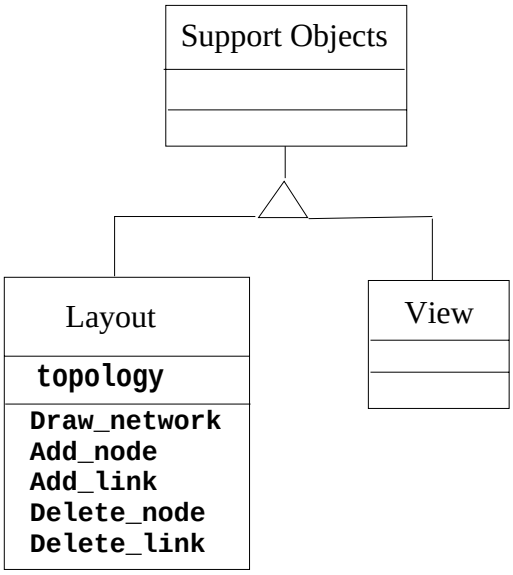

Figure 2.8: Specialization of support objects

Attribute

class_name

2. **Reactive**: This class inherits from the Managed Object class. This has been modeled in order to embed constraints into the model. Details of embedding rules into a database can be found in [5]. Any class that inherits from this class subscribe to 'Rules' and can receive notifications in case of particular 'Database Events'.

   Attributes

   consumers

   cons2b

   Operations

   subscribe()

   unsubscribe()

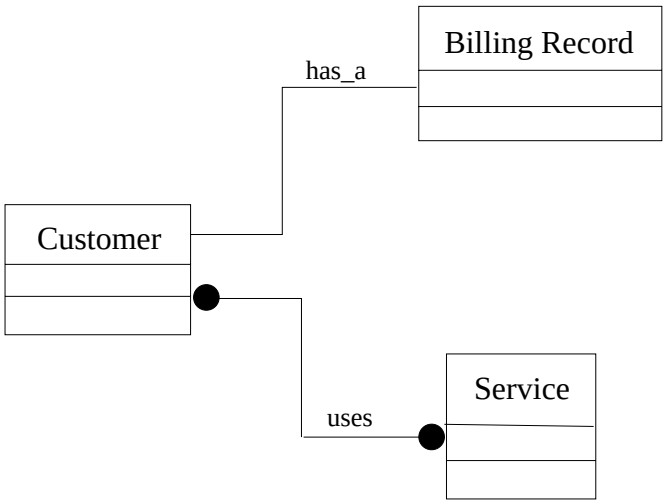

Figure 2.9: Relationships for the Customer class

notify()

3. **Log**: This class inherits from the Managed Object class. Log is meant to be an abstract class i.e there is no instance of this class. It is always important to maintain a historical record of different items like performance statistics, events, errors. The Log class is meant to model such a generic record.

4. **Operator**: Typically a Network Management Center has many operators who manage the network sitting at their terminals. Each operator may be executing a different network management application. Different operators may have different access restrictions depending on the task assigned to them.

   Attributes

   login name

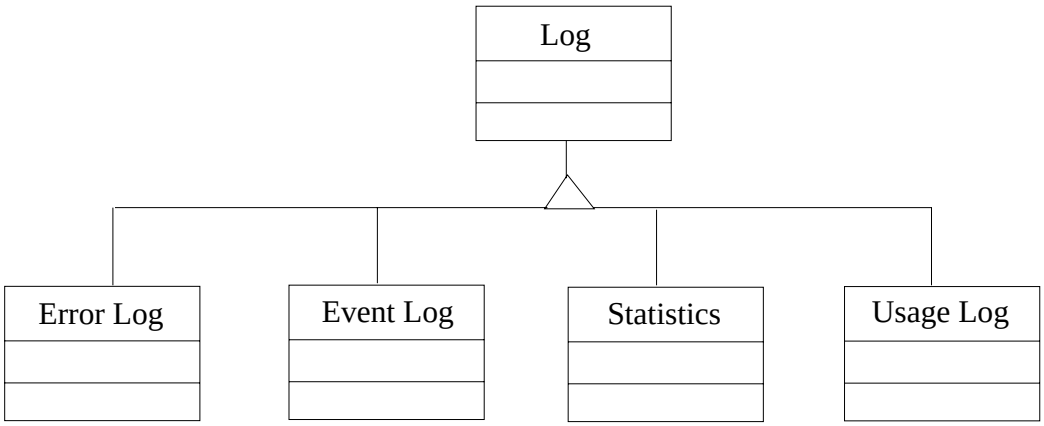

Figure 2.10: Specialization of the Log class

access priority

Operations

change_passwd()

5. **Billing record**: As mentioned earlier, a record of each customer's usage of the network must be kept. This class is meant to model that record. One Billing record is associated with each customer.

6. **Customer**: This class maintains a record of every customer using any service provided by the service provider. Each customer is associated with a billing record. Each customer also has a **uses** relationship with the various Services. This is a many-many relationship as shown in figure 2.9.

   Attributes

   name

   address

   phone

   access level

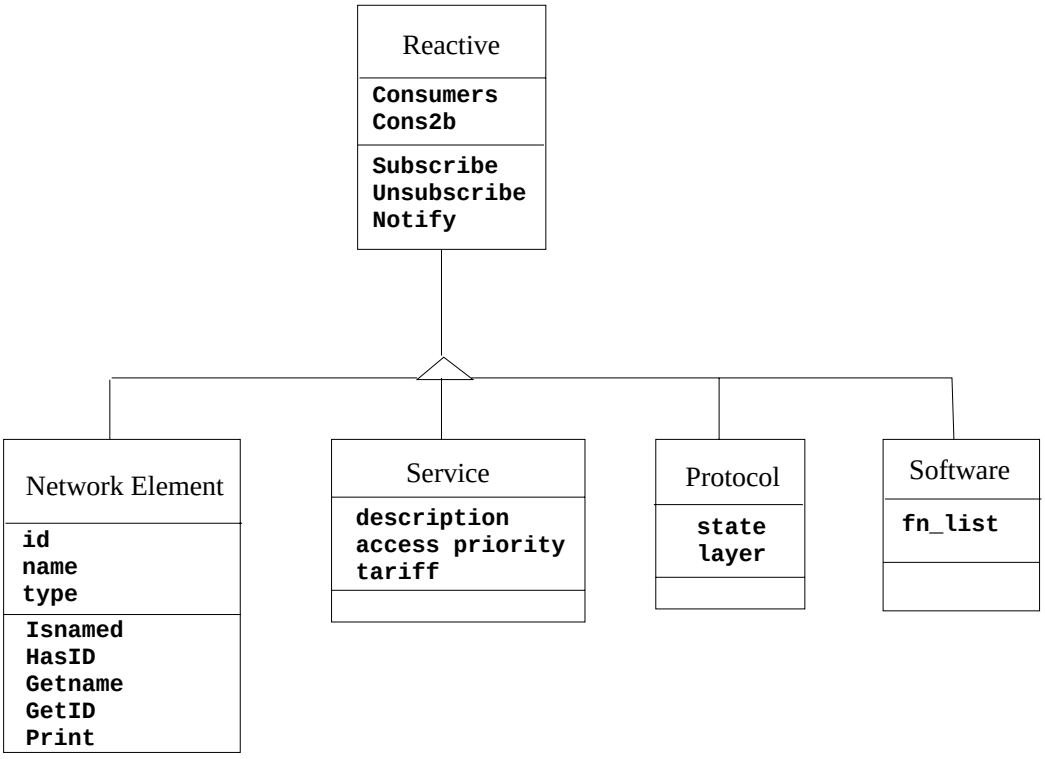

Figure 2.11: Specialization of the Reactive class

Operations

change_name()

change_address()

change_level()

7. **Event**: This class models generic network events. Events related to faults will result in Alarms. Other events could be the creation or deletion of Managed Objects. Each event has a description, source and reason associated with it. It is also important to maintain the time at which the event occured.

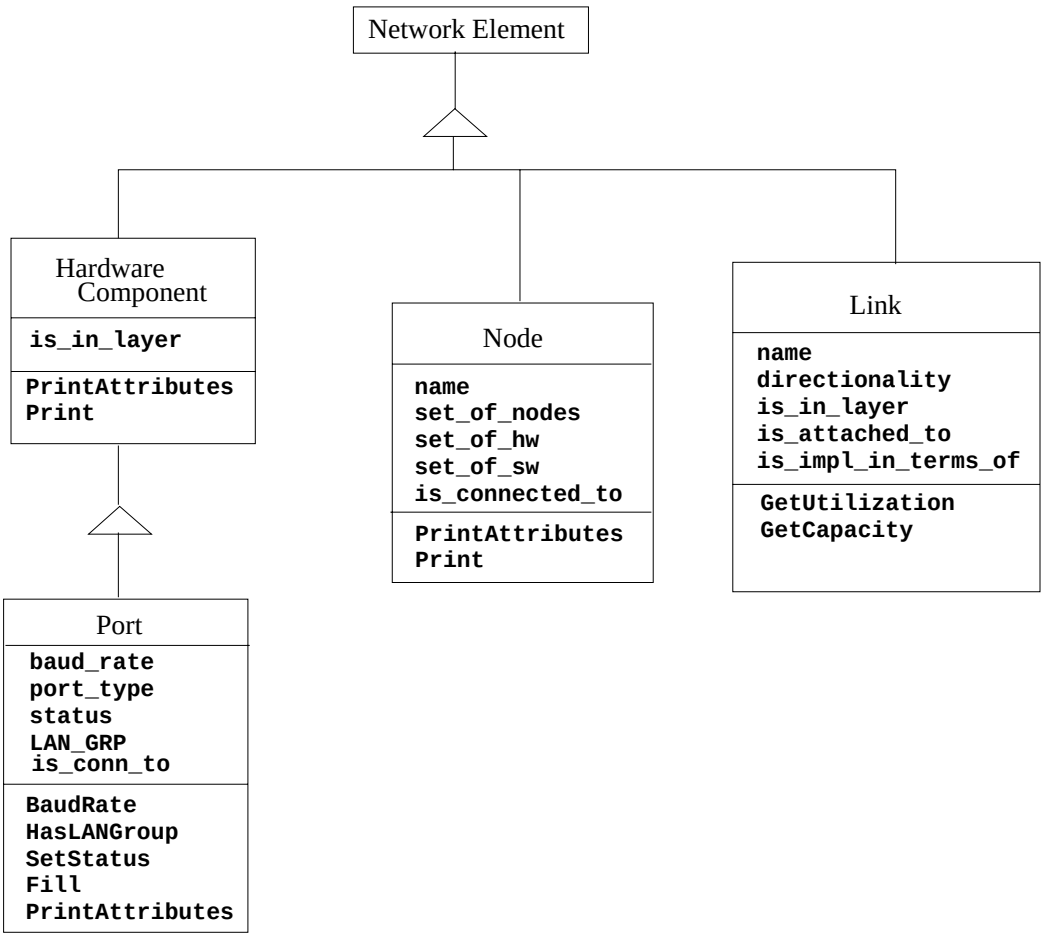

Figure 2.12: Specialization of the Network Element class

Attributes

reason

source

description

time_stamp

8. **Trouble Ticket**: Each time the operator receives a report regarding a network fault, he opens a Trouble Ticket as a record of the fault report.

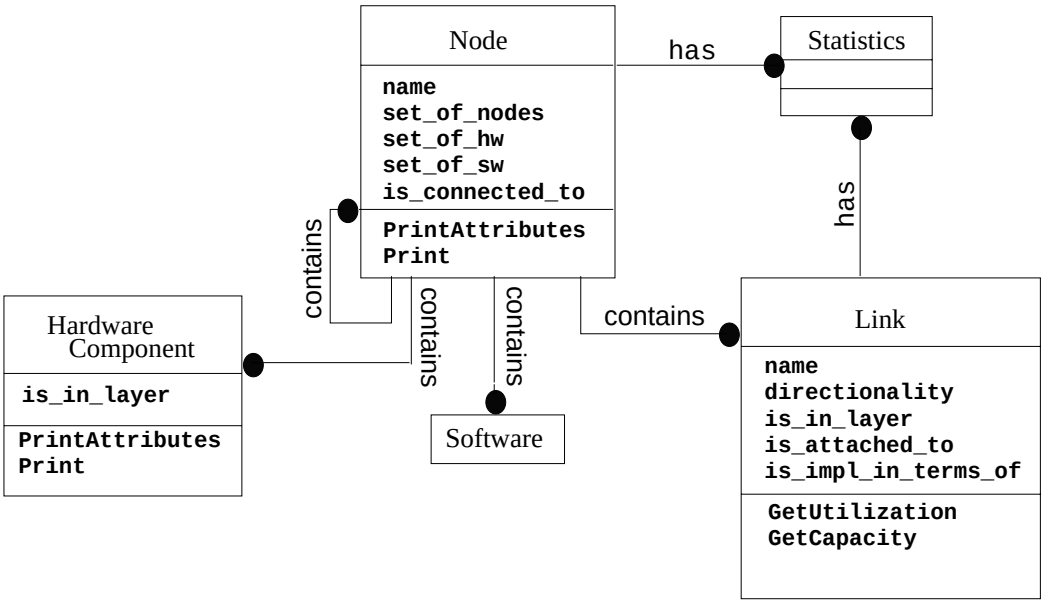

Figure 2.13: Relationship between Nodes, Links, Hardware Components and Statistics

Several alarms may result in a single trouble ticket since they may all refer to the same fault. As the faults are rectified the trouble tickets are closed and stored in a Log.

Attributes

time opened

description

status

priority

probable cause

9. **Notifiable**: This class has been created in order to model Database Events and Rules [5]. These Rules can be dynamically subscribed to or un-

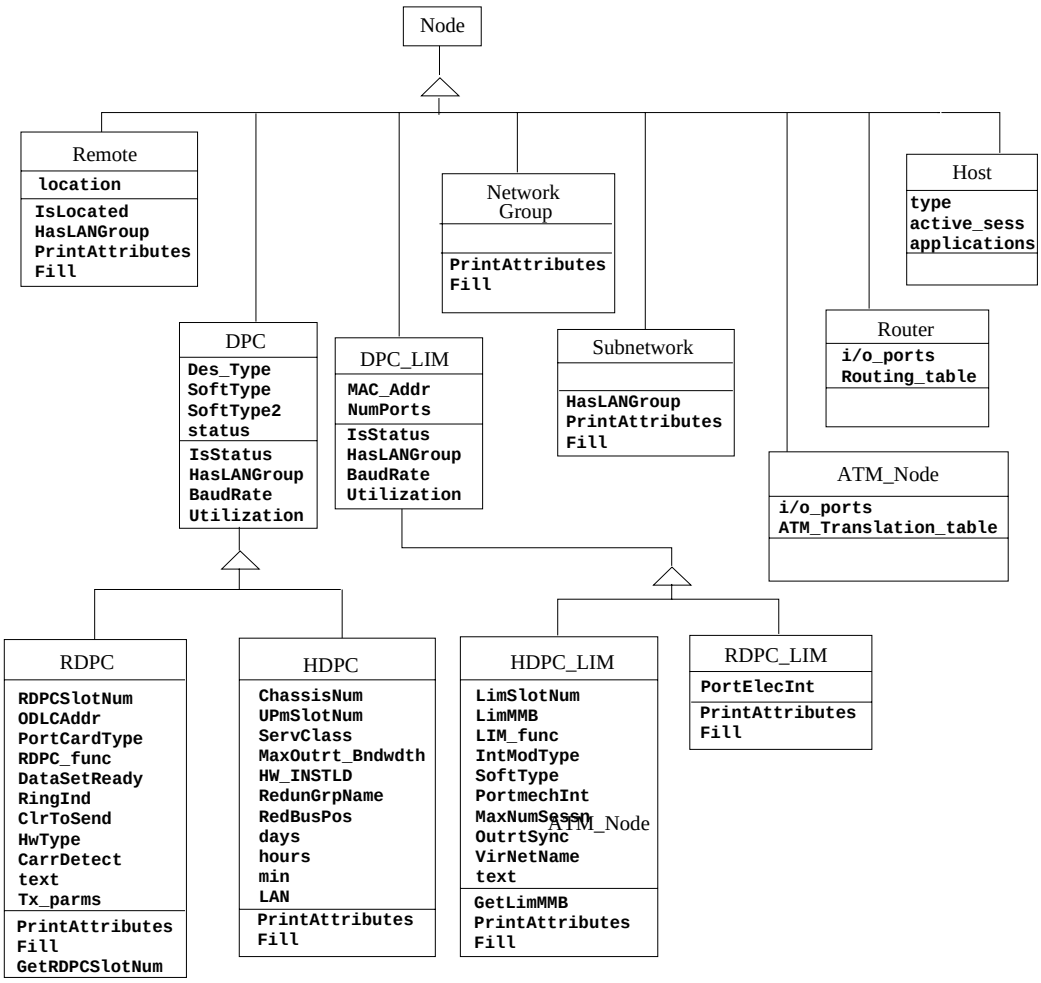

Figure 2.14: Specialization of the Node class

subscribed to as mentioned earlier. The occurrence of database events causes rules to fire. Database events are different from network events described earlier. Rules and Database Events inherit from the Notifiable class.

10. **Support Objects**: The importance of Support Objects was mentioned earlier. These are not part of the network per se, but are very essential for the implementation of the system. This is also an abstract class with no

instances.

11. **Network Element**: This class models a generic network device. It could be a Node, Link or a Hardware Component. It inherits from the Reactive class.

    Attributes

        id

        name

        type

    Operations

        Isnamed()

        HasID()

        Getname()

        GetID()

        Print()

12. **Service**: As mentioned earlier, the various services offered by the service provider need to be modeled. This serves as a generic model for the services offered. Any specific service would then inherit from this class. Certain 'Rules' may apply to services as well. Hence the service class inherits from the Reactive class.

    Attributes

        description

        access level

tariff

set_of_current_customers

Operations

add_customer()

delete_customer()

13. **Protocol**: This serves as a model for a generic protocol. Specific protocols like TCP/IP, X.25, ATM will inherit from this class. Each protocol is associated with some performance statistics. There is a **uses** relationship between 'Session' (which represents a user connection) and a Protocol since every connections uses a specific protocol.

    Attributes

    state

    layer

14. **Hardware Component**: This class inherits from the Network Element class and is used to model simple network devices that aren't made up of other components. Typical examples are an Ethernet card or a Modem card.

    Attributes

    is_in_layer

    Operations

    Print

15. **Node**: The Node class inherits from the Network Element class. It is used to model complex network devices that may contain other Nodes or Hardware Components. Each Node has a **uses** relationship with the Software class. Nodes have a many-many **has_links** relationship with Links.

    Attributes

        name

        set_of_nodes

        set_of_hw

        set_of_sw

        is_connected_to

    Operations

        Print

16. **Link**: This class inherits from the Network Element class and is a model for a generic Link. A Link is a communication path between Nodes. A link may be physical or logical. A Physical link represents a direct path of communication between Nodes, while Logical links themselves can contain other Logical links or Physical links. Links have a **has** relationship with Statistics.

    Attributes

        name

        directionality

is_in_layer

is_attached_to

Operations

getUtilization()

getCapacity()

17. **ATM_Node**: This class inherits from the Node class. This serves as a model for a generic ATM_Switch. Vendor specific ATM_Nodes are created by inheriting from this class. An ATM_Node contains a translation table. The translation table maintains routing information the entries of which are used to switch Virtual Channels (VC) or Virtual Paths (VP) depending on the level at which switching is taking place.

    Attributes

    i/o ports

    translation_table

    switching_level (VC or VP)

18. **Router**: Router inherits from the Node class. A router is a 'Network' layer device in OSI 7 layer model. Each router contains a routing table the entries of which determine the output network interface for each incoming packet. Vendor specific routers inherit from this class.

    Attributes

    set_of_network_interfaces

routing_table

19. **Host**: The Host class inherits from the Node class. It models a generic host. Many different hosts like Workstations, X-Terminals, Mainframes, PC's and Machintoshes are found on today's LANs. These further inherit from the Host class.

    Attributes

    type

    number_of_applications

    number_of_active_sessions

20. **Physical Link**: This is a model for a generic link. It inherits from the Link class. A physical link represents a direct path of communication between two network devices. The medium could be either wired or wireless and the link could be a point-to-point link or a multi-drop link like a bus.

21. **Terrestrial Link**: This class models any generic wired physical link.

22. **Fiber**: This class models a fiber optic physical link.

23. **Coax**: This class models a co-axial cable link. It could be thin coaxial cable or thick co-axial cable.

    Attributes

    impedance

24. **Wireless Link**: This class inherits from Physical Link and models any wireless link. The wireless link could be terrestrial and in different frequency ranges like cellular, PCS, microwave or infrared or they could be Satellite links. Terrestrial wireless links could be indoor or outdoor. Depending on the environment and frequency the propagation characteristics vary.

    Attributes

        frequency_range

25. **Logical Link**: This class inherits from the Link class and is used to model a generic Logical Link. Logical links could contain other Logical Links which will ultimately be implemented in terms of multiple physical links. Hence Logical Links have a many-many **is_implemented_by** relationship with Physical Links.

    Operations

        Collapse()

        Expand()

26. **Virtual Circuit**: This inherits from the Logical Link class. The Virtual Circuit construct was defined for packet switched networks. ATM networks also used this construct sometimes under the name of Virtual Channel. A Virtual Channel is a Logical Link defined between two ATM Switches or user machines.

27. **Virtual Path**: Virtual Paths also inherit from the Logical Link class. A Virtual Path is a construct defined specifically for ATM networks. This was done so that bandwidth in an ATM link could be partitioned between several connections. Bandwidth is allocated to Virtual Paths by the operator. Certain Virtual Paths are reserved for carrying signaling information. An ATM Link could contain several Virtual Path and a Virtual Path could contain several Virtual Channels.

28. **PVC**: A Permanent Virtual Circuit (PVC) is also a Logical Link. It is a permanent connection defined between two ATM switches or end user machines. A PVC is defined by the operator in order to facilitate connection establishment.

2.7 Summary and innovative concepts

In this chapter an Object-Oriented data model for Integrated Network Management is presented. The benefits of using the object oriented framework as a modeling paradigm were presented as the first step. The different aspects of network management were discussed along with their respective functionality. The different requirements on the model were outlined. The notation used to develop the model was described and finally the model was presented and discussed.

The model was developed along OSI specified guidelines. The object classes can be mapped to the GDMO (Guidelines for Definition of Managed Objects) class templates with little effort. The class hierarchy structure isn't too deep, which is an important feature. It is a unified model that covers all aspects of Network Management. In many of the networks that exist today the different

management tasks are handled by different applications that are based on different models. Fault and Configuration management may be handled separately and may be built on different database systems altogether with different models of the network. In our model there is just one MIB on which the model is developed. It is left to the implementor to decide on whether he/she wants a central MIB or a distributed MIB. The model developed is an object-oriented model. However this can be implemented on either a Relational Database platform or an Object-Oriented database platform or on a hybrid system.

The operator only interacts with the model/database. The operator issues the commands to the database and does not have to be concerned with how these get translated to actual actions on the network elements. The differences between different vendor equipment is shielded from the central management system. Certain generic performance related quantities were identified. The different "Agents" in the network or Mediation Devices in a TMN framework map the different counters and gauges to the generic performance quantities and report the mapped values.

The model has support for Graphical Objects. It is very important that the Graphical User Interface and the MIB use the same model of the network. If this is not the case then when an graphical application is developed on top of the MIB, a mapping has to be done between the database model and the user interface model each time a query is executed on the database. If multiple copies of the same application are to operate simultaneously then the state of the different user interfaces need to be made persistent in order that consistency is maintained between the different interfaces. Graphical Objects serve both these purposes. This part of the model was developed with an eye on the implementation.

The operator can define his/her own views of the network and make them persistent in the database. A view may be a particular snapshot of the network. For example, all the links carrying connections of customer B is a view that might interest the operator. Customer B may be a valued customer and the operator might want to ensure that proper service is provided to all customer B connections. Each time the operator access a view, it is first updated and is then made available to the operator. This ensures that the operator always gets the most up to date information.

Structural data which is relatively static is modeled as Configuration objects. Sensor data which is more dynamic is modeled as Statistics objects. There is an association between the Configuration and Statistics objects. However it is left to the implementor to implement these in the most efficient way, so as to maximize the availability of data to answer queries and minimize the query time. Logs are modeled to maintain historical data.

The model provides support for embedding rules. These rules can be fired to perform appropriate actions if certain gauge thresholds are exceeded. This is very useful for automating network management. Rules can also be fired in response to database events. Parameter and other control settings in the network can also be automatically modified without operator intervention.

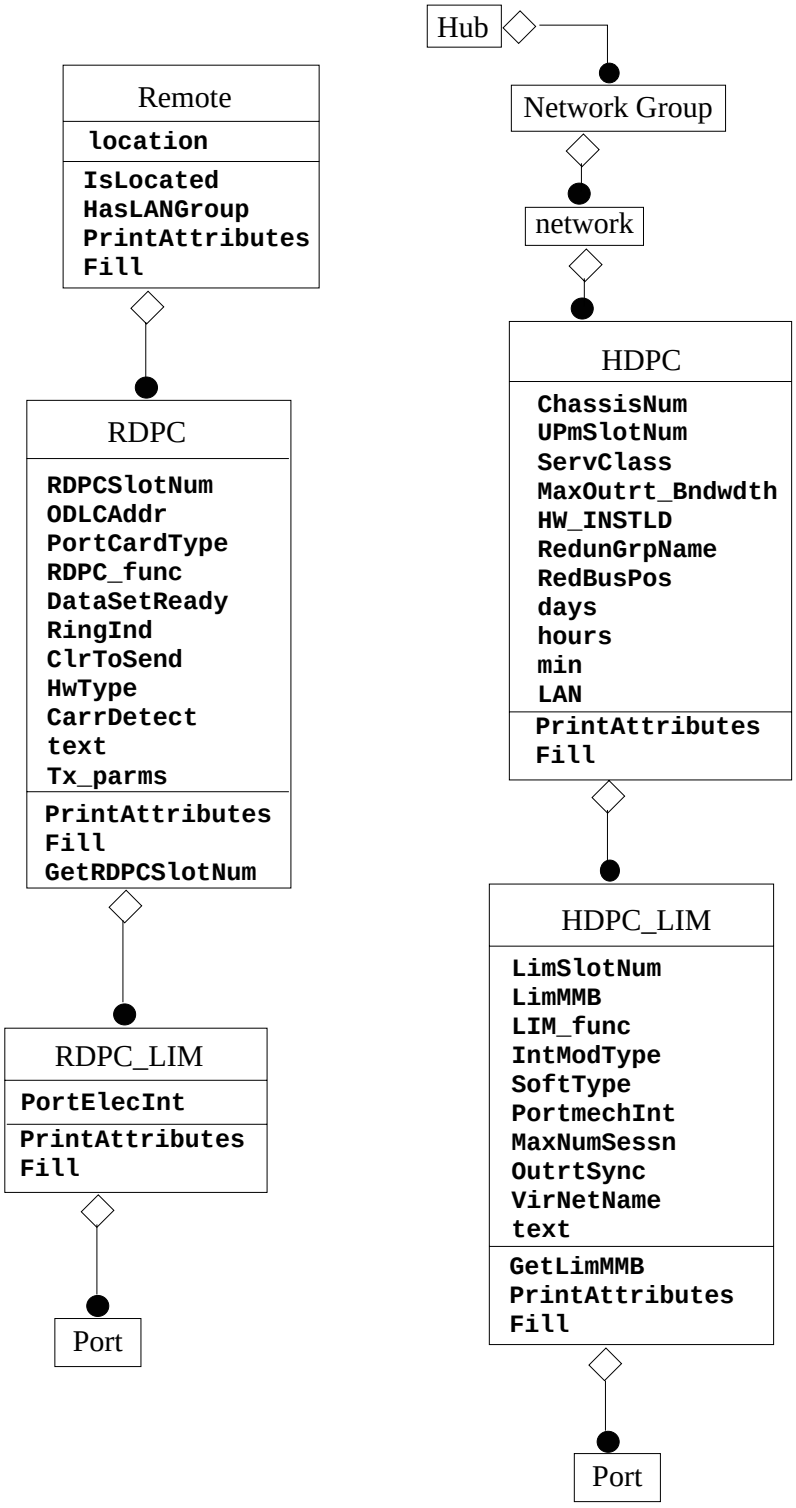

Figure 2.15: part/whole relationship between various Nodes

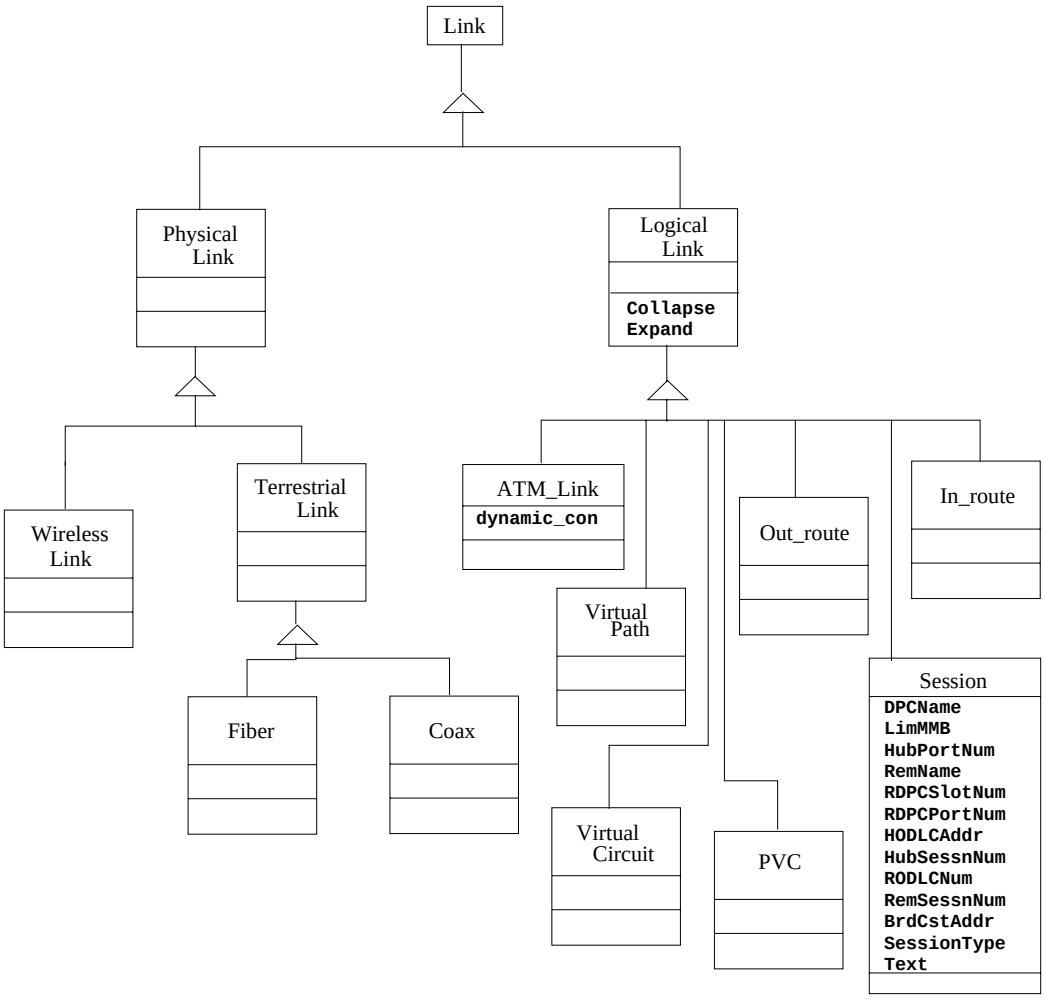

Figure 2.16: Specialization of the Link class

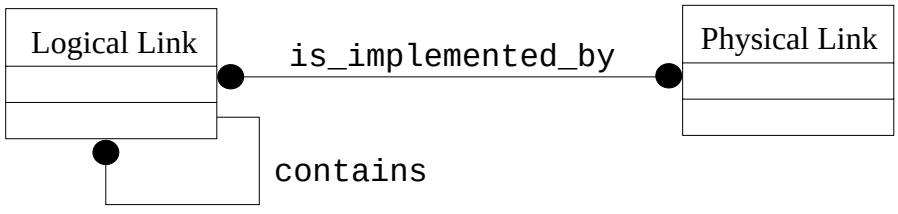

Figure 2.17: Relationship between Logical and Physical Link classes

Chapter 3

Performance Management Models

3.1 Introduction

In the competitive communications market of today it is important to ensure that optimum use of network resources is made. To this end, it is first necessary to periodically monitor the state of network resources. This information is used by network operators to adjust parameters and resource configurations to improve network performance. The functional area of Network Management which is responsible for this, is called Performance Management.

There are three major tasks involved in Performance Management.

- Monitoring: This task involves gathering the performance or state data from all the network elements in the network. Most network elements monitor the amount of incoming traffic, outgoing traffic, traffic lost to errors etc. These numbers are stored as counters. The outgoing traffic counter is incremented each time a packet is successfully transmitted. Example of such counters for ATM are **successfullyTransmittedCells** or **discardedCells**.

- Analysis: This data gathered from the network elements needs to be analyzed in order to determine if the Quality of Service (QoS) requirements of customer connections are being met. It is also possible that adequate service is being provided to existing connections, but the over all service isn't satisfactory because other connections are being blocked. The over all performance may be improved by modifying the resource configurations or allocations.

- Control: Based on the analysis performed on the data from the network, certain parameters or resources are modified to improve network performance. It is possible to have certain automated processes that perform this task. These are the kinds of systems that are envisioned for the future. Most systems today however, require the operator to analyze the data and perform the necessary modifications.

The data collected from the various network elements is also used for long term planning and resource allocation. It is apparent from the above discussion, that there is a need to store both the current values as well as a history for network performance statistics. A database called the Management Information Base (MIB) is used to store this information. In our integrated model developed earlier we have an integrated MIB for all aspects of Network Management. Before the data can be stored a model is required. The intent here is to provide an efficient model for storing performance data, so that this data can be expeditiously retrieved to answer operator queries. The model being developed is for the central MIB.

3.2 Model

The Integrated Network Management model developed earlier, identified a **statistics** class. Statistics are associated with Nodes as well as Links. The statistics class is used to maintain the performance data collected for the network element.

Each node monitors parameters such as **successfullyTransmittedCells**, **discardedCells** and stores them as counter values in a local database. The period between successive updates typically varies between 1 second and a 10 seconds. Every update is not reported to the central MIB. The counter value would be reported to the central MIB at 5 to 15 minute intervals during normal operation. The central management system then has to compute from these counters, information on throughput, utilization, error rate etc. For example, if a Node A reports the bytesTransmitted counter value to be P_1 at time t_1 and P_2 at time t_2, then the throughput is computed as follows:

$$throughput = \frac{P_2 - P_1}{t_2 - t_1}$$

Other quantities may be computed similarly.

There are two modes of data collection:

- element reporting: In this mode the network elements periodically report data to the central network management system.
- polling: In this mode the central management system periodically polls the network elements for performance data. The network elements only send notifications in case of faults or abnormal conditions.

The central management system can choose to store the raw data i.e the counter values or can map these to more meaningful quantities like throughput, error rate etc and store them. The counter values are meaningless to the operator who wants to monitor performance. A typical operator query would be

- show the throughput of a certain link at a particular time.

Hence ultimately the counter values have to be mapped to other quantities. The only issue is whether the mapping should be done at the time the operator query is issued or at the time the data arrives. There is a trade-off involved in this process. If the counter values are themselves stored then no processing is required when the data arrives, however there is processing delay before an operator query can be answered. This delay may be significant if the operator wishes to see for example, the utilizations of all links lying in a particular subnetwork. If the mapping is done prior to storing the data, then a significant amount of processing is required since there are a large number of network elements in the network. Most of this processing may not be necessary since the operator may query only a small fraction of all the values stored. However the operator query is processed much faster in this case. We can tolerate some extra processing before storage, and since our priority is answering the operator query in real time, we choose the latter approach in our model. In our model utilization, delay and error rate are stored as statistics for each network element.

Note: Network Elements from separate vendors and those supporting different protocols have different counters to maintain statistics. Some might store the number of packets transmitted while others may store the number of bytes transmitted. The packets may be fixed sized like in ATM or may vary in size. Hence the mapping of the counters to generic performance quantities will be different

for different network elements. Protocols like SNMP have only the capability of reporting the counter values, but with the more complex TMN framework, it may be possible to have the network elements themselves report throughput or error rate. This will shield the differences between the different network elements from the central Network Management System.

There are two kinds of user queries:

- Queries on single objects: These queries need to process data only from a single object. Typical queries are:

  1. utilization of a Link A at time t_i
  2. average throughput and error rate for Link B in the time interval t_j to t_k
  3. buffer capacity of Node L at time t_i

- Queries across objects: These queries are more complex and involve attributes of several objects. A typical query would be:

  1. error rates on all Links lying in SubNetwork K and connected to Nodes with loads $> 80\%$

Both the above kind of queries can refer to either a

- snapshot: or view of the network at a specific time instant
- time interval: some statistic of a performance quantity over a time interval.

For snapshot queries the value for the time instant requested may not be available. Interpolation may be used to obtain the required value or the value at the closest time instant to the one requested, may be reported. For all user queries we would like to satisfy the following conditions:

- Coherency: Values reported as a result of any query should have been recorded at approximately the same time instant. For example, if the user requests the utilization of all links connected to Node A at time t_1, it wouldn't be meaningful if we reported the utilization for Link 1 at t_1-50 and Link 2 at t_1. Some interpolation would be necessary in this case.

- Recency: It is important to report values recorded as close in time as possible to the one requested by the user.

Note: It is worth noting that there are two different time instants that can be used as time stamps. One is the time at which the value was recorded in the network. The other is the time at which the values is actually stored in the database. The time at which the value was actually recorded is more important and is used as the time stamp.

The operator may wish to view the performance data recorded prior to a fault or a network performance degradation. This might give the operator some insight into the nature of the fault and can also be helpful in preempting future performance bottlenecks. Performance data is also useful for long term planning purposes and to predict the future demand for resources. However, as the data gets older it gets less valuable. Hence from a storage standpoint it would be useful to aggregate some of the older data and store fewer values. This could also help improve the query performance since the data sets on which the queries have to be executed will be smaller.

In our model, we propose to use 3 different levels of precision or granularity. Updates arrive from the network typically every 5 minutes for each network element. This will constitute the highest level of precision or the finest granularity. The most recent data is stored at this precision in a high precision list. Slightly

"older" data is aggregated and stored in a medium precision list. Even "older" data is aggregated further and stored in a low precision list. The concept of "old" and "older" data is ambiguous since there is no quantitative method of determining this. The experience of network operators and the kind of information they require for managing network performance gives some insight into what we may define as "old" and "older". This is a method of determining how much data needs to be stored at each level of precision.

The next question that arises is, how much data should be aggregated at each level. One method of determining this, is by computing the correlation between the recorded data values. All values having a correlation above a certain fixed threshold can be aggregated into 1 value. The disadvantage of this approach is that additional processing is required to determine the correlation. Depending on the data we have we may conclude that 10 values can be aggregated to 1. Alternatively, for very uniform data 20 values could be aggregated into 1 or for non-uniform data only 5. The additional processing overhead of determining the correlations may not be acceptable. In our model we assume that at the highest precision, a data point is recorded at five minute intervals. The medium precision list has data for every 1 hour and the low precision list has data for every 24 hours. These numbers are parameters that the operator should be able to specify, so we provide a mechanism to allow the operator to specify these numbers. The default values are those that are specified above.

When the high precision data is aggregated or compressed, N values recorded at K minute intervals are taken from the high precision list. Some statistics for these values are computed. The maximum, minimum and average are examples of statistics used. The time stamp in the high precision list is a single number.

This gets converted into a time range with start and end times. The statistics computed from these N values are inserted into the medium precision list along with a time range and those N values are dropped from the high precision list. Hence N values are aggregated into 1 entry. The default values for N and K in the high precision to medium precision case are N = 12 and K = 5. A similar procedure is carried out to aggregate values from the medium precision list and put them in the low precision list. The maximum to average ratio does give a measure of the variation of data. However we may choose to compute the variance also in addition to the maximum, minimum and average, while aggregating data.

Note: This will be implemented as three different processes. Each process will handle inserting values into 1 list. This will involve accessing values from higher precision lists, aggregating them and storing them in the next lower precision list.

The model proposed above is a simplistic model where N values get compressed to 1. Network traffic variation has patterns depending on the time of day. There are certain peak hours of network usage during the working hours. A recent study at GTE Laboratories showed that, cellular phone usage varies significantly depending on the time of day and reaches its peak during the 5pm to 7pm period. Hence a more complex model to aggregate data is more appropriate. This means that the value of N varies depending on the time of day.

We would like our system to operate in real time. However the central MIB may be overwhelmed by the large number of updates from a large network. To alleviate this problem, we might consider storing the current updates in memory and doing a block of updates to the database rather than doing an update each

time a value is reported. This is a block of appends and not updates. A database append should be more efficient than an update since an append only adds new values and does not modify existing values. In case of a system crash the values that are lost can be recovered from the respective network elements themselves.

Another option to reduce the amount of data stored is not to record a value in the database unless the new value differs from the previous value by more than p%. Interpolation is used to answer an operator's query for time instants that are not recorded in the database.

A summary of the issues and alternatives discussed, is shown in Table 3.1.

The data model for Configuration management already contains a model for Network Elements. The statistics are related to the Network Elements. Since the statistics can be potentially very large it would not be practical to store the performance information with the configuration information (structural object) which is more static. Some form of concurrency control would be required every time the statistics are accessed and if they are stored along with the structural objects then the structural objects will be unnecessarily locked and may not be accessible to answer queries. The two should be stored separately but a link (or pointer) should be maintained between the two in order to associate the structural object with its statistics object.

We propose two different models for storing the statistics:

1. In this model there is one object for each performance parameter. In our case we have 3 different paramters i.e Utilization, Error Rate and Delay. Each object contains the data for all the network elements and is called a "View" object. For example the Utilization_Stat view object contains a list of Individual_Utilization structures as shown in figure 3.1. The Rum-

Issues/Alternatives	**Advantages**	**Disadvantages**
Map counter values to utilization, error rate etc., prior to storing in the database	Operator queries can be answered with smaller delay, which is extremely important	Requires significant processing prior to storing data and hence may not be able to operate in real time
Store counter values in the database and compute utilization, error rate etc., when the operator query is issued	No processing is required prior to storing the data	There may be a significant processing delay when the operator query is issued
Keep a few sensor updates in memory and do a block of appends to the database	The system is not overwhelmed by the large amounts of data pouring in	Memory requirement may become very large. Also in case of a system crash large amounts of data may have to be retrieved from the network elements
Do not store an update in the database unless a performance parameter value changes by more than p%	Fewer updates required to the database reducing the overhead	

Table 3.1: Summary of issues and alternatives

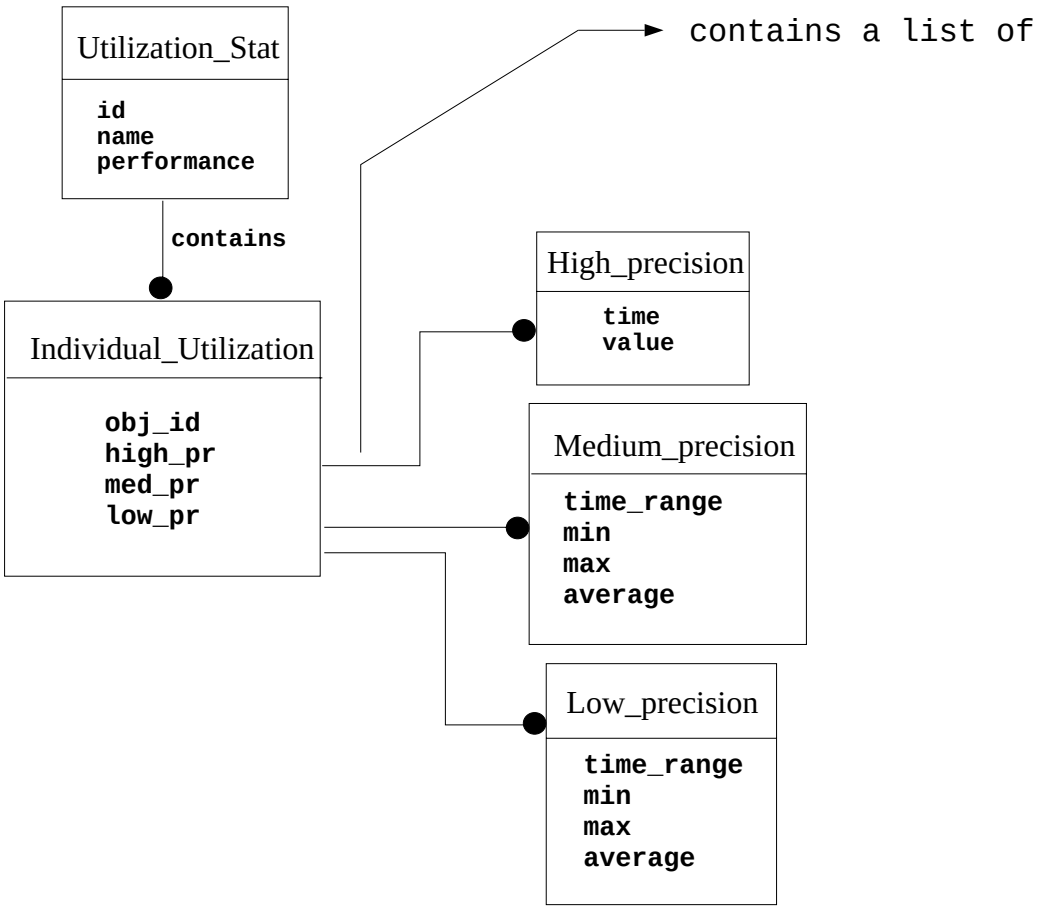

Figure 3.1: View object performance model

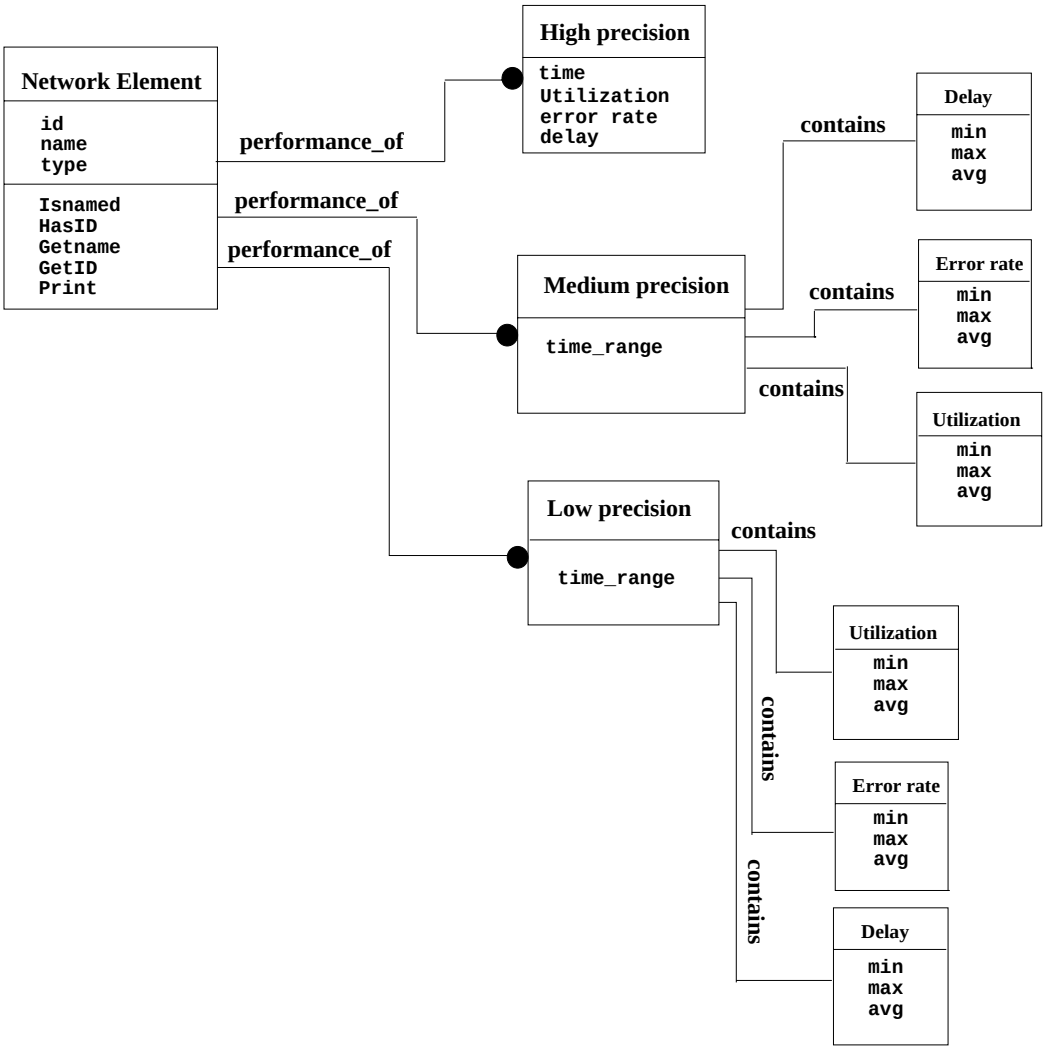

Figure 3.2: Integrated performance model

baugh notation described earlier is used to depict this. Each instance of the Individual_Utilization structure refers to one Network Element. Each instance of the structure contains a network element identification number (or pointer) and 3 different lists to maintain the data at the 3 levels of precision. The high precision list contains a list of $< timestamp, value >$ pairs. The medium precision and low precision lists contain a list of $< timerange, maximum, minimum, average >$ 4 tuples.

The motivation behind having such a structure is to be able to efficiently answer queries across objects. Such queries, as described earlier, need to access data from multiple network elements. Hence if the data for all the network elements is clustered together then the query performance should improve. This is much like the present system where there is a loose coupling between the performance data and the corresponding network elements. The only way to associate network elements and their corresponding performance statistics is by the Network Element id.

This structure however comes with a price. The network elements send their updates as a block reporting all the performance data together. With the present structure the block received from each network element has to be parsed and we require as many updates as there are view objects, for each network element. In our model we would require updates to 3 objects per network element. This is quite a significant overhead and may prevent the updates from being processed in real time. If we wanted to add another statistic or view object then a separate list would have to be created and the number of updates would increase. This object can become potentially very large and it will slow down the query performance. To

alleviate this problem we could split each performance object, based on a certain grouping for the NEs. All NEs in a subnetwork or administrative domain can be grouped together.

2. In this model all the performance statistics associated with a network element are stored together. The structure for this, in the Rumbaugh notation is shown in figure 3.2. Each network element has pointers to 3 lists corresponding to the 3 levels of precision. The high precision list contains a time stamp along with all the performance data. The medium precision and low precision lists contain a time range and the minimum, maximum and average values of all performance data for that time range.

   All the performance parameters of a network element are reported together and since they are all being stored together, only a single list needs to be updated each time. Hence this will be significantly faster than the previous model. If an operator issues a query across objects then several lists have to be visited, one for each object involved in the query. This might require many disk accesses and might perform worse than the previous model. In cases of faults or performance bottlenecks the operator might want to see all the performance data related to a particular network element. This model performs better than the previous model for such queries. This model is also well suited for distributed implementation. All the performance data for a particular network element can be migrated to another server if necessary.

   The performance of this model can be improved by maintaining an extra object. This object is like the view object of the first model and maintains

information on the statistics of all network elements. However instead of storing entire statistics objects we just stores pointers to the high precision, medium precision and low precision lists of each network element. Since we are maintaing just pointers the overhead isn't very much.

Another improvement that can be made is to cluster the statistics objects of those network elements that have a high probability of being queried together. If we assume that geographical proximity plays the greatest role in network elements being queried together, then we can use that as a basis to cluster the statistics objects of different network elements.

So to determine the relative performance of these models we need to model a cost for executing each type of query on both the systems. We also need to know what percentage of queries belong to each type. The task of modeling a cost for queries across objects isn't an easy one since the cost would be a function of how many disk accesses are required. The number of disk accesses depends on the clustering algorithm used and also the size of the objects themselves.

3.3 Summary and innovative concepts

In this chapter we have tried to develop a performance management model, which is integrated with the configuration management model. This performance data model is meant to capture sensor data that is reported from the network. Every network element periodically reports its state along with other performance parameters to the central management system. A record of this must be maintained in the central MIB and is used for post-mortem analysis of faults and for long term resource planning.

Most network management systems of today monitor the state of the network but don't use this data. One of the reasons for this, is that no model exists to store this data. Quite often the data is stored in flat files. Unless the performance data is stored in the same database along with the structural data the network operator will not be able to associate statistics with a particular network element.

As information gets older it becomes less valuable. Hence we need not store all of it. Older data should be aggregated and only certain statistics computed from the data are stored instead of the entire data. Examples of statistics that can be stored are minimum, maximum, mean, etc. We propose to have 3 levels of precision to store performance data. In other words, all of the "recently" reported data is stored as it is. "Older" data is aggregated over a short interval and "even older" data is aggregated over a much longer interval. This leads to a tremendous saving in memory. A simple calculation can illustrate this fact. Consider 5 values being reported for each network element every 5 minutes. If each value occupies 5 bytes, and we stored all the data for one week, then we would require about 170K bytes for each network element. Now consider our model where we store all the data only for the last 3 hours. This requires around 3K bytes. The data for the remaining hours in the day is aggregated over 1 hour intervals. This means we store only 20 bytes of information for each hour for the previous 21 hours, requiring a total of 420 bytes. The data for the remaining 6 days of the week is aggregated over 12 hour intervals which results in only 12 values or 240 bytes. Hence with our model we require a total of 4K bytes compared to 170K bytes if all the data is stored. For a network with 1000 network elements this results in a saving of about 165M bytes, for statistics collected over a week. Not only is there a saving in memory, the query

performance also improves since fewer blocks of data have to be transferred between main memory and the disk.

This is one of the first efforts made in developing an integrated performance data model. We propose two different models for storing the data. Different models may be better for different types of queries. One model creates an object for each parameter monitored and contains the data for all the network elements. On the other hand, the other model has all the parameters for each network element in an object and there is a tight coupling between this object and the network element. Both models have the 3 levels of granularity. Certain more complex models such as varying the amount of aggregation based on patterns in the data have also been discussed.

Chapter 4

Database Integration Methodology and Implementation

4.1 Introduction

Database systems have been in operation since the 1960's. A database is a repository of data which is managed by a Database Management System (DBMS). The data in a database is logically organized according to a data model. The earliest databases were developed using the Hierarchical and Network data models. Further progress in database technology led to the development of the Relational model, which is used by a large number of databases today e.g Ingres, Oracle, Sybase, Informix. Developments in database technology in the last decade have led to the Object-oriented data model and Object Oriented Database Management Systems (OODBMS).

It is clear from the above discussion that there exist many database systems with diverse data models. Even the databases based on the same model are implemented quite differently and may have slight differences in their query languages.

Most applications developed so far operate over a single DBMS, and all the database systems from the various vendors were developed with such applications in mind. With advances in networking technology there is an increasing interest in distributed database systems. In a distributed database the management system is distributed and the data is split over many physical locations but the entire system is still supplied by the same vendor and there is no heterogeneity.

Till recently there was no need to address the issue of interoperability between heterogeneous database systems since all systems operated in isolation. It is necessary today to interconnect some applications that have been built over different database systems. Current and future needs dictate that it should be possible to develop complex applications that access data transparently and efficiently from different diverse databases. The naive solution to this problem is to convert and transfer data from one database to another. It is easy to see that this solution is not scalable since a "bridge" must be built between every pair of existing systems and adding a new system would require a bridge between the new system and every existing system.

This problem is very relevant to Network Management as well. Consider the example of one company that has developed a Configuration Management System for its network using an Object oriented database system. A Fault Management system was developed independently on another platform. The company would now like to integrate these systems into an Integrated Network Management System. This would require interoperability between the two applications and consequently the two databases. It is evident that the problem of interoperability has to be addressed since it isn't practical to transfer the tremendous amount of network management data from one database to another.

No generalized model exists to achieve the interoperability between heterogeneous databases. Benefits of integration:

- It will permit data sharing in a transparent manner.
- It will facilitate integration of existing applications without the need for data replication.
- It will shield heterogeneity and promote Systems Integration in general.

4.2 Architecture

This section describes an architecture for database integration. The architecture is as shown in figure 4.1

Any application should be able to access data from multiple databases in a transparent manner. The databases themselves could be present on diverse hardware platforms with different operating systems and supporting different data models. Transparency has several facets:

- The application should be able to access data without knowledge of the physical locations of the data.
- The data should be uniquely identifiable across various systems, by using a common nomenclature.
- The access should be platform, operating system and data model independent and the details of these should be shielded from the application.

The architecture is based on the client-server paradigm. This is the same paradigm on which the multi-user single database systems are based. The client

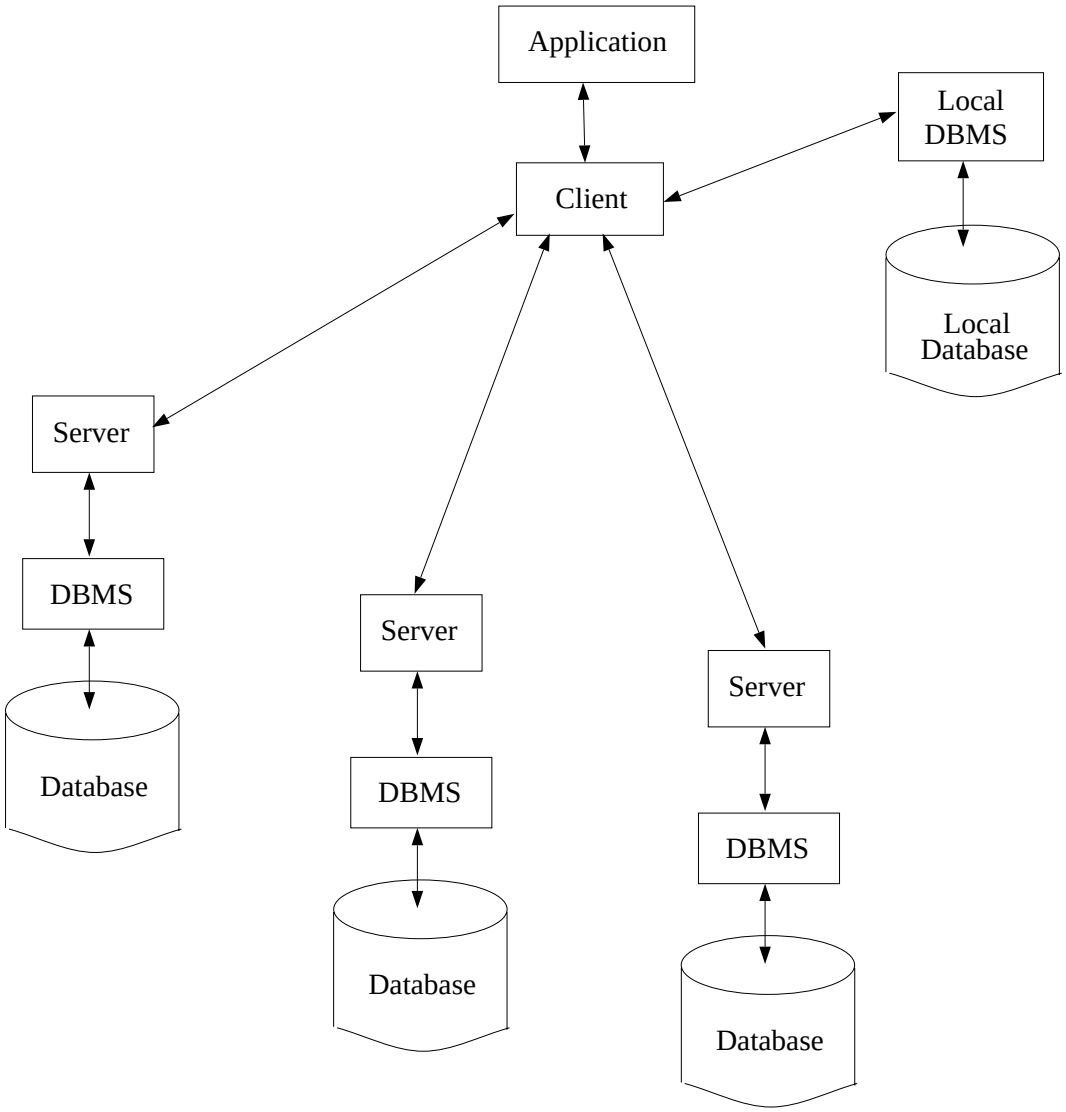

Figure 4.1: Database Integration Architecture

in the multi-database architecture is much more powerful and is capable of interacting with multiple databases. The client consists of three modules as shown in figure 4.2.

- Query Parser module: This module receives the queries and commands issued by the application. The command is parsed and the location of the operands is determined. The operands could be local or remote. The appropriate command is then sent to the command module.

- Command module: This module receives the command to be executed along with the location of execution. It is then responsible for sending the commands to the appropriate servers and receiving the results from them.

- Result module: Any results returned on execution of the command are sent to this module by the Command module. This module converts the results to the required format and passes them to the requesting application.

Note: Any reference to a client or server refers to our client or server and does not refer to any part of the DBMS.

4.3 Approach

Before addressing the more general issue of integration across heterogeneous hardware platforms and operating systems we have tried to develop a testbed for integrating Relational database systems on a UNIX platform, based on architectures proposed in [8, 10]. The same approach can be extended to solve the more general problem. External Data Representation (XDR) can be used for a common data representation format.

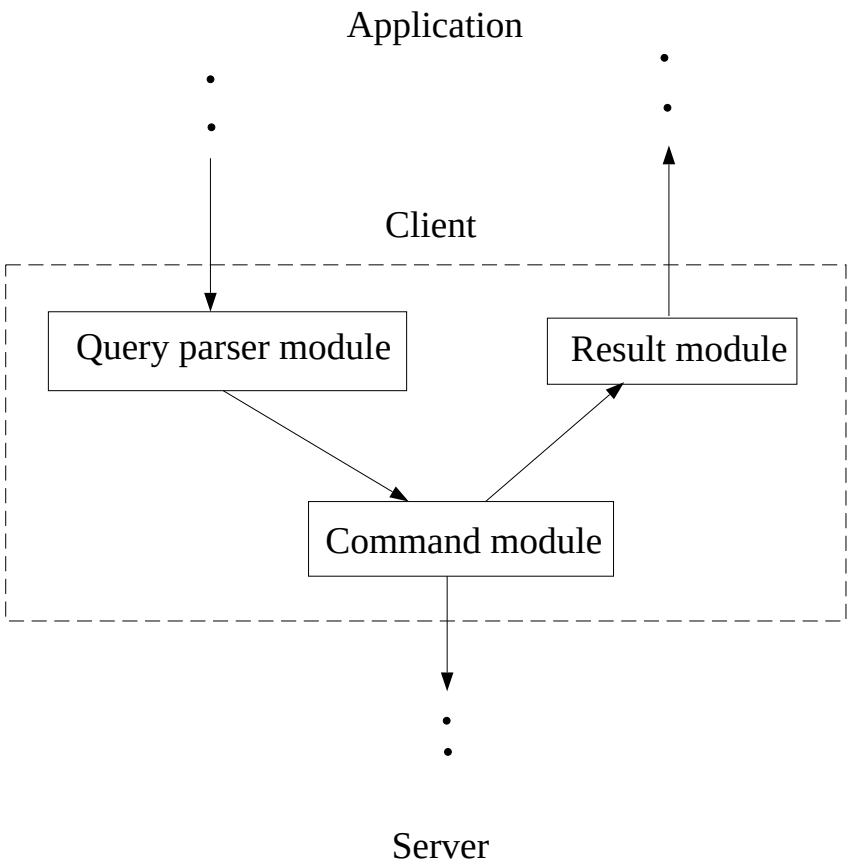

Figure 4.2: Client modules

As part of earlier work, ADMS a relational database system was integrated with Ingres and Oracle [17]. This is being extended to integrate ADMS with Illustra. Illustra is an Object-Relational database system with an enhanced SQL (Structured Query Language) query language. To start with only the relational capabilities of Illustra and being integrated.

ADMS is the client system which connects to the other servers. The same client can interact with all the servers. The ADMS client already exists so only a server module needs to be developed on the Illustra side.

4.3.1 Features

1. The client maintains a list of all Shared Base Relations (SBR) along with their locations as part of the shared database catalog. An SBR is essentially a relational table. However some additional information is maintained for each SBR, since it has to be shared between the client and the servers.

2. Each transaction is logged in a log file along with a time stamp. This information is useful in case of a crash so that the appropriate rollback can be performed.

3. The server maintains a time stamp value for each tuple in every SBR.

4. Every server maintains some information tables in addition to the application tables.

    - SBRS: This table maintains information regarding the SBRs at the server. Each row corresponds to one SBR. The name of the SBR, the last time of modification and the current number of views of this SBR are the fields of each row.

- INS_SBRS: This table maintains information on all the SBRs created at the server. The name of the SBR and the time of creation is the information maintained.
- DEL_SBRS: This table stores information on the SBRs deleted from the server. The name of the SBR and the time of deletion are maintained.
- BINDS: This table stores bindings for selections and joins. A binding is a view of one table or a combination of tables. A client typically may store one or more views of a table in its local database.

5. Whenever a remote SBR is queried by the application, the client creates a view of the SBR based on the query and stores the view in its local database. If the view already exists then it is updated, and the query processed. Each subsequent time the application sends the same query, the view created by the query is updated from the server and the query processed. A binding is created for the query at the server. The view for the binding is updated by the servers by sending the tuples that have been added to the SBR since the last update.

6. There are two ways in which queries can be processed.

   - The entire table can be down loaded from the server and the client could process the query locally.
   - The client can send the query over to the server which processes the query and returns the results.

   We use the latter approach. By using this approach the processing gets

distributed to the servers instead of the being done only at the client. Down loading entire tables over the network can be an expensive operation, since tables can be very large. Quite often the queries relates to a small fraction of tuples in the entire table.

The client caches the results of the query from the server in a view. Hence for subsequent queries, the server only needs to process the query on the tuples that have been added since the last query. The server also informs the client of any tuples that may have been deleted since the last query.

7. Even though all the servers have SQL as the query language, the exact syntax may vary. For example, the SQL **copy** command that copies a file into a table, requires a format specification in Ingres but does not require the format specification in Illustra. When the more generalized problem of integration is addressed the query language also may not be the same for the different servers. Hence the actual command itself isn't sent from the client to the server. Instead the client sends a code which the server interprets to be a certain operation. The appropriate arguments are sent depending on the operation. For example for the **select** operation the selection condition needs to be sent. This ensures that the details of the server query language are shielded from the client. Hence the addition of another server does not require any changes on the client side.

4.4 Implementation

The server and client are started as two separate processes. The server is created on the machine on which the DBMS it represents, is present. The client can be

present on any machine. As only the server was implemented as part of this work only the server modules are discussed in detail. The various server modules are described below.

4.4.1 server

This is the main server module. The user is prompted for a login name. Once the login name is verified the server is started. The name of the database is passed as an argument to this module. A catalog file is read to verify that the database the user wants to access is indeed a "shared" database i.e it contains Shared Base Relations (SBR). If the database isn't a shared database the program exits with an error message.

The next step is to complete all initialization necessary to accept a TCP/IP socket request from the client. Each server has a specific TCP port associated with it. The client knows the TCP port for each server. A TCP stream socket is set up using the 'socket' system call. The local address is bound to a specific value using the 'bind' system call. If the address is successfully bound, the server issues the 'listen' system call to wait for client connections. Any request coming in on the socket is accepted using the 'accept' system call. The server is a concurrent server i.e it can accept new connections while there are existing connections with other clients. Hence as soon as a client connection is accepted the server spawns another process. The new process handles the new client connection while the existing process waits for other connections.

Certain ADMS specific initializations are necessary for all servers. Once the initializations are completed a connection is set up with the Illustra DBMS for the database specified by the user using a call to the Illustra function 'mi_open'.

This call returns a pointer to an Illustra connection, which must be passed as an argument to any command issued to Illustra. The server is now ready to process any client commands. As mentioned earlier the client sends a code to refer to a particular command. In this implementation the code is simply a 'C' enumerated data type. The different codes available are:

- IS_CRE_SBR_NUM: This command creates a new Shared Base Relation (SBR) by calling the 'is_creSBR' module.
- IS_DRO_SBR_NUM: This command deletes an existing SBR by calling the 'is_droSBR' module.
- IS_INS_SBR_NUM: This command inserts tuples into an existing SBR by calling the 'is_insSBR' module.
- IS_DEL_SBR_NUM: This command deletes tuples satisfying a particular condition, from an existing SBR by calling the 'is_delSBR' module.
- IS_ALT_SBR_NUM: This command is used to alter tuples in an SBR by calling the 'is_altSBR' module.
- IS_CRE_SEL_NUM: This command selects tuples satisfying a given condition from an SBR, by calling the 'is_creSEL' module.
- IS_UPD_SEL_NUM: Whenever the client selects tuples from an SBR, it creates a view in its local database corresponding to that selection. This command updates that client view of the SBR by sending only those tuples that have been added since the last update by calling the 'is_updSEL' module.

- IS_DRO_SEL_NUM: This command drops a selection binding from the BINDS table by calling the 'is_droSEL' module.
- IS_CRE_JOI_NUM: This command computes a join between two SBRs by calling the 'is_creJOI' module.
- IS_UPD_JOI_NUM: This command updates the client view (join binding) by calling the 'is_updJOI' module.
- IS_DRO_JOI_NUM: This command drops a join binding from the server BINDS table by calling the 'is_droJOI' module.
- IS_UPD_CAT_NUM: This command updates the client catalog by calling the 'is_updCAT' module.
- IS_CRE_IND_NUM: This command creates an index on an SBR by calling the 'is_creIND' module.
- IS_RMV_IND_NUM: This command deletes an index created on an SBR by calling the 'is_rmvIND' module.
- IS_CRE_SSJOI_NUM: This command is used to create a join between tables on two different servers. This is handled by the 'is_creSSJOI' module.
- IS_UPD_SSJOI_NUM: This command updates the server-server join binding by calling the 'is_updSSJOI' module.
- IS_DRO_SSJOI_NUM: This command drops the server-server join binding from the BINDS table by calling the 'is_droSSJOI' module.
- IS_CRE_SCJOI_NUM: This command is used to create a join between a client table and a server table. This is handled by the 'is_creSSJOI' module.

- IS_UPD_SCJOI_NUM: This command updates the server-server join binding by calling the 'is_updSCJOI' module.
- IS_DRO_SCJOI_NUM: This command drops the server-server join binding from the BINDS table by calling the 'is_droSCJOI' module.

4.4.2 is_ modules

Each client command is handled by a module whose name starts with 'is_'. These modules are responsible for reading all the parameters necessary for the command, from the socket. Each 'is_' module then calls another module to handle all the interaction with the database. Hence the 'is_' modules have no database specific code and can be used for all the servers. These modules were developed earlier for the Ingres and Oracle servers and the same ones are used for the Illustra server as well. The name of the module called by an 'is_' module is constructed as follows: the suffix remains the same and 'is_' is replaced by 'rs_'. For example is_creSBR calls rs_creSBR, is_creSEL calls rs_creSEL.

Examples of parameters read in from the socket are:

- SBR name
- Transaction number
- Client host name
- Select condition for the 'select' command
- Join condition for the 'join' command

Note In each 'rs_' module a transaction number is one of the arguments. This is used to log all the operations performed in the module in a log file.

4.4.3 sql command

This module is the dynamic SQL command handler. This module takes an SQL command, an Illustra connection, a field delimiter and a socket as arguments. This module handles the task of executing an SQL command and sending the results of the command back to the client on the socket.

DBMSs like Ingres provide the 'EXEC SQL' statement which can be used to issue SQL commands from a 'C' program and get the results directly into program variables. Since Illustra is an Object-Relational system and supports richer data types it isn't easy to provide such a mechanism. The only way to execute an SQL command from a 'C' program is to use the Illustra library function 'mi_exec', which takes an Illustra connection and an SQL command as arguments and sends the command to the server to be executed. However it does not return the results of the command.

To get the results, the Illustra function 'mi_get_results' needs to be called. If row values are available from the database then another module called 'output_tuples' is called to read the results and write them to the socket so that they can be read by the client. The 'output_tuples' module calls the Illustra function 'mi_get_row_desc_without_row' to get a description of what is contained in a row. The return value of the above function is used to determine the number of columns present in a row. A row pointer is obtained by calling the Illustra function 'mi_next_row'. This function is called repeatedly until there are no more rows to fetch. To get the actual values of the columns of any row the function 'mi_value' is called. The columns are fetched in a loop. Illustra only returns values as character strings. It is left to the user to convert these strings to actual data types. The tuples are written to a buffer with each column value being

delimited by the field delimiter. This buffer is then written to the socket.

It is necessary to issue certain local SQL commands as well. The results of these commands are required for local processing and don't have to be sent to the client. The same procedure described above is used to process local SQL statements. However instead of writing tuples out on a socket the results are returned in a result buffer. For our application the local SQL statements request only one or two values from a row in a table hence the above mechanism will suffice. Rows of tuples for which a cursor is required are never requested. It may be necessary at a later stage to improve on this mechanism.

4.4.4 rs_creSBR

A transaction number, the client host name, the SBR name and a socket descriptor are passed as arguments to this module. This module is responsible for creating a new SBR in the Illustra database. The following operations are performed in this module.

- An SBR is a shared table. Every table must have a schema according to which it is created. A schema is just a list of fields with their corresponding data types. The client sends the schema file over the socket.

- The function 'tcp_read_file' is called to read the file from the socket and store it in the database directory under the same name as the SBR.

- The list of field names with their types must be read in from the schema file so that the SBR can be created. The function 'readsch_for_illustra' is called to read the schema file. This function returns in a buffer the field

names followed by their data type in a format that can be understood by Illustra.

- Another field, the time stamp, an integer, is prepended to the schema and an insertions table is created in accordance with the schema. Corresponding to every SBR there are two tables created. One is the insertions table and the other is the deletions table. The insertions table name is obtained by prepending 'ins_' to the SBR name and it contains all the tuples currently in the SBR.

- The deletions table name is obtained by prepending 'del_' to the SBR name and it contains information on the tuples dropped from the SBR. It has two fields. One is a time stamp and the other is the condition based on which tuples were dropped from the SBR. This table is also created.

- A tuple is added in the SBRS table corresponding to this SBR. Recall that the SBRS is a table containing information on all the Shared Base Relations existing at the server.

- A tuple is added in the INS_SBRS table also. Recall that the INS_SBRS is a table containing information on all Shared Base Relation created at the server.

4.4.5 rs_insSBR

This module is responsible for inserting tuples into an SBR. This takes as arguments the SBR name, a transaction number, the client host name and a socket descriptor. The following operations are performed in this module.

- The SBRS table is queried to determine if an Shared Base Relation exists on the server with the above specified name. If no such table exists an error message is printed.
- The last time stamp value for this table is retrieved from the SBRS table.
- The tuples that are to be inserted into the Shared Base Relation must be sent by the client. The client sends the tuple file over the socket. This tuple file has to be read in. Illustra SQL has a 'copy' command that can directly load the contents of a file into a table. However this requires that the file have each tuple on a separate line and each field of a tuple must be separated by a TAB character. This is not the format in which the client sends the file. Hence the file must be read in from the socket and stored in a buffer. While in the buffer the necessary changes in format must be made and then this buffer should be written to a file. All this functionality is provided by a function called 'tcp_server_read_and_convert_file'. This function reads the tuple file from the socket and stores it in a UNIX file in the proper format.
- The SQL 'copy' command is then issued to copy this tuple file into a table.
- The last time stamp value for this Shared Base Relation is then modified in the SBRS table.

4.4.6 rs_droSBR

This module is responsible for dropping an SBR from the database. It takes as arguments an SBR name, a transaction number, and a client host name. The following operations are performed in this module.

- It is first ensured that the insertions table and the deletions table corresponding to the SBR are indeed present in the database. If either of those tables is absent an error message is printed and the module terminates.
- The SBRS table is queried to determine if a tuple corresponding to the Shared Base Relation exists.
- If the number of views derived from this table are non zero then this table cannot be dropped.
- If there are no views derived from this table the tuple corresponding to this SBR is dropped from the SBRS table and the INS_SBRS table.
- The insertions and deletions tables of the SBR are dropped from the database.
- A tuple containing the SBR name and time stamp is inserted into the DEL_SBRS table for this Shared Base Relation.

4.4.7 rs_delSBR

This module is responsible for deleting tuples from an SBR given a certain deletion condition. It takes as arguments an SBR name, a transaction number and a deletion condition. The following operations are performed in this module.

- The SBRS table is queried to ensure that an SBR with the given name exists in the database. If no SBR exists an error message is printed and the module returns.
- The last time stamp value for the SBR is retrieved from the SBRS table.

- The tuples corresponding to the deletion condition are inserted into the deletions table of the SBR.
- The tuples corresponding to the deletion condition are deleted from the insertions table of the Shared Base Relation.
- The last time stamp value for the Shared Base Relation is updated in the SBRS table.

4.4.8 rs_creSEL

This module executes the select operation on a given SBR. It also creates a selection binding for the given selection condition. A selection binding is a view of the SBR. This module takes as arguments the operand SBR, the transaction number, the client host name, the selection condition and the socket descriptor. The following operations are performed in this module.

- The SBRS table is queried to ensure that the operand SBR exists in the database.
- The last time stamp value for the SBR is determined from the SBRS table.
- The tuple in the SBRS table corresponding to this SBR is updated by incrementing the number of derivations field. A derivation is a view of the SBR.
- The BINDS table stores all the currently active bindings (views) for all the Shared Base Relations. This table is queried to determine if any binding already exists corresponding to the given selection condition.

- If a binding exists then the BINDS table is update to increment the usage of the binding by one.
- If no binding already exists then a new binding is created for the selection condition.
- The binding name and time stamp value are returned to the client by writing them to the socket.
- The Illustra SQL 'select' command is created from the selection condition.
- This SQL command is executed by invoking the 'sql_command' module which executes the SQL command and returns the results to the client.

4.4.9 rs_updSEL

For every selection binding the client creates a table in its local database that serves as a view on an SBR. Every time that view is queried or used at the client it must be updated first since more tuples may have been added to the SBR since the last update. This module is responsible for updating the selection binding. It takes as arguments an SBR name, a selection condition, a transaction number, a binding name, the previous time stamp value and a socket descriptor. The following operations are performed in this module.

- The SBRS table is queried to verify that the Shared Base Relation exists in the database.
- The last time stamp value is retrieved from the SBRS table for the given Shared Base Relation. This time stamp represents the last time, tuples were added to the Shared Base Relation and is called the new time stamp.

- If the old time stamp value is equal to the new time stamp value then it implies that the client view of the SBR is up to date, hence nothing further needs to be done.

- If the new time stamp is greater than the old time stamp value, then the new time stamp is returned to the client for future reference.

- Some tuples may have been deleted from the SBR since the last update. An SQL command is executed by invoking the 'sql_command' module. This command returns to the client from the deletions table of the SBR, the tuples that were deleted since the last update.

- An SQL 'select' command is invoked on the insertions table of the SBR to select those tuples that satisfy the selection condition and have a time stamp value greater than the time stamp of the client. These tuples are returned to the client by the 'sql_command' module. Hence only the new tuples satisfying the selection condition are returned to the client.

4.4.10 rs_droSEL

This module is responsible for dropping a selection binding. It takes as arguments an SBR name, a transaction number, a client host name and a binding name. The following operations are performed in this module.

- The BINDS table is queried to ensure that a binding of the given name exists.

- If the usage of the binding is 1, then it implies that only one view exists and hence this binding can be dropped from the BINDS table.

- The SBRS table is updated to decrement the number of derivations (views) corresponding to the given Shared Base Relation.

- If the usage of the binding is greater than 1 then the BINDS table is updated to decrement the usage of the binding.

4.4.11 rs_creJOI

This module is responsible for executing a 'join' on two SBRs and creating a binding corresponding to the join. The client stores the results of the join in a table in its local database. This serves as a view on the two SBRs. This module takes as arguments two SBR names, a transaction number, a join condition and a socket descriptor as arguments. The following operations are performed in this module.

- The SBRS table is queried to ensure that both the Shared Base Relations exist at the server. If they don't an error message is printed and the module returns.

- The last time stamp values for both the SBRs are determined from the SBRS table.

- The SBRS table is updated to increment the number of derivations for both the Shared Base Relations.

- The BINDS table is queried to determine if any binding already exists corresponding to the given join condition.

- If a binding already exists then the BINDS table is updated by incrementing the usage of the binding.

- If no binding exists then a new binding is created for the join condition in the BINDS table.
- The binding name and the last time stamp values of the two SBRs are returned to the client for future reference.
- The Illustra SQL 'join' command is composed from the join condition.
- The 'sql_command' module is invoked to execute the command and return the results of the join to the client.

4.4.12 rs_updJOI

As mentioned above, the client stores the results of a join in a table in its local database. Whenever this table is used or queried, it must first be updated by the server. This module is responsible for updating the client view of the join. It takes as arguments two SBR names, the old time stamp values for the SBRs, a join binding name, a join condition, a transaction number and a socket descriptor as arguments. The following operations are performed in this module.

- The SBRS table is queried to determine if the two Shared Base Relations specified exist at the server. If they don't exist then an error message is printed and the module returns.
- The last time stamp values are retrieved for the two Shared Base Relations from the SBRS table. These time stamps represent the last time these SBRs were modified. These now represent the new time stamps.
- If both the new time stamp values are equal to the corresponding old time stamp values then the client view is up to date and no further processing

is needed.

- If however any one of the new time stamp values is greater than the corresponding old time stamp values then some processing is needed to update the client view.
- The new time stamp values are returned to the client for future reference.
- Some tuples may have been dropped from the SBRs since the last client update. The tuples that have been dropped from each SBR are returned to the client by executing an SQL 'select' on the deletions tables of the SBRs.
- We needn't recompute the entire join. We only need to update the client view. So for this some temporary tables are required. If the new time stamp of SBR1 is greater than its old time stamp then a temporary table is created by selecting only the new tuples from SBR1. The same procedure is carried out for SBR2 as well. If any one of the new time stamps is equal to the old time stamps, then only one temporary table is required.
- A join is computed between the temporary SBR1 table and the entire SBR2 table. These results are returned to the client. However this isn't the complete updated join.
- A join is computed between the temporary SBR2 table and the tuples in the SBR1 table, that have a time stamp less than or equal to the old time stamp for SBR1. This is done to avoid duplicating tuples. The results of this join are returned to the client.
- The temporary tables are then dropped.

4.4.13 rs_droJOI

This module is responsible for dropping a join binding. It takes as arguments the two SBR names, a transaction number, a client host name and a binding name. The following operations are performed in this module.

- The BINDS table is queried to ensure that a binding of the given name exists.
- If the usage of the binding is 1, then it implies that only one view exists and hence this binding can be dropped from the BINDS table.
- The SBRS table is updated to decrement the number of derivations (views) for both the operand Shared Base Relations.
- If the usage of the binding is greater than 1 then the BINDS table is updated to decrement the usage of the binding.

4.4.14 rs_creIND

This module creates an index on an SBR on a specific attribute. It takes as arguments an SBR name, an attribute name and a transaction number. The following operations are performed in this module.

- The SBRS table is queried to ensure that an SBR with the given name is indeed present in the database.
- The index name is not returned to the client. Hence the only way the client can refer to an index on an SBR is by specifying the SBR name and the attribute name. The index name is constructed as follows: All index

names start with 'i_'. This is concatenated with the SBR name followed by an '_' followed by the attribute name. As an example consider an SBR called **person** which has an attribute called **person_name**. The name of the index on this attribute would be 'i_person_person_name'.

- The 'create index' SQL command is constructed and executed.

4.4.15 rs_rmvIND

This module removes an index on an SBR on a specific attribute. This module takes as arguments an SBR name, an attribute name and a transaction number. The following operations are performed in this module.

- The SBRS table is queried to ensure that an SBR with the given name is indeed present in the database.
- The index is constructed from the SBR name and the attribute name as explained above.
- The 'drop index' SQL command is constructed and executed.

4.4.16 rs_updCAT

This module updates the client catalog. It is invoked each time the client starts a new session with this server. It takes as arguments a transaction number, the old time stamp value read in from the client and a socket descriptor. The following operations are performed in this module.

- The server has the new time stamp value, which is compared with the old time stamp. If they are equal then the client catalog is up to date and

nothing further needs to be done.

- If the new time stamp is greater than the old time stamp then the names of Shared Base Relations that have been added and deleted since the last update must be sent to the client.

- The 'sql_command' module is invoked to send to the client the names of Shared Base Relations that have been deleted since the last client update. This information is retrieved from the DEL_SBRS table.

- The names of Shared Base Relations added since the last update are also sent to the client. This information is retrieved from the INS_SBRS table. The schema files for the new SBRS are also sent to the client.

4.4.17 rs_creSCJOI

This module is responsible for computing a join between a client table and a server SBR. It takes as arguments an SBR name (SBR1), a client SBR name (SBR2), a join condition, a transaction number and a socket descriptor as arguments. The following operations are performed in this module.

- The SBRS table is queried to ensure that SBR1 is indeed present at the server. If it isn't present an error message is printed and the module returns.

- The last time stamp value for SBR1 is determined from the SBRS table.

- The SBRS table is updated by incrementing the number of derivations for SBR1.

- The BINDS table is queried to determine if any join binding exists for the given join condition on the two SBRs. If a join binding exists then the usage of the binding is incremented by one.
- If no binding exists then a new server-client join binding is created for the join condition and the two SBRs.
- The new time stamp value for SBR1 and the scjoin binding name is returned to the client.
- To create a join with a client table the client table must be present on the server. Hence a temporary table has to be created on the server.
- The schema file for the client table is sent by the client and is read from the socket.
- The projection of the client table on the join attributes is also read from the socket and converted into a format that Illustra requires.
- The schema for the client table is read in from the schema file and the client table is created.
- The Illustra SQL 'copy' command is executed to copy the tuples from a file into the client table.
- The SQL 'join' command is constructed from the join condition. This command is executed and the results are returned to the client.
- The temporary client table is dropped from the server database.

4.4.18 rs_updSCJOI

The client view of a server-client join must be updated each time, before it can be used. This module is responsible for this task. It takes as arguments a server SBR name (SBR1), a client SBR name (SBR2) their respective old time stamps, a binding name, a transaction number, a join condition and a socket descriptor. The following operations are performed in this module.

- The SBRS table is queried to ensure that SBR1 is present in the server database. If it isn't present an error message is printed and the module returns.
- The SBRS table is queried to determine the new time stamp value for SBR1.
- The client sends the new time stamp value for its client table.
- If both the new time stamp values are equal to the respective old time stamps, then nothing further needs to be done and the module returns.
- The new time stamp of SBR1 is returned to the client.
- The updated join has to be computed at either the server or the client. ADMS has a method of optimizing the join site based on the table sizes. The updated server-client join is computed in two parts similar to the plain join. One part is computed at the client and the other part is computed at the server.
- If the new time stamp of SBR1 is greater than the old time stamp, then the changes to the table need to be sent to the client. The deletions and

the insertions to the server SBR since the last time stamp, are selected and sent to the client. If the client is the join site then it computes the join between the new server SBR tuples and its entire client table. If the server is the join site then the client only computes the join between the new server SBR tuples and the old tuples in the client SBR.

- If the client table has been modified since the last update then the new tuples are read in from the client and copied into a temporary table. The client makes the adjustment locally for tuples that may have been deleted from its table since the last update. If the server is the join site then the server computes the join between the new client tuples and its entire table. If the client is the join site then the server only computes a join between the new client tuples and the old server tuples. The server sends the results of its join back to the client.

- These two joins ensure that all the tuples are covered and the updated join is available. The temporary client table is then dropped.

4.4.19 rs_droSCJOI

This module is responsible for dropping a server-client join binding. It takes as arguments a server SBR name (SBR1), a client SBR name (SBR2), a binding name and a transaction number. The following operations are performed in this module.

- The BINDS table is queried to ensure that a binding with the specified name exists in the server database.

- If the usage of the binding is 1, it is dropped from the BINDS table. The SBRS table is updated by decrementing the number of derivations for the server SBR.

- If the usage of the binding is greater than 1, then it cannot be dropped. The BINDS table is then updated by decrementing the usage for the given binding.

4.4.20 rs_creSSJOI

This module is responsible for computing a join between a local server SBR and a remote server SBR. It takes as arguments a local SBR name (SBR1), a remote SBR name (SBR2), a join condition, a transaction number, the join site and a socket descriptor as arguments. The following operations are performed in this module.

- The SBRS table is queried to ensure that SBR1 is indeed present at the server. If it isn't present an error message is printed and the module returns.

- The last time stamp value for SBR1 is determined from the SBRS table.

- The SBRS table is updated by incrementing the number of derivations for SBR1.

- The BINDS table is queried to determine if any join binding exists for the given join condition on the two SBRs. If a join binding exists then the usage of the binding is incremented by one.

- If no binding exists then a new server-server join binding has to be created for the join condition. If the join site is local, then a new binding is created in the BINDS table and the name is returned to the remote server. If the join site is remote, then the new binding name is read in from the socket and a new entry in created in the BINDS table.

- The new time stamp value for SBR1 is returned to the other server.

- To create a join with a remote server table, the server SBR must be present on the server. Hence a temporary table has to be created on the server.

- The projection of the local SBR on the join attribute is computed and the results are sent to the client.

- The schema file for the remote server table is sent by the other server and is read from the socket. The schema file is actually sent from the other server to the client which forwards it.

- The projection of the remote server table on the join attributes is also read from the socket and converted into a format that Illustra requires.

- The projection schema for the remote server table is read in from the schema file and the temporary table is created.

- The Illustra SQL 'copy' command is executed to copy the tuples from a file into the temporary table.

- The SQL 'join' command is constructed from the join condition. This command is executed and the results are returned to the client.

- The temporary table is dropped from the server database.

4.4.21 rs_updSSJOI

The client view of a server-server join must be updated each time, before it can be used. This module is responsible for this task. It takes as arguments a server SBR name (SBR1), a remote server SBR name (SBR2) their respective old time stamps, a binding name, a transaction number, a join condition and a socket descriptor. The following operations are performed in this module.

- The SBRS table is queried to ensure that SBR1 is present in the server database. If it isn't present an error message is printed and the module returns.
- The SBRS table is queried to determine the new time stamp value for SBR1.
- The client sends the new time stamp value for the remote SBR.
- If both the new time stamp values are equal to the respective old time stamps, then nothing further needs to be done and the module returns.
- The new time stamp of SBR1 is returned to the client, which then forwards it to the other server.
- The updated join has to be computed at either the local server or the remote server. ADMS has a method of optimizing the join site based on the table sizes. The updated server-server join is computed in two parts similar to the plain join. One part is computed at each server.
- If the new time stamp of SBR1 is greater than the old time stamp then the changes to the table need to be sent to the client which then forwards

them to the other server. The deletions and insertions to the local SBR since the last time stamp, are selected and sent to the client.

- If the other server table has been modified since the last update, then the changes to the table are read in from the client and copied into a temporary table.

- Depending on the join site, one server computes the join between the remote server's new tuples and its entire table, and the other server computes a join between the new remote tuples and its old tuples.

- The results of both joins are forwarded to the client. These two joins ensure that all the tuples are covered and the updated join is available. Temporary tables are dropped from both servers.

4.4.22 rs_droSSJOI

This module is responsible for dropping a server-server join binding. It takes as arguments a local server SBR name (SBR1), a remote server SBR name (SBR2), a binding name and a transaction number. The following operations are performed in this module.

- The BINDS table is queried to ensure that a binding with the specified name exists in the server database.

- If the usage of the binding is 1, it is dropped from the BINDS table. The SBRS table is updated by decrementing the number of derivations for the local server SBR.

- If the usage of the binding is greater than 1, then it cannot be dropped. The BINDS table is then updated by decrementing the usage for the given binding.

4.5 Summary and innovative concepts

In this chapter an architecture and model for integrating heterogeneous databases was presented. This methodology has been implemented and the implementation details and features were discussed. There is a growing need for integration of heterogeneous database both for network management and other applications. Applications can now transparently access data from multiple database systems.

The architecture is not a plain client-server architecture. The clients are more powerful and handle some of the processing overhead. Since memory is relatively cheap the memory at the client side is enhanced. The clients have a local DBMS. The client caches some of the data it receives from the servers. Client caching is a very useful concept and significantly improves performance. Relational database tables are potentially extremely large. It is very inefficient to download large tables of data each time an application at the client side requests some data. Instead the first time the data is requested it is stored as a table in the client's local database. Each time that table needs to be accessed the client first updates its view of the table from the server and then it is available for query processing. By the use of time stamps for all tuples in each table the server only sends the new tuples added to the table instead of sending the entire table again. This significantly reduces the amount of data that needs to be transferred making query processing much faster. This is the method

of incremental updates. In cases where joins need to be computed between two relational tables the processing is distributed between the client and the server or between two servers. This parallel processing of queries also leads to significant improvement in performance. The same principles apply to clients keeping views of server tables. As a result of using techniques such as client caching, incremental updates and distributed query processing the performance of this system is extremely good. As we saw in the chapter on modeling for Network Management, views are always maintained by network operators.

Another very important feature is that a whole new query language did not have to be developed for this. The query language used here is just an enhanced version of SQL. It would be relatively easy to take existing SQL applications and make them operate over this system. The architecture is very scalable. If a new server has to added no modifications are necessary to the client or to any of the other servers. The client does not transmit actual queries to the server but in fact sends only a codeword. This ensures that even systems with slightly different versions of SQL can be integrated easily and the client need not be concerned with the differences in the query languages at the servers. It should be possible to integrate Object oriented and Object-Relational Database systems using this architecture. However in that case the client itself should have some object-oriented or object-relational capability.

Chapter 5

Configuration Management over Heterogeneous Databases

In the previous chapter a model for integrating heterogeneous databases and it's implementation were described. This chapter discusses a prototype Configuration Management System that was developed on the integrated database platform.

5.1 Network Configuration

The various functions performed by the Configuration Management functional module, were dealt with in detail in chapter 2. A one line description of Configuration Management functionality is, defining, monitoring and controlling of network resources and data. The network that is considered in this work is a Satellite Telecommunications Network operated by Hughes Network Systems(HNS). It is a wide area star network.

The network consists of a Satellite, a central Hub and several Remotes. There are many geographically dispersed **Remotes**. A Remote is a satellite dish along

with some equipment. Each Remote station is what is called a Personal Earth Station by HNS and is typically connected to some user equipment (e.g LAN). A lot of different equipment can be connected to a single Remote through the Remote ports. Each customer can have several Remotes. All communication between customer Remotes must go through a centrally located **Hub**. All traffic between any two Remotes first goes up to the satellite and then down to the Hub and then up to the satellite again and finally down to the destination Remote. Along with the hub there is a systems control center (SCC) that manages the network. The SCC handles the entire Network Management functionality. All satellite links into the Hub are called **inroutes** and the outgoing ones are called **outroutes**.

The Configuration Management system for this network handles two tasks:

- Session setup: Consider a customer connection between Remote A and Remote B. Due to certain faults in some part of the network this connection may have to be taken down. Once the fault is rectified the operator is responsible for setting up the connection once again. Another example is customer wanting a permanent connection established between two of his/her Remote stations. In either case the network operator has to set up the session and should have a good tool that would aid him in his task.

- Configuration: Each time a new customer Remote station is installed the operator is responsible for configuring it and creating a new Remote object, in the database that contains all the configuration data. This information is useful later on for setting up sessions and in isolating faults etc.

There is just one Hub that is shared by all the customers. The Hub is

hierarchically organized into the following:

- Network Groups: The Hub is usually partitioned into many disjoint Network Groups. Usually customers with a large number of Remotes are assigned a single Network Group. Each Network Group may be assigned a database of its own. Each Network Group is partitioned into many Networks.

- Network: There could be up to 7 disjoint Networks in any Network Group. A Network is a further partitioned into Data Port Clusters.

- DPC: A Data Port Cluster (DPC) is a chassis containing a collection of hardware cards. A Network could have up to 12 DPCs. A DPC is partitioned into many shelves. Each shelf is called a Line Interface Module.

- LIM: A Line Interface Module (LIM) is a shelf containing many slots, into which hardware cards are inserted. Each DPC can contain up to 3 LIMs.

- Ports: A hardware card typically contains one port. Each LIM can contain up to 8 ports. A Port is where any equipment is ultimately connected. Hence though a piece of equipment may be logically connected to a Network or DPC it is ultimately connected to a port.

A Remote also contains a similar hierarchy. A Remote however has no Networks. A Remote is partitioned into many Data Port Clusters (DPC). Each DPC is then subdivided into Line Interface Modules (LIM) and finally LIMs contain Ports. Customer LANs or workstations are connected to these Remote Ports. Typical customers are Petroleum companies. A gas station would be installed

with a remote to check credit card information or down load billing information. Each message transmitted usually tends to be short.

5.2 Network Model

In order to manage the above network the configuration information has to be stored in a database or databases in our case. Before this can be accomplished a model has to be developed for the network. (As part of previous work [6], a Configuration Management System has been developed for this network based on an object oriented data model and using ObjectStore, an object-oriented database, to store the data.) Since we are dealing with relational databases in this application, we have developed an E-R model for this network. Figure 5.1 illustrates the Entity-Relationship diagram that was developed to model this network.

The following Entities were identified.

- **Network**: This is the Network described in the previous section. It has the following attributes. The id is the primary key.

  Attributes

  id

  name

- **DPC**: This is the Data Port Cluster entity described in the previous section. The id is the primary key.

  Attributes

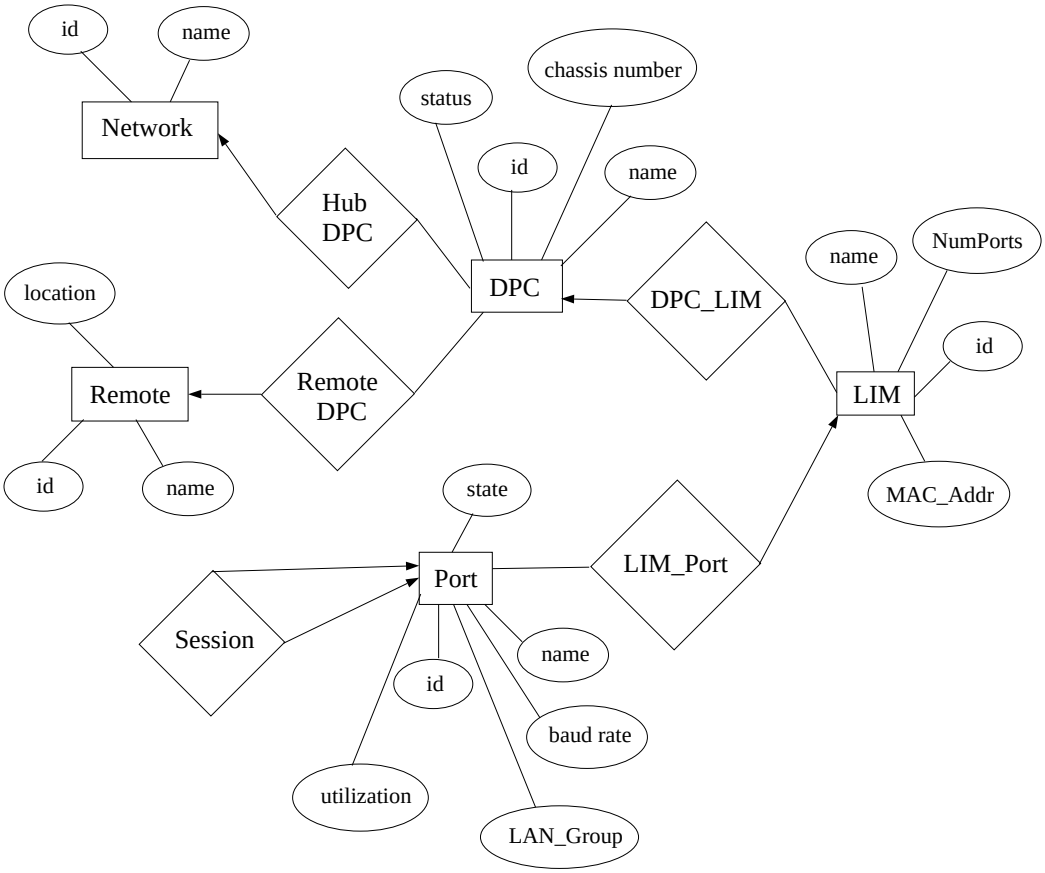

Figure 5.1: Entity-Relationship diagram for Configuration Management

id

name

status

chassis number

- **Remote**: This is the Remote entity described earlier. The id is the primary key.

  Attributes

  id

  name

  location

- **LIM**: This is the Line Interface Module entity described earlier. The id is the primary key.

  Attributes

  id

  name

  MAC_Address

  NumPorts

- **Port**: This is the Port entity described earlier.

  Attributes

id

name

state

baud_rate

utilization

LAN_Group

The following Relationships have been identified between the entities.

- There is a one to many contains relationship between a Network and DPCs. It is called HubDPC.
- There is a one to many contains relationship between a Remote and DPCs. It is called RemoteDPC.
- There is a one to many contains relationship between a DPC and LIMs. It is called DPC_LIM.
- There is a one to many contains relationship between a LIM and Ports. It is called LIM_Ports.
- There is a one to one relationship between ports. This is called a Session. Typically one hub port is connected to one remote port.

5.3 Graphical User Interface Design

It is very important that the operator has a good Graphical Interface that allows him/her to interact with the database easily and at the same presents the infor-

mation to him in a meaningful manner. Current Network Management Systems make the operator fill out a large number of forms. The operator console is cluttered with forms and it becomes difficult for the operator to concentrate on the task at hand. It is also difficult for the operator to remember which forms have been filled and which ones have not. Our aim is to provide the operator with a simple and easy to use interface. A checklist of tasks completed by the operator is also provided so that the operator does not have to remember how many tasks he still needs to complete.

From the description of the network being managed it is evident that there is a hierarchy on both the hub side as well as the remote side. Earlier work [12] has shown that for hierarchically organized data two visualization tools, the **Treemap** and the **Treebrowser** are especially good.

- **Treemap**: The Treemap is a tool that maps hierarchical information in 2-dimensions. At the top level the screen is split vertically to show the subdivisions of the top level. As the hierarchy is traversed the screen splits horizontally and vertically alternately to show all the subdivisions. It has been shown in earlier work that this tool is capable of displaying an order of magnitude more data than traditional tools.

- **Treebrowser**: This is also a good visualization tool for viewing hierarchical data. This just draws the logical tree structure on the screen. This cannot represent as much information as the Treemap and is a good tool if the data set isn't extremely large. Usually a scroll bar is provided for the user to be able to scroll through and view the entire tree, since there are limits to what can be displayed on a screen.

In addition to these visualization tools another scroll bar is provided so that the operator can execute a dynamic query on the database. The user can also update the database through the user interface.

5.4 Implementation

Recall that the integrated database platform was developed by integrating several Relational databases. Hence the E-R model developed is converted into Relational tables to be stored in these Relational databases. Each entity identified corresponds to one relational table, with the attributes mentioned earlier. In addition, one table is required for each relationship. The attributes of tables corresponding to the relationships are just the primary keys of the entities involved in the relationship. The tables created are:

- **Network** - id, name
- **Remote** - id, name, location
- **DPC** - id, name, status, chassis number
- **LIM** - id, name, MAC-Addr, NumPorts
- **Port** - id, name, state, baud_rate, utilization, LAN_Group
- **HubDPC** - network_id, dpc_id
- **RemoteDPC** - remote_id, dpc_id
- **DPC_LIM** - dpc_id, lim_id
- **LIM_Port** - lim_id, port_id

- **Session** - port1_id, port2_id

Figure 5.2 shows the architecture of the system. Plus-Minus (+/-) [8] is the name of the Integrated Database Architecture developed in the previous chapter. The client developed at the ADMS end is called ADMS_Minus. Each of the servers developed is called a Plus. For this application we only use Illustra_Plus, ADMS_Plus and Ingres_Plus. Illustra_Plus was developed as part of this work. Ingres_Plus [17] and ADMS_Plus were developed earlier. ADMS_Plus is a local server. Recall that these servers are built on top of the supporting DBMS. The Graphical User Interface interacts with ADMS_Minus. ADMS_Minus processes the client queries and determines where the tables being queried are located and sends the query to the respective Plus server. The results of the query are returned to ADMS_Minus which then returns them to the application.

The code is written in C and X/Motif is used to develop the graphical user interface. The Treemap, Treebrowser and Range widgets are not standard Motif widgets. They were developed as part of earlier work and are reused in this application. The interaction with the database is through C function calls that have SQL commands embedded in them. A C function is called to issue queries to the database and return the results. The results are processed by the application. The actual location of the tables is transparent to the application. This is one of the most important features of this application.

The purpose of this system is to allow an operator to view the elements available in the network. The operator can specify a certain parameter and then dynamically query the database to search for ports lying with the range specified for the parameter. The operator can then select a port from the query result and configure a session.

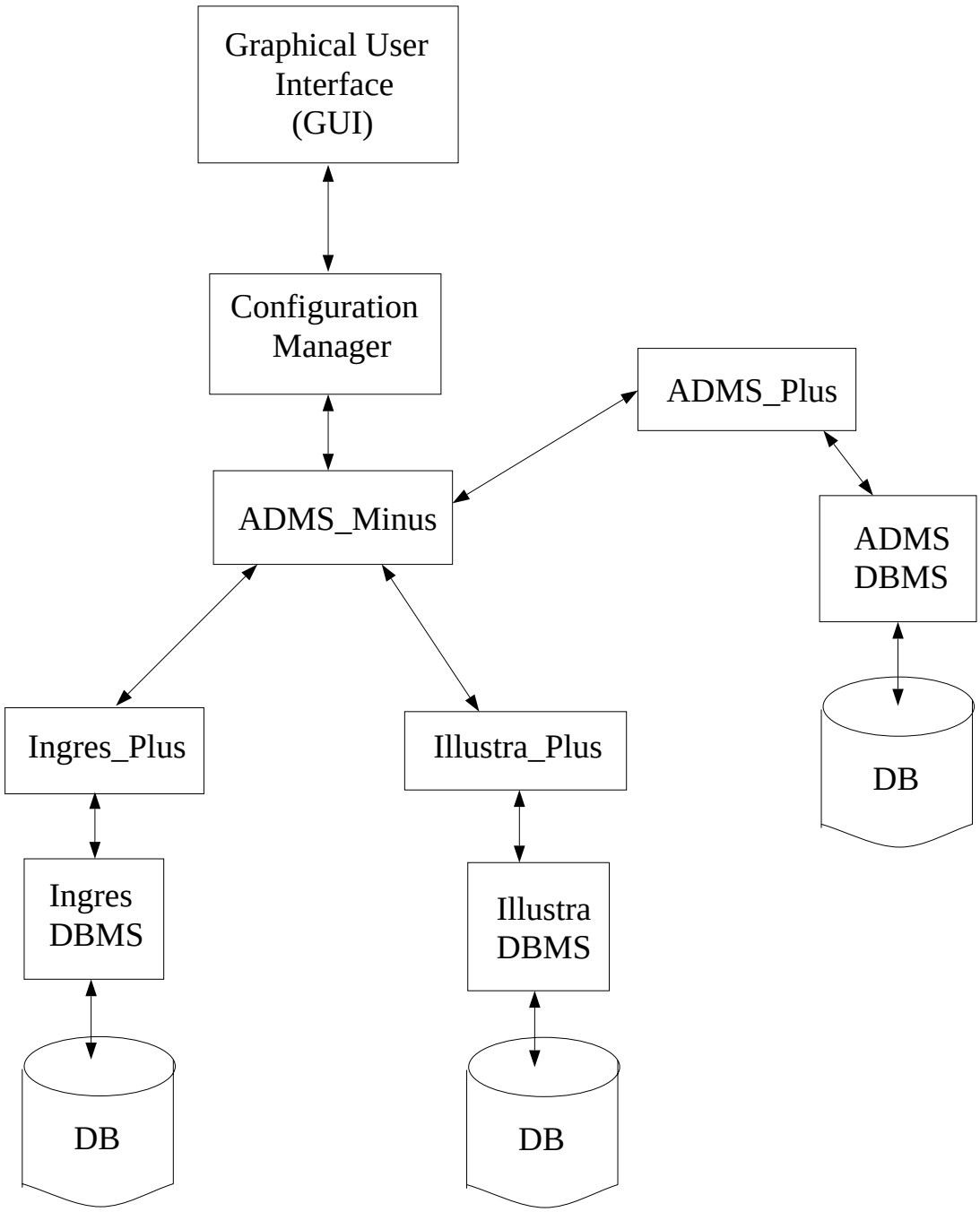

Figure 5.2: Configuration Management Architecture

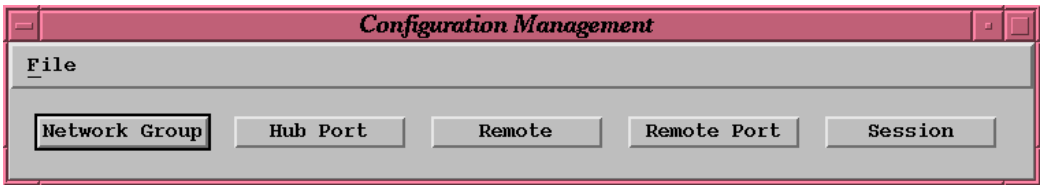

Figure 5.3: Configuration Management Main Panel

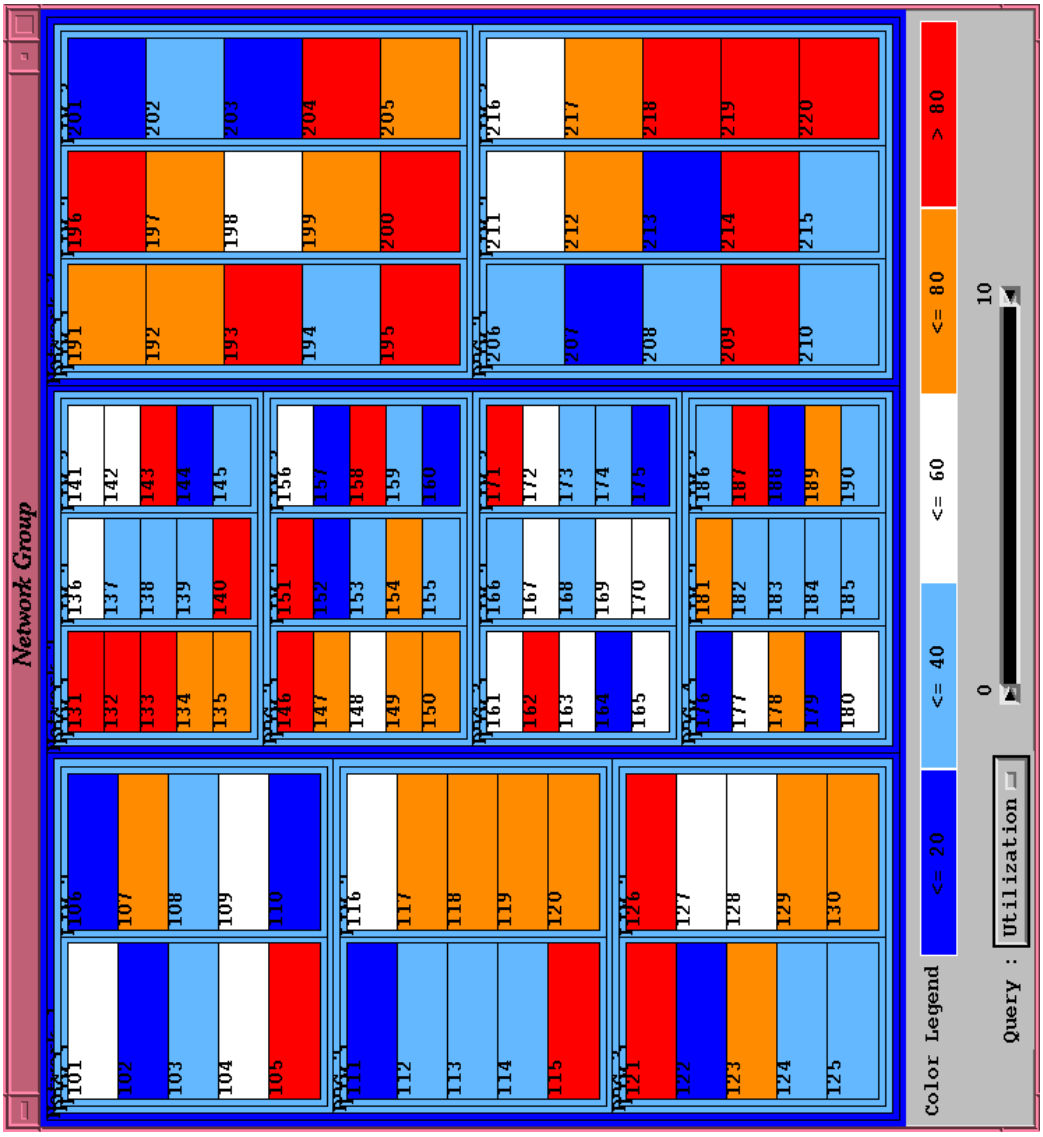

Figure 5.4: Treeemap Display

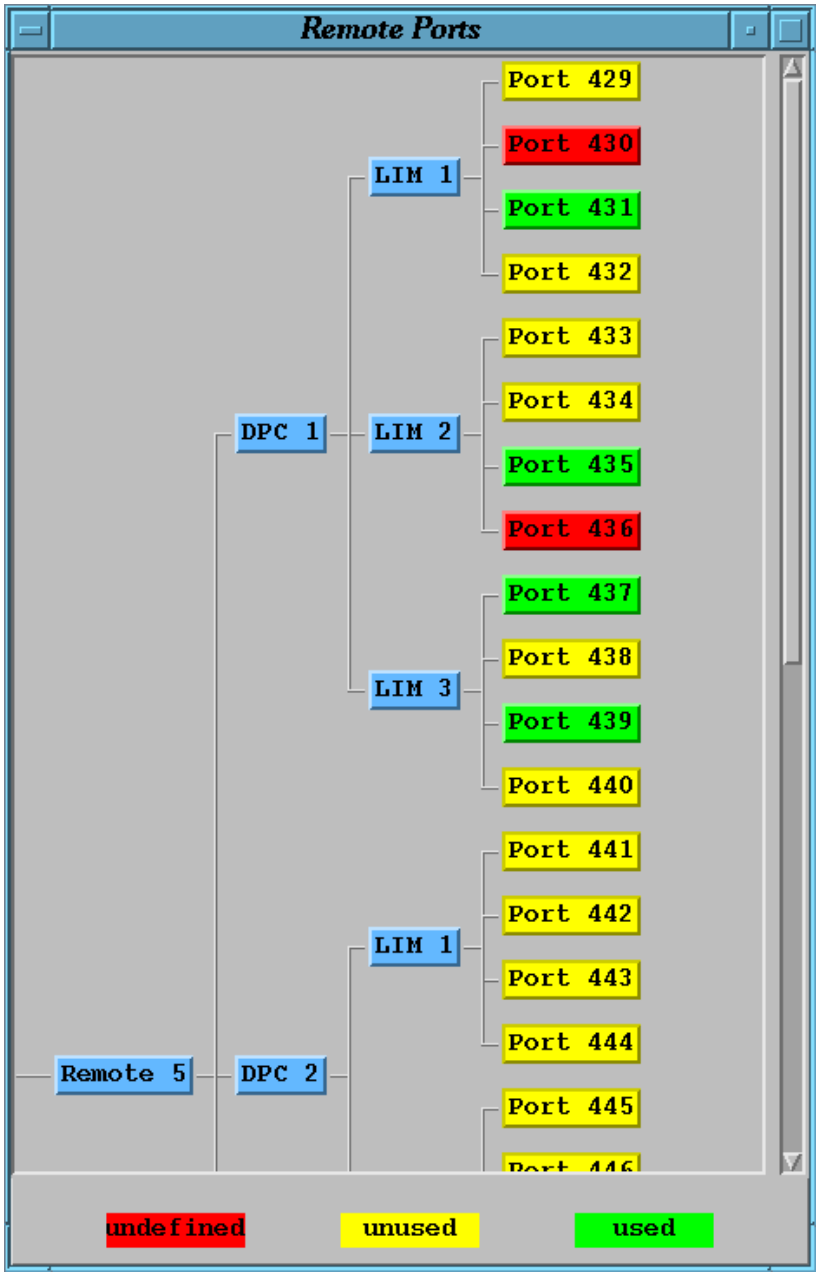

Figure 5.5: Treebrowser Display

Figure 5.3 shows the main panel of the User Interface. It has 5 buttons. The first button is a Network Group button. If the operator clicks this button a treemap is brought up on the screen showing only the Networks present within that Network Group. When the button is pressed a database query is sent out that will read from the database all the Networks present in the Network Group. The application has a tree data structure that is used to store the hierarchical information. The results for the Network, fill the first level of the tree. The Treemap widget is created from the application tree data structure, and is displayed on the screen. For simplicity buttons for displaying DPCs and LIMs are not provided. Moreover the operator will not be looking for DPCs and LIMs but will be looking for Ports. The present interface shows all the Ports present inside a Network Group. The buttons for displaying DPCs and LIMs can be added later without much effort.

The next button is used to display all ports that lie within the given Network Group. Figure 5.4 shows a screen dump of the Treemap display. For this application we consider only 1 Network Group. When this button is pressed a database query is issued that first reads the DPCs from the database. These DPCs are inserted as children of the appropriate Networks. Then the LIMs are read in from the database and inserted as children of their respective DPCs and finally the Ports are read in from the database and the tree is completely filled. The Treemap widget created earlier is filled with the data from the application tree structure and all the ports lying within the Network Group are displayed. The color of the ports is determined by their current utilization values. A color legend is displayed below the Treemap to show the ranges corresponding to the different colors. Currently only utilization is used to determine the color of the

Ports. This can be enhanced to provide a choice to the operator so that the color can be made to reflect other attributes also.

A Range widget with two sliders is also provided below the Treemap. An option menu is provided alongside so that the operator can select the attribute he/she wants to query on. Currently the operator can query Ports only on the basis of Utilization and Baud Rate. Each time the slider is moved by the operator a dynamic query is issued to the database. The attribute on which the query is being executed is passed along with the minimum and maximum values of the query range. A database query is constructed from these values and executed. Only those Ports that satisfy the operator query remain colored while the others get grayed out. Based on the results of the query the operator selects a Port. This completes the selection of a Hub Port and this makes the Network Group and Hub Port buttons turn green indicating to the operator that these tasks have been completed.

The next task for the operator is to select a Remote. When the Remote button is pressed a query is sent out to the database requesting the names of all the Remotes that exist in the database. When the results are returned, a scrolled list of Remote names appears on the screen. The operator can then select one of those Remotes. This makes the Remote button turn green.

The operator presses the Remote Port button to see all the ports present in the Remote just selected by the operator. When this button is pressed a query is sent out to the database that reads all the Ports present inside the selected Remote. Similar to the Hub side the DPCs are read in first, then the LIMs and finally the Ports. The application tree structure is created from this data. The Treebrowser widget is created from the application tree data structure.

Figure 5.5 shows a screen dump of the treebrowser. A Remote typically has fewer ports than a Hub or Network Group so a Treebrowser can be used to display the Remote Ports. Remote Ports can have three states: used, unused and undefined. The three types of ports are shown in different colors. The operator selects an undefined port. This causes a form to pop up on the screen. The baud rate for this port has to be filled in by the operator. The baud rate entered by the operator is returned to the database and the relevant tuple in the Port table is updated. Once that is completed the Remote Port button also turns green.

The final task is actually setting up the Session. For this the operator presses the Session button which is the fifth button on the main panel. This brings up a form with all the information on the Hub and Remote Ports selected. The operator can ensure that the Ports selected are satisfactory and enter a Session name. This creates a new Session and a database update is sent to add a tuple to the Session table. The Remote Port is also marked as used in the database.

5.5 Summary and innovative concepts

In this chapter we developed a Configuration Management application for a Satellite network. This application has already been developed over an object-oriented database in previous work. The purpose of this effort was to demonstrate the same application running over the Integrated Database Platform. A brief description of the network being managed was given. A model was developed for the network and was illustrated by an E-R diagram. The different Graphical User Interface techniques used were discussed and finally the implementation was described.

A large number of organizations today, are faced with the problem of having many isolated applications running on different systems, which they now want to integrate. A properly implemented client/server middleware infrastructure can help them meet their needs. The middleware should provide seamless technical integration of disparate systems. By seamless we mean the differences between the environments should not be visible to the applications programmer and the communication should be transparent to the application. Performance and scalability are critical to the success of such an architecture. Performance is especially critical for applications such as Network Management where certain operations may have to be performed in real time. The important issue is that communication between the client and server should not become the bottleneck.

In the earlier chapter we discussed the implementation of a client-server architecture that can serve the purpose of a middleware. In this chapter we demonstrated through an application, the ability of the above architecture to perform the functions of a middleware. This is the first application to be developed over this integrated database platform and demonstrates how an application can transparently access data from remote disparate databases. The performance of this application was very satisfactory. The different tables required by this application could be present on any of the databases for which a "plus server" (as described in the previous chapter) exists. The syntax of the queries written to access the data is independent of the actual location of the data. This insulates the application from the differences in the query syntaxes of the different database systems. The use of client caching, incremental updates and the distribution of processing between the client and the servers significantly improve performance. The application interacts with the client so every time the appli-

cation access a table it is not necessary to access the data from the respective server. To ensure that the client has a consistent view of the data, each time a table is accessed by the application, the client is notified by the respective server, of any changes that may have been made to the server table. This is also the case for a table that may have formed as the result of a 'join' between two server tables or a server table and a client table.

Based on these ideas more complex Network Management Systems can be developed. It is also possible to take existing applications and modify them for this platform. Organizations can protect their investments in different applications, if they don't have to rewrite the applications each time they want to change the database platform on which the applications were developed. The purpose that such a platform serves is that the query syntax doesn't have to be made database specific and entire databases don't have to be transferred between different systems. As long as a Plus server exists for a DBMS it can be made to interoperate with other systems. Since the methodology for developing Plus servers exists very little effort is required to develop a Plus server for a DBMS.

Chapter 6

Conclusions and Future work

With technological advancement and increase in the complexity of telecommunication networks, it is imperative that there be commensurate advances in the tools and techniques used to manage these networks. However, as discussed earlier, today's network management tools haven't been able to keep abreast with the new networks and services being deployed. One of the main areas where development is required is in the modeling of the networks for management purposes, and the implementation of these models on database platforms. A large number of network management systems that exist today, don't cover all the network management functional areas or have completely independent applications to manage different functional areas. In this work we have developed an object oriented data model for hybrid network management. Our model is an integrated model covering all network management functional areas.

We have implemented our model on ObjectStore, an object oriented database system, to form the Management Information Base (MIB) of an Integrated Network Management System. We have developed Configuration and Performance Management modules based on this model. A simulation was developed to set up

connections between various nodes in the network and then vary the traffic over these connections. The simulation periodically reports performance statistics that are stored in the MIB.

Performance statistics have a utility in both the short term as well as the long term management. However, as the data gets older some form of compression can be tolerated. Based on this concept we proposed to have three levels of precision for storing performance data. This is effective in reducing the amount of storage required for this data. Since the number of network elements is very large, the total reduction in storage is significant. This also improves query performance.

We proposed two different models for storing performance data. These models were integrated with the structural or relatively static configuration management data. This is in itself a step forward from the way current systems handle performance statistics. The operator can now associate performance data with a particular network element. Since all the data is in the same database the operator can also issue complex queries that combine structural and statistics attributes. The relative merits of the two models were compared and one of the models was implemented and used to store performance statistics reported by the simulation.

Extensive experiments and simulations need to be undertaken to get a more complete comparison of the two performance data models. We implemented the simplest model as part of our work. Some complex extensions of these simple models like having different amounts of compression depending on the time of day, that were discussed in chapter 3 can also be implemented and their performance studied. An optimal strategy for the physical clustering of data in a

database also needs to be determined if one exists at all. The network operator uses performance statistics to study performance degradations and also for predicting the long term usage of network resources. Compression of data results in lower precision data. The effects of compression on prediction of resource usage hasn't been studied in this work. The time intervals over which data was to be compressed was ad hoc in our work. The time intervals could be chosen such that prediction isn't adversely affected. This can be studied as part of future work.

In the next step we would like to develop a Fault Management module. The simulation can be modified to artificially generate "faults" or performance degradations in the network. Certain patterns can be observed from performance data collected prior to faults. We might even try to automate certain fault management functions. Intelligence can be incorporated into the system so as to predict faults based on observations of performance statistics. We have modeled Alarms in our model but this hasn't been actually implemented in the MIB.

Alarm correlation is another area where there are interesting problems to be solved. In response to a single fault all the network elements that are affected in some way, generate an Alarm. All these alarms are reported to the central network control center. The operator has to sift through these alarms to actually determine the source of the fault. The system itself should have certain alarm correlation features built in, so that the onus is no longer on the operator. The system should be able to determine the actual source of the fault from all the alarms that are reported. There has been a lot of interest in this area and many studies have been conducted. An expert system or neural network may also be used for such purposes.

In chapter 4 we presented an architecture for the integration of heterogeneous database systems, and its implementation. We successfully developed a Configuration Management system on this platform. This demonstrated the ability of such a platform to perform the functions of a "middleware". The architecture is scalable and provided a very satisfactory performance. Hence it provides a "middleware" that can be used to integrate many network management applications that have already been developed over heterogeneous DBMSs. The MIB is the heart of any network management system and hence database interoperability is an important issue that has to be resolved before there can seamless integration of network management applications. This is very significant effort made in direction of database interoperability. There are certain products like ODBC that are available in the market today, but they lack many of the performance advantages that we have in our system which we obtained by using client caching, incremental updates etc.

The client-server architecture developed had its client on ADMS, and had servers for Ingres, Oracle and Illustra. All of these are relational database systems. (Even though Illustra is Object-Relational only the relational part has been used here.) The next step would be to integrate object-oriented and object-relational database systems into this integrated database system. The same methodology could be used to achieve this integration. However, the client would have to be developed on a system that has both object as well as relational capabilities. Most object-oriented DBMSs that are in the market today don't have relational capabilities. This makes an Object-Relational system such as Illustra, an ideal candidate system for the development of a client.

There are many challenges in developing here. In our implementation, a

relational tuple was the unit of tranfer. Each tuple had simple data types that are supported by all relational systems. However it isn't clear what the unit of tranfer will be when dealing with complex user defined data types. We may choose to transfer the entire object or only an object pointer. Returning object pointers has the disadvantage that client server communication is required every time the object is referenced. The advantages of client caching are lost. Communication overhead is increased and so is the processing at the servers. Objects contain pointers some of which may be pointers to other objects. This makes matters more difficult if entire objects are transferred. Also the OODBMSs available today don't conform to any standard and have different object models.

So far our network management system has been developed entirely on a centralized database system. As part of future work we would like to extend this development to a distributed database system. There are several interesting issues such as object migration, maintaining consistency between various replicas of an object etc. Recently there has been a growing interest in Distributed Object Management Systems. CORBA is an industry standard for the development of distributed object management applications. We would like to extend our object model and make in completely CORBA compatible. This would facilitate integration with a wide variety of systems.

Relational database systems have proved to be very useful for commercial business applications and are commonplace in financial and business data processing applications. They however don't have support for the requirements of advanced applications such as network management. Network management systems require to define many complex data types and require persistent storage for instances of such data types. This requirement has been clearly demon-

strated in the work presented in earlier chapters. The object model developed in chapter 2 can implemented on either a relational or object-oriented DBMS. If a relational DBMS is used, then it is the programmer's responsibility to flatten out the object hierarchy and map the objects into relational tables. The advantages of inheritance and polymorphism are also lost. The mapping between objects and tables and vice versa results is computational overhead and degrades performance. This is due to the inability of relational DBMSs to directly store user defined data types. Joins between relational tables is required to determine the relationship between two objects. In an OODBMS this is achieved by simple pointer chasing.

On the other hand OODBMSs aren't the panacea for network management applications either. Most OODBMSs available in the market today don't conform to the ODMG specifications for OODBMSs developed recently. The "engines" on which these DBMSs are developed are different. This makes interoperability difficult. Each of these DBMSs has limitations of its own. In our experience with ObjectStore we encountered certain scalability issues. Certain prelimnary experiments have shown that a relational DBMS may perform better for data that is essentially tabular. All performance statistics are basically tabular. Hence it might be more efficient to store this data in a relational DBMS. An OODBMS would be used to store the structural data. Hence a database having both relational as well as object oriented capabilities may be a better platform for the MIB.

The recent ODMG specifications for object oriented databases have tried to incorporate most of the features provided by relational systems. The query language OQL, has many of the features provided by SQL. Simultaneously, ANSI

and the ISO SQL standardization committees have been extending SQL by adding the capability to support user defined abstract data types and other object oriented capabilities such as inheritance and polymorphism. There is also a possibility that these standards may merge some time in the future. Recently there has been a growing interest in another class of systems called Object-Relational database systems (ORDBMS) such as Illustra and UniSQL. These systems are "SQLish" and have a relational engine on top of which object oriented features have been added. So they provide all the features of a relational system. The query language is a extension of SQL. Sets and collections for user defined types are also provided and can be queried. They sacrifice atomicity to provide object oriented features. Here an element of a tuple can be one of the simple data types or a user defined data type.

Using an ORDBMS as a MIB and studying it's performance is one of the directions in which the current work can progress. The object oriented features can be used for the development of the relatively static structural data model while the relational part can be used for sensor data. Once we have successfully integrated object oriented and relational database systems we may even be able to split the development of our data model over two different database systems.

Bibliography

[1] S. Bapat. OSI Management Information Base Implementation. In *Integrated Network Management II*, pages 817–831. North Holland, 1991.

[2] S. Bapat. *Object-Oriented Networks: models for architecture, operations and management.* Prentice Hall, 1994.

[3] A. Datta. A data model for Object-Oriented Network Management. April 1994.

[4] J. Haritsa, M. Ball, N. Roussopoulos, J. Baras, and A. Datta. Design of the MANDATE MIB. In *Integrated Network Management III*, pages 85–96. North Holland, 1993.

[5] J. Haritsa, S.K. Goli, and N. Roussopoulos. ICON: A System for Implementing Constraints in Object-based Networks. April 1994.

[6] Systems Integration Lab. Network Configuration Management System Design Document. April 1994. Hughes Network Systems and University of Maryland.

[7] E.H. Mamdani, R. Smith, and J. Callaghan. *The Management of Telecommunication Networks.* Ellis Horwood Limited, 1993.

[8] N. Roussopoulos and H. Kang. Principles and Techniques in the Design of ADMS+-. December 1986.

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling & Design*. Prentice Hall, 1991.

[10] D. Shen. Network Database Internet Draft. June 1991.

[11] W. Stallings. *SNMP, SNMPv2 and CMIP: The practical guide to Network Management Standards*. Addison-Wesley, 1993.

[12] M. Teittinen, C. Plaisant, H. Kumar, and B. Shneiderman. Visual Information Management for Network Configuration. March 1994.

[13] K. Terplan. *Communication Networks Management*. Prentice Hall, 1992.

[14] R. Valta. Design Concepts for a Global Network Management Database. In *Integrated Network Management II*, pages 777–788. North Holland, 1991.

[15] P. Wegner. Dimensions of Object-based Language Design. In *SIGPLAN Notices*, volume 22(12), pages 168–182. 1987.

[16] O. Wolfson, A. Dupuy, S. Sengupta, and Y. Yemini. Design of the NETMATE Network Management System. *IEEE Network*, March 1991.

[17] Z. Yao. Ingres Server using TCP/IP and Heterogeneous Database Join Method. April 1993.