Design and Implementation of a Fault Management Service for Heterogeneous Networks using the TINA Network Resource Architecture

Chetan Parshotam Chiba

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, December 2001

Declaration

I declare that this project report is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Signed this ___ day of _____ 20___

Chetan Parshotam Chiba.

Abstract

Faults are unavoidable and cause network downtime and degradation of large and complex communication networks. The need for fault management capabilities for improving network reliability is critical to rectify these faults. Current communication networks are moving towards the distributed computing environment enabling these networks to transport heterogeneous multimedia information across end to end connections. An advanced fault management system is thus required for such communication networks. Fault Management provides information on the status of the network by locating, detecting, identifying, isolating, and correcting network problems thereby increasing network reliability. The TINA (Telecommunication Information Networking Architecture) standards define a Network Resource Architecture (NRA) that provides a framework of a transport network that is capable of transporting heterogeneous multimedia media information across heterogeneous networks. TINA also defines a Management Architecture that follows the functional area organization defined in the OSI (Open Systems Interconnection) Management Framework, namely fault, configuration, accounting, performance, and security management (FCAPS). The aim of this project is to utilise the TINA NRA and Management Architecture concepts and principles to design and implement a distributed Fault Management Service for heterogeneous networks. The design presented here utilises TINA's fault management specifications, together with UML modelling tools to developed this Fault Management Service. The design incorporates the use of CORBA and SNMP to provide a distributed management functionality capable of providing fault management support across heterogeneous networks. The generic nature of the fault management service is tested on the SATINA Trial platform which consists of both an ATM network as well as an IP MPLS network. The report concludes that the Fault Management Service is applicable to any connection-oriented network that is modeled using the TINA NRA specification and principles.

Acknowledgements

This work was performed under the Centre for Telecommunications Access and Services at the University of the Witwatersrand. The centre is funded by Telkom SA Limited, Siemens Telecommunications and the Department of Trade and Industry's THRIP programme. This financial support is much appreciated.

I would like to thank my supervisors, Prof. Hu Hanrahan and Mr. Setumo Mohapi, for their guidance and support throughout the duration of the research project. In addition I would like to extend my appreciation to my colleagues, Hitesh Morar, Chien-Yu Chen, and the rest for their input, help and advice. Lastly, I would like to thank my parents and brother for their continuous support and encouragement throughout the duration of my studies.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CC	Connection Coordinator
CORBA	Common Object Request Broker Architecture
CO	Computational Objects
CP	Connection Performers
DPE	Distributed Processing Environment
EML	Element Management Layer
FC	Fault Coordinator
FCAPS	Fault Configuration Accounting Performance and Security
FM	Fault Management
FMS	Fault Management Service
GUI	Graphical User Interface
IDL	Interface Definition Language
IP	Internet Protocol
LNC	Layer Network Coordinator
MIB	Management Information Base
NRA	Network Resource Architecture
NGN	Next Generation Networks
NGTN	Next Generation Transport Networks
NML	Network Management Layer
NMS	Network Management System
NRIM	Network Resource Information Model
ODP	Open Distributed Processing
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
POTS	Plain Old Telephony Service
PSTN	Public Switched Telephone Network
RA	Resource Adapter

RCM	Resource Configuration Management
SATINA	South African TINA Trial
SML	Service Management Layer
SNMP	Simple Network Management Protocol
TCM	Tandem Connection Manager
TDS	Testing/Diagnostic Server
TINA	Telecommunications Information Networking Architecture
TMN	Telecommunications Management Network
ITU	International Telecommunications Union
UML	Unified Modeling Language

Chapter 1

Introduction

1.1 Fault Management

Communication networks consist of a multitude of hardware and software components that are bound to fail eventually. Such faults are unavoidable in large and complex communication networks, but quick detection and identification is needed to significantly improve network reliability. A single fault in a large communication network may result in many fault alarms, making isolation of the primary source of failure a difficult task. Since faults can lead to service unavailability, unacceptable network degradation and unanticipated disruptive behaviour, fault management is perhaps the most widely implemented network management element. As networks become larger and more complex, the need for advanced fault management capabilities become critical.

Fault Management provides information on the status of the network by locating, detecting, identifying, isolating, and correcting network problems. The primary benefit of fault management is that it increases network reliability by giving the network engineer tools to quickly detect problems and initiate recovery procedures to maintain a complete and continuous connectivity session between the users and the network. Fault management ensures network reliability and availability. In this sense, fault management serves as the foundation of other network management functions.

1.2 Distributed Networks and Fault Management

A distributed computing system consists of heterogeneous computing devices, communication networks, operating system services and applications. Fault management of distributed systems is necessary to provide the means to detect problems, isolate and locate their causes,

and perform the actions required to correct the problems. However, the the rapid growth of distributed systems presents many complexities. Some of the reasons that make the fault management of distributed systems complex are [1]:

- Components and services may be provided by different vendors with diverse and incompatible management interfaces, which make it difficult to apply a consistent integrated approach to fault management.
- The components and services being managed are physically distributed. Global state information on which to base management decisions, or to synchronise fault management actions on different components is therefore difficult to obtain.
- The size and complexity of future large distributed networks introduce problems of scale due to the number of components that has to be managed. Treating each component as an independent entity for fault management purposes is therefore made possible.

A distributed fault management service that deals with these complexities is therefore required to ensure network reliability and survivability in distributed networks.

1.3 Objective of this Project

The objective of this project is to design, implement and evaluate a distributed fault management service for heterogeneous transport networks. The TINA Management Architecture and the NRA provides the necessary tools to develop such a management system.

The TINA standard defines a NRA and Management Architecture that deals with the development of such management functionality. According to the TINA NRA, any transport network that is implemented using the TINA NRA specifications and principles can be managed using the TINA Management Architecture [2]. TINA's Management Architecture covers the concepts and principles for managing TINA systems and networks and draws heavily on the ITU's (International Telecommunications Union) TMN (Telecommunications Management Network) architecture adopting the management functional areas, FCAPS, from the OSI Management Framework [3]. The TINA NRA and Management architecture together offer a technology independent abstraction of network management functionality. The TINA-C Architecture also defines a DPE (Distributed Processing Environment) that allows application interoperability in a transparent way. This transparency enables different software components, contained in several heterogeneous computing nodes, to interact across different network domains in a distributed transparent way.

TINA thus provides the necessary tools for designing and developing a distributed fault management service.

The specific goals of the project are to:

- Utilise the TINA NRA and Management Architecture specifications and principles to design, implement and evaluate a distributed Fault Management Service for heterogeneous networks.
- Develop a Fault Management Service that monitors, detects and localises faults across heterogeneous networks.
- Demonstrate the Fault Management Service on the SATINA Trial platform.
- Evaluate the functionality of the fault management service.

1.4 SATINA - South African TINA Trial Platform

The SATINA Trial platform is a project undertaken [4] to investigate and develop an experimental TINA-based service and network connectivity platform, reflecting the requirements of future telecommunication service providers and network operators. The principal aim is to develop high level advanced telecommunication services, service management and network management based on a TINA Distributed Processing Environment using CORBA. The design and implementation of a Fault Management Service using TINA NRA detailed in this report forms part of the overall SATINA Trial project as one of the several other contributing sub-projects.

1.5 Overview of this Report

Chapter 2 provides an outline of TINA and its sub-architectures. Both the TINA NRA and the Management Architecture with respect to Fault Management are reviewed in this chapter. This chapter studies Fault Management activities, requirements, computational structure and their interactions with other Computational Objects (CO) in the TINA NRA environment. This chapter aids the design process described in chapter 3.

Chapter 3 presents a detailed analysis of the Fault Management Service (FMS) design process. The functional requirements for this particular Fault Management System are discussed in this chapter. The use of SNMP and CORBA in the design of the FMS are presented as well. UML modeling language is use to show graphically the relationships and

interactions between computational objects. Finally, the software components that fulfill the functional requirements for this FMS are described in this chapter.

Chapter 4 discusses the implementation of these components on the SATINA trial platform. The current status of the SATINA platform is described. The main focus of this chapter is the flow of events scenario of a typical Fault Management Service. To aid in this explanation, UML sequence diagrams and screen capture pictures of the Graphical User Interface (GUI) is used. This chapter concludes with the limitations of this design.

The conclusions drawn from this work are reviewed in Chapter 5. Also, certain recommendations for further work in this field are made.

Chapter 2

Network Management in the TINA Environment

The TINA (Telecommunications Information Networking Architecture) initiative provides a framework for all telecommunications software, encompassing components ranging from connection establishment through network and service management to service delivery and operation. TINA strongly supports the concepts of ODP (Open Distributed Processing) via its modeling techniques and viewpoints, and the use of a DPE (Distributed Processing Environment) for providing the generic facilities required by all software running in a distributed fashion. The TINA standard defines an approach that merges existing and established telecommunications software architectures and technologies such as the Intelligent Network (IN) and the Telecommunications Management Network (TMN).

The architecture provides for an all-encompassing business model, which includes the various role players in the industry such as the service providers, service users, network and DPE designers and retailers. The distinguishing feature of TINA is that all the services are software-based applications that operate on a distributed computing platform. The platform hides the underlying technologies and distributed concerns from the service applications, thus facilitating the construction of portable and interoperable code.

The TINA framework is decomposed into four architectures [3]:

- **The Service Architecture** defines the concepts and principles for the analysis, design, deployment, re-use and operations of service related software components supporting current and future advanced applications.
- **The Network Architecture** provides concepts for modelling the underlying network which provides the basic communication services required by the Service Architecture. The network architecture includes a high level view of network connections that is used by the services to satisfy their connectivity needs, and also includes generic

descriptions of technology-independent network element that can be specialised to particular technologies and characteristics.

- **The Management Architecture** covers the principles and concepts for managing TINA systems and networks and draws heavily on the ITU's TMN architecture. The management architecture aims to provide a set of generic management principles that are applied to the management of services, resources, software and underlying technology.

- **The Computing Architecture** specifies the framework for deploying and operating the software components of the Service and Management architectures. The computational architecture defines a DPE that provides the support for the distribution execution of software components in a transparent way. This architecture achieves inter-operability, distribution and component re-use through the facilities of the DPE.

The Network Architecture and the Management Architecture are the two sub-architectures applicable to this project. Network Management aspects in TINA are defined by the Network Resource Architecture (NRA).

2.1 TINA Management Architecture

TINA is based on the TMN layering principles and decomposes each of TINA's sub-architectures into Service, Resource and Element Layers. TINA's Management Architecture can therefore be split into service management, resource management and network element management. This project deals with the management of network elements.

2.1.1 Functional Areas in the Management of the Network Architecture

The TINA Management Architecture is decomposed according to the five OSI functional areas, Fault, Configuration, Accounting, Performance and Security (FCAPS). For the management of the resources that are inside the scope of the Network Architecture, TINA-C specializes the classical Configuration Management functional area. The TINA standard emphasizes the management of connections by identifying a new functional area, called Connection Management. The Connection Management functional area includes all the activities needed for the management of the connections. The rest of the activities within the classical Configuration Management functional area (resources installation, activation, status control, etc.) are classified within another new TINA-C functional area named Resource Configuration. Therefore, TINA-C defines six functional areas for the management

of the network resources: Accounting Management, Fault Management, Performance Management, Security Management, Resource Configuration and Connection Management (see Figure 2.1 [5]), the last two replacing and specialising the traditional Configuration Management functional area.

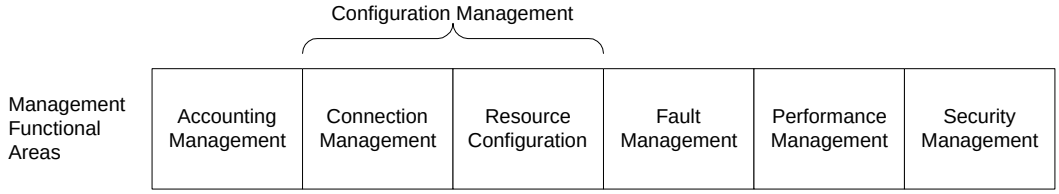

Figure 2.1: TINA Network Architecture Management Functional Areas

This project concentrates on the fault management of connection-oriented networks. Connection-oriented networks are based on TINA's Connection and Resource Configuration Management functional areas.

2.2 TINA Network Resource Architecture (NRA)

The TINA NRA provides a set of generic concepts that describe the network resources providing the connectivity service. The NRA is a complex and broad architecture dealing with aspects such as connection, fault, accounting and network topology management. The network management aspects in TINA are addressed by its NRA, which has so far concentrated on Connection and Configuration management, with Accounting, Fault and Performance management aspects to follow.

The TINA Connection management was primarily developed to support stream-based, multimedia telecommunication services. The ATM technology was the primary source of influence in characterising technical features of the Transport Network.

A typical TINA Connection Management Architecture (transport network) consists of various layer networks each with their own characteristics. An ATM specific transport network, called the ATM Connection Management Architecture is shown in Figure 2.2 [2].

From the management point of view, the TINA NRA segments the network architecture into specialised management layers, adopted from the ITU-T TMN Recommendations [6], for the purpose of providing a connectivity service. The key management layers are the Service Management Layer (SML), Network Management Layer (NML), and the Element Management Layer (EML).

The ATM Connection Management Architecture consists of 5 computational objects, namely

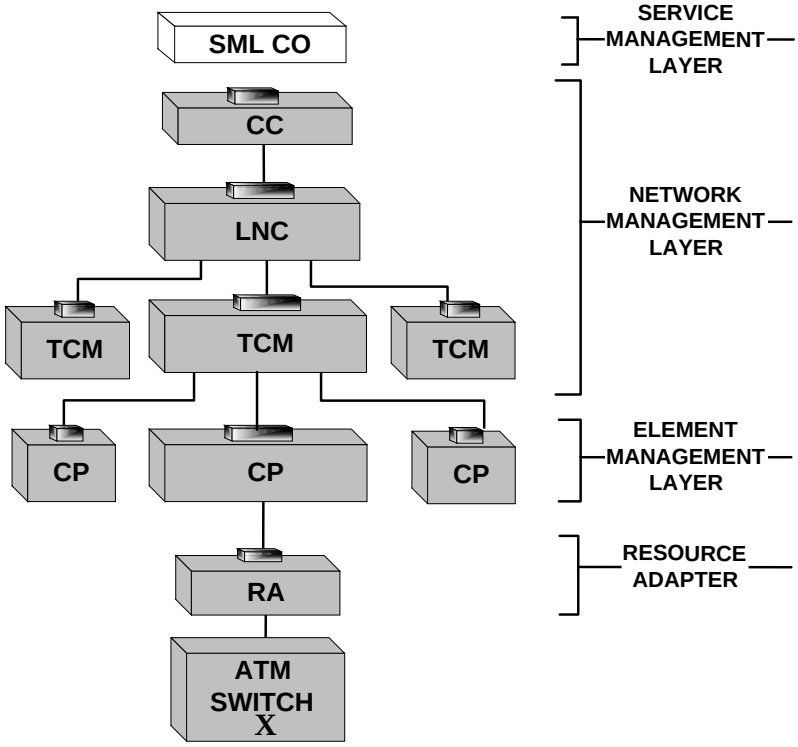

Figure 2.2: TINA ATM Connection Management Architecture

Connection Coordinator (CC), Layer Network Coordinator (LNC), Tandem Connection Manager (TCM), Connection Performers (CP) and the Resource Adapter (RA).

- **CC** required by the service components for establishing end-to-end Network Flow Connections.
- **LNC** controls a single layer network domain and is responsible for establishing end-to-end trail.
- **TCM** is created by the LNC of each layer network domain in which a connection must be created. A TCM is responsible for routing tandem connection through the layer network
- **CP** are responsible for establishing end-to-end sub-network connections within one sub-network. A CP controls a sub-network at the Network element level. CP are also used for mapping the request coming from the controlling TCM (with technology-independent concepts) into requests on the ATM Resource Adapter interface.
- **RA** acts as an intermediate between the EML-CP and the element it controls. The RA is dedicated to a given switch and is responsible for mapping ATM features onto switch-specific features.

These components reside on the data plane of the TINA NRA and are involved in the establishment of connection and the transmission of connection-related data within a singe layer network.

2.3 Fault Management in the TINA NRA

Within the context of TINA, fault management is related to the following areas [2]:

- within service management: recognize and handle faults related to telecommunication services.
- within network resource management: management of faults within NE's and communication facilities between them.

This project deals specifically with the management of faults in the context of network resource management.

2.3.1 Fault Management Activities

Fault management activities are performed through interactions between the fault management service user and the fault management functions. This section describes the fault management activities in terms of operations done within fault management functions and interactions between fault management functions and fault management service users. Depending on the case in which fault management is active, the following five different activities are identified [5]:

- **Alarm Surveillance:** includes collection and logging of alarm notification from the network resources.
- **Fault Localisation:** analyses the collected alarm information, detects the root cause of alarm, and notifies the result to the clients of the alarm surveillance.
- **Fault Correction:** restores and recovers the computational objects that represent the resources in which a root cause alarm is detected.
- **Testing Function:** invokes a test capability of a resource object upon a request from the clients of the service. It may also support a test of series of resource objects.
- **Trouble Administration:** enables the reporting of troubles due to fault conditions and the tracking of their status.

2.3.2 Functional Requirements of Fault Management

This section describes the functional requirements of Fault Management. Fault management provides a set of functions which enables the detection, isolation and correction of abnormal operation of the telecommunication network and its environment [2]. The scope of the functional requirements are for providing information and computational specification which conform to the TINA-C telecommunication management. The functional requirements of fault management are discussed below in terms of each fault management activity.

Alarm Surveillance

The functional requirements listed below, allow the fault management service to perform the alarm surveillance activity.

- Managed Objects (MO) must issue an alarm when a fault condition arises. Faults must be detected by fault management functions.
- The received alarm must be recorded for use by activities such as fault localisation, fault correction and alarm summary report.
- The detected faults must be filtered and reported to the fault manager.
- In the NML, the fault management functions must perform fault correlation and filtering activities. These activities use network information to remove redundant alarms and narrow the range of possible root causes of the fault.
- In the EML, the fault management functions must report alarms and provide an alarm summary. The alarms must be logged allowing alarm information to be retrieved for a NE. Alarm correlation and filtering must be performed in this layer as well.

Fault Localisation

The functional requirements listed below, allow the fault management service to perform the fault localisation functions.

- A fault within a network may results in multiple alarms from MOs. These alarms may be directly and indirectly related to the fault. The fault management functions must perform fault localisation capabilities, which determines the root cause of the fault.

- In the NML the fault management functions analyses filtered alarms and diagnostic test results to identify the root cause of the fault.
- In the EML the fault management functions must select and schedule tests and diagnostics to be performed on NE. If a root cause of the fault is found, the fault must be reported to the fault manager.

Fault Correction

The requirements of a fault management service to perform the fault correction functions are listed below.

- The fault management service is responsible for the repair of faults. This service must also control the procedures that use redundant resources to replace equipment or facilities that have failed.
- In the NML, the fault management service determines what analysis, testing or repair activity is required to be performed.
- In the EML, the fault management service isolates a unit with the fault. The service must also report on automatic restoration process carried out within the NE.

Testing/Diagnostic

The testing/diagnostic function in a fault management system is primarily used to perform tests and run diagnostic algorithms on NE or groups of NE.

- Testing is an activity within fault management functions, for verifying the functionality of resources in the system.
- A Diagnostic algorithm analyses the test results to determine root cause of the problem.
- In the NML, the testing function is used to perform systematic tests on each segment of a connection to determine which segment is faulty. The testing function receives requests to test specific part of a connection. An appropriate test suite is then selected and the test results are returned to the requestor.
- In the EML, the testing function controls the performance of test or a suite of tests. The testing function also reports test results and status information to the fault manager.

Trouble Administration

- A fault management service must provide an activity between manager and agent that enables troubles to be reported and their status tracked.
- The fault management service must coordinate actions to investigate and clear the troubles.

2.3.3 Computational Viewpoint of a TINA Fault Management Service

This section describes the computational viewpoint of the network resource fault management architecture. Figure 2.3 [2] shows the basic computational architecture for Fault Management. This architecture shows the computational objects (COs) identified by the TINA NRA for managing the network at various levels.

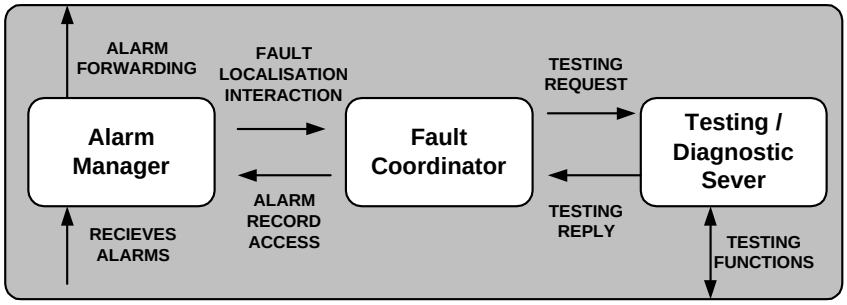

Figure 2.3: Computational Objects for Fault Management within a management layer

The Fault Management computational architecture consists of the following computational objects:

- *Alarm Manager (AM)*
  The AM receives fault-related alarms from MO's and performs relevant procedures for alarm correlation, alarm filtering, forwarding the alarm to the fault coordinator, the fault management service user and for the alarm record management.
- *Fault Coordinator (FC)*
  The FC includes capabilities to internally analyse the alarms received from the multiple MO's to determine the next possible step for fault localisation and correction. As a result, the FC correlates all information available to refine information concerning the root cause of the fault. During analysis, the FC invokes the Testing/Diagnostic Server to run tests on the managed objects.

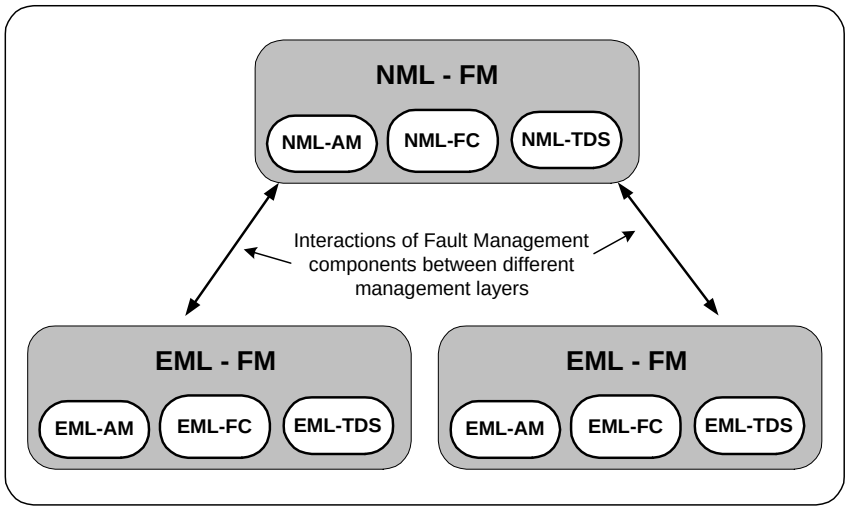

Figure 2.4: Basic Fault Management Computational Architecture using the basic structure shown in Figure 2.3 at each layer

- *Testing/Diagnostic Server (TDS)*
  The TDS is concerned with the testing of MOs for the purpose of service and function verification of MOs. From the fault management perspective, the TDS is invokes either my the FC, or fault management service user or by other CO in the system, such as the resource configuration and connection management objects.

The basic fault management architecture, as shown in figure 2.3, is applicable to both the bottom of the subnetwork (EML) and the top level subnetwork (NML) [2]. Figure 2.4 [2] shows the fault management architecture applied at various management layers. The hierarchial arrangement of this fault management architecture plays both a manager role (at the NML) and an agent role (at the EML).

Figure 2.5 shows the control plane of the TINA NRA with the fault management components. These components are used for the transmission of control information within a layer domain. Since the TINA Management Architecture uses the TMN Functional Layering hierarchy, the FM COs are integrated into the Network and Element Management Layers of the transport layer. The FM architecture uses the same layering concept as the connection management architecture and places the Fault Management Components in the same positions where the CO classes are placed in the transport network.

2.3.4 Functionality of the Computational Objects

The functionality of each of the computational objects presented in Section 2.3.3 is discussed from both a NML as well as an EML perspective. The structure and functions of the

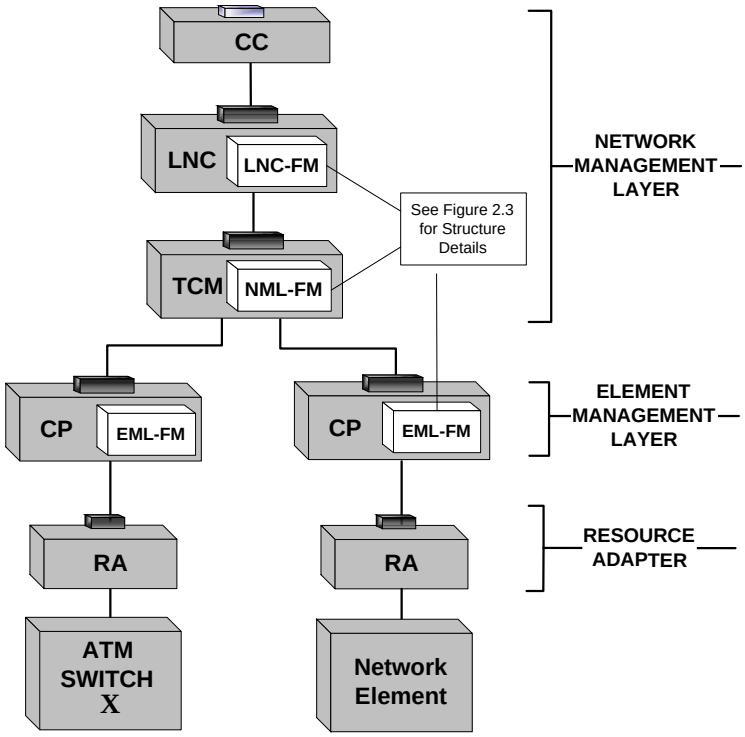

Figure 2.5: Control Plane of the TINA NRA incorporating the basic fault management architecture at each layer

fault management CO in the EML and the NML are similar. Figure 2.6 uses the computational objects shown in Figure 2.3 to show a detailed fault management architecture that is incorporated at various management layers. Figure 2.6 also shows the interactions that occur between the computational objects of the fault management architecture.

Alarm Manager

Figure 2.6 shows the functional architecture for both the EML-FM and NML-AM, i.e. parts **A** and **B** respectively .

The MO generates an alarm event, which is first received by EML Alarm Manager (ELM-AM). The EML-AM first makes the event log record and filter the alarms. It then prepares the alarm record for further alarm analysis and reporting purposes. The filtered alarm passes the alarm correlation function, which removes redundant alarms and initiates fault localisation procedure. The alarm is passed to the EML Fault Coordinator (EML-FC) to isolate the fault. The alarm is finally passed to NML Alarm Manager (NML-AM) through alarm forwarding functions. The alarm reports from an EML-AM may or may not specify the root cause of the alarm. As with the EML case, the NML-AM interacts with the NML-FC to identify the root cause of alarms and forwards alarm records to users. NML-AM makes use

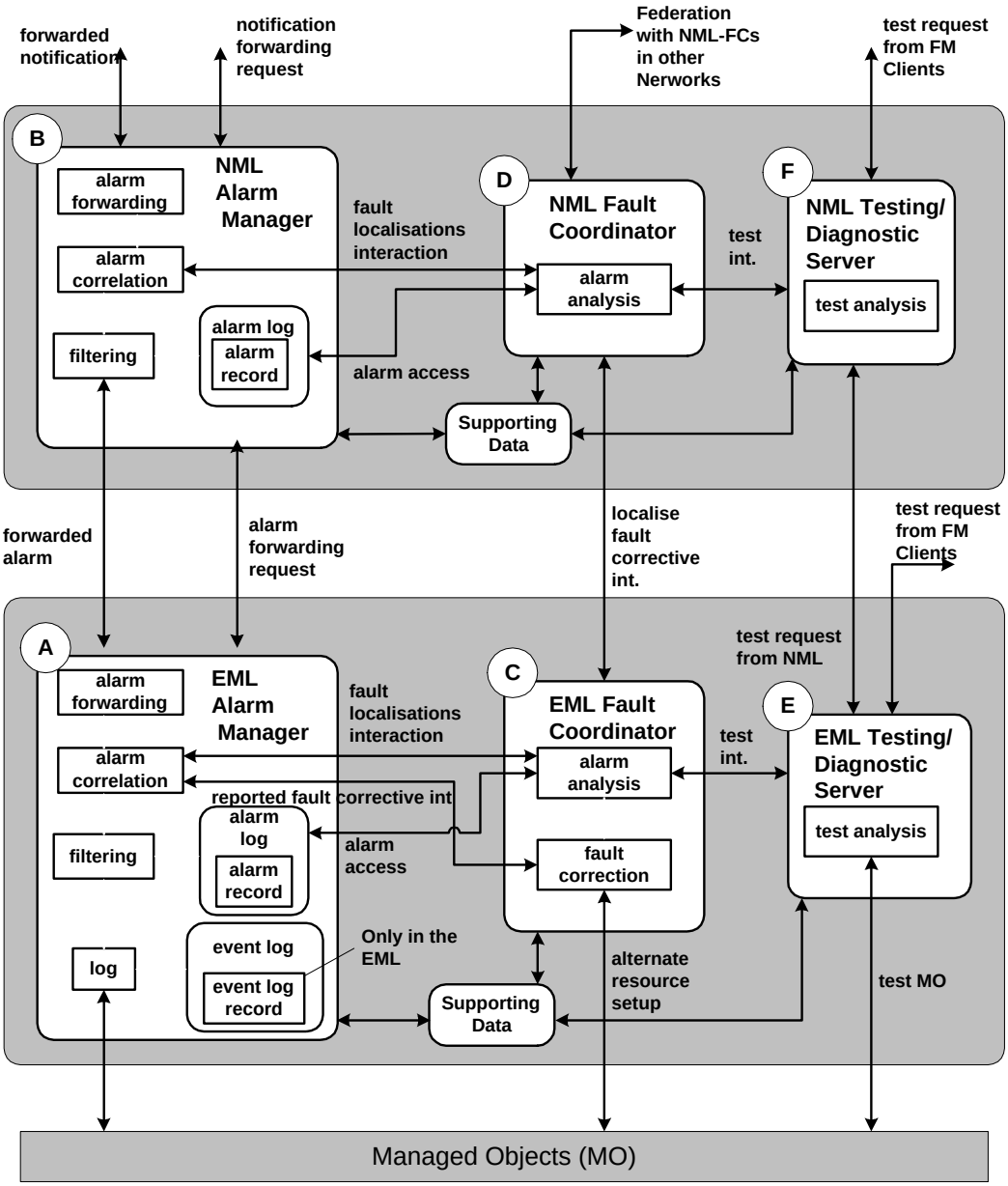

Figure 2.6: Fault Management Functional Architecture showing details of Element and Network Management Layers

of connection topology information between subnetworks as supporting data. Functions of EML and NML Alarm Manager:

- *Log Functions:* This function is applicable only to the EML-AM. The log function receives event alarms from the MO and prepares an event log record if the alarm satisfies the criteria defined in the log. It then passes the alarm to the alarm filtering function.

- *Alarm Filtering Functions:* Performs alarm filtering that checks local data to determine whether reporting of such events as alarms are inhibited. Forwards the alarms to the alarm correlation function.

- *Alarm Correlation Functions:* Performs alarm correlation analysis using equipment hierarchy and connection topology as supporting data. Duplicated alarms are discarded. In EML-AM, the correlation functions interact with EML-FC for fault localisation to determine the root cause of a set of related alarms. The fault localisation results are then reported to the NML-AM. The EML-AM also forwards alarms to the EML-FC. In NML-AM, the alarm is forwarded to the NML-FC after the correlation process has taken place.

Fault Coordinator

The functions of the Fault Coordinator are alarm analysis and fault correction. The NML-FC only performs alarm analysis. Figure 2.6 shows the Fault Coordinator functionality in both the EML and the NML, i.e. parts **C** and **D** respectively.

- *Alarm Analysis Functions:* activated by a request from the EML-AM or the NML-AM. This function uses information about the equipment hierarchy, connection topology and current alarm records to determine the root cause of the set of related alarms. In the case of locating faults that span multiple networks, federation is used for the cooperation of other NML-FCs.

- *Alarm Correction Functions:* is activated by the EML-AM or the NML-FC when a resource is identified to be faulty. The EML-FC performs automatic restoration by activating backup resources.

Testing/Diagnostic Server

The EML-TDS and the NML-TDS provides capabilities for performing tests and diagnostic on a set of resources. Testing is a simple function verification of a set of resources. Diagnostics includes the analysis of the test results to determine the main cause of the abnormal behaviour. The function of the Testing/Diagnostic Server is test analysis function. Figure 2.6 on page 15 shows the Testing/Diagnostic Server's functionality in both the EML and the NML, i.e. parts **E** and **F** respectively.

Test Analysis Functions: Performs testing on a set of resources which are predefined or specified at the time of invocation. The diagnostic results are sent to the client or fault management service user.

2.4 Chapter Summary

This chapter presents an approach in which distributed transport networks can be managed using the TINA Network Resource Architecture and the TINA Management Architecture. The TINA NRA and Management Architecture together offers a technology-independent abstraction of network management functionality. This technology-independent abstraction is used to design this heterogeneous fault management service. In particular this chapter concentrates on the Fault Management architecture, the fault management components and concepts from a TINA viewpoint. TINA adopts the specialised management layering hierarchy as defined by the TMN recommendations, which highlights the integration of TMN into the TINA environment.

This chapter also describes the activity that a typical fault management system must perform and presents the functional requirements needed for each fault management activity. A computational viewpoint of a TINA fault management service is presented showing the three main computational objects, namely, Alarm Manager, Fault Coordinator and Testing/-Diagnostic Server. These computational objects interact with each other to perform these fault management activities.

The concepts, specification and component structure described in this chapter facilitates the proposed design and implementation of a Fault Management Service which is discussed in subsequent chapters.

Chapter 3

Design of a Distributed Network Fault Management Service (FMS)

This chapter reviews the methodology used in designing the distributed network Fault Management Service. The reader is presented with detailed concepts, components and elements used in this design.

Chapter 2 presents a network fault management architecture as defined by the TINA NRA. Figure 2.5, shows the control plane of the TINA Connection Management Architecture with the fault management components, hereafter referred to as the Fault Management Architecture (FMA). The proposed FMS design is based on the FMA. The FMA, shown in Figure 3.1, is used as an outline for the proposed fault management service architecture. Fault Management software components will replace the components in the network management layer, element management layer and the resource adapter layer.

This chapter presents the functional requirements for the proposed fault management service. Since current network elements have SNMP (Simple Network Management Protocol) interfaces, the designed is constrained to use SNMP to access information from the managed objects. The use of SNMP in the design of the FMS is discussed. CORBA (Common Object Request Broker Architecture) is used to provide location and implementation transparency for the distributed FMS. The use of CORBA in the design of the FMS is presented in this chapter. The functionality of the proposed FMS is described using Unified Modeling Language (UML). Finally, the components, together with their various IDL interfaces, that performs the required fault management function is presented in detail.

3.1 Functional Requirements of an FMS

The fault management activities defined in section 2.3.1, provides a foundation on which the fault management functional requirements are defined. The only activity not defined in

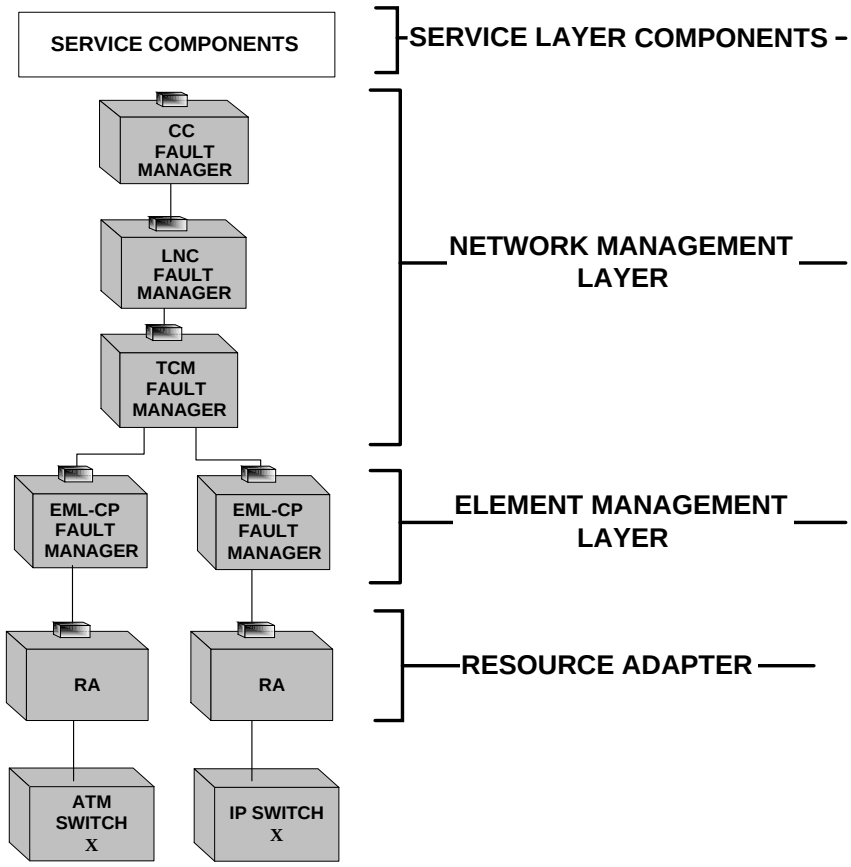

Figure 3.1: Fault Management Architecture: Control Plane of the TINA NRA

this design is the Fault Correction activity. The functional requirements pertaining to this design are defined for each activity listed in section 2.3.1.

3.1.1 Alarm Surveillance

1. Faults must be detected using alarms received from the MO.
2. Fault alarms must be filtered to remove redundant alarm information and then logged in a database.
3. The filtered alarms are reported to the Fault Manager or Network Administrator.
4. Filtered alarms are also forwarded to component that handles fault localisation activity.

3.1.2 Fault Localisation

1. The fault alarm information received from the Alarm Surveillance activity is analysed to localise the fault.

2. To isolate the faults, the fault management service queries a database containing equipment topology information.

3. This activity invokes test capabilities on the faulted managed objects, to diagnose the reason for the fault condition.

4. All localised alarm information is reported to a fault manager's console.

3.1.3 Testing Function

1. The sole activity for this function is to test faulted managed objects.

2. Examples of test capabilities used are the PING test and TRACEROUTE test.

3. Test results are sent to the Fault Manager or Network Administrator.

3.1.4 Trouble Administration

1. SNMP is used to capture trap information received from the network elements.

2. This activity involves real-time notification of fault conditions to the Alarm Surveillance activity.

3. All alarm information reports to a fault manager's console.

3.2 Design Consideration

Before designing the Fault Management Service, several design considerations had to be taken into account. These considerations include the use of SNMP to access network elements, the need to develop a scalable fault management service, the need to develop a fault management service that manages heterogeneous networks and the need to design reusable components . These considerations are presented below.

3.2.1 Simple Network Management Protocol (SNMP) - A Design Constraint

To gain knowledge of the managed object's status, a management protocol is required. Since most network elements have SNMP interfaces, we are constrained to use Simple Network Management Protocol (SNMP) to access the managed objects.

SNMP is a communication protocol that has gained widespread acceptance since 1993 as a method of managing TCP/IP networks, including individual network devices, and devices

in aggregate. SNMP was developed by the IETF (Internet Engineering Task Force), and is applicable to any TCP/IP network, as well as other types of networks. The advantages of using SNMP is that it is simple to use, interoperable, easy to implement and consumes minimal processor and network resources. SNMP has become the most widely-used network management tool for TCP/IP based networks [7]. SNMP is thus a tool for building bare-bones network management facility [8].

SNMP defines a client/server relationship. The client program (called the network manager) makes virtual connections to a server program (called the SNMP agent) which executes on a remote network device, and serves information to the manager regarding the device's status. The database, controlled by the SNMP agent, is referred to as the SNMP Management Information Base (MIB), and is a standard set of statistical and control values. SNMP additionally allows the extension of these standard values with values specific to a particular agent through the use of private MIBs. Through the use of private MIB variables, SNMP agents can be tailored for a myriad of specific devices, such as network bridges, gateways, and routers.

In essence, the SNMP protocol provides four functions [8]:

- **Get Request:** Specific values can be fetched from the network element via the "get" request to determine the performance and state of the device. Typically, many different values and parameters can be determined via SNMP without the overhead associated with logging into the device, or establishing a TCP connection with the device.
- **Get Next Request:** The SNMP standard allows network managers to "walk" through all SNMP values of a device (via the "get-next" request) to determine all names and values that a device supports. The "walk-through" feature is accomplished by beginning with the first SNMP object to be fetched, fetching the next name with a "get-next", and repeating this operation until an error is encountered (indicating that all MIB object names have been "walked".)
- **Set Request:** The SNMP standard provides a method of effecting an action associated with a device (via the "set" request) to accomplish activities such as disabling interfaces, disconnecting users, clearing registers, etc. The "set request" provides a way of configuring and controlling network devices via SNMP.
- **Trap Message:**The SNMP standard furnishes a mechanism by which devices can alert the network manager on their own (via the "trap" message) to notify the manager of a problem with the device. The trap function typically requires each device on the network to be configured to issue SNMP traps to one or more network devices that are awaiting these traps.

For this design, SNMP was primarily used for receiving Trap messages from the managed network elements (ATM switches, IP switches and routers). The limitation of using the Trap messages facility is that the manager's network address had to be pre-configured into the SNMP agents, so that the agents can issue traps to them.

3.2.2 The need for a Scalable Fault Management System

The rapid growth in size of networks has brought into question the scalability of fault management systems. If the size (number of devices), complexity (number of manageable variables), or the speed of the network increases, the management system becomes unmanageable. The proposed distributed fault management system, needs to be scalable to accommodate the growth of distributed networks.

The proposed FMS design adopts some of the properties of the centralised management paradigm. A platform-centred paradigm is characterised by agents that monitor the system and collect data, which is then accessed by applications via management protocols [9]. However, the platform-centred paradigm has limitations in that makes it non-scalable. As the number of managed network elements increases, the network load on the centralised manager increases proportionally making management of such networks difficult.

The FMS design uses the TINA management architecture and CORBA to resolve the scalability issue. The FMS design uses the hierarchial arrangement of the TINA Management Architecture to distribute intelligence thus decreasing the network load on the centralised manager. The proposed FMS, being hierarchial (and based on CORBA) should be scalable.

3.2.3 Development of reusable software components

The design of the proposed Fault Management Service must incorporate the design of reusable software components. This project focuses on the design and implementation of a fault management service in the Element Management Layer. However, the TINA NRA has defined the same fault management computational objects to be used in both the NML and the EML (see Figure 2.4). The same fault management software components used in the EML, can be used in the NML as well. These reusable software components ensures rapid deployment of the FMS to upper management layers.

3.2.4 Implementation across heterogeneous networks

The TINA NRA provides a technology-independent abstraction of potentially heterogeneous underlying networks. This means that the FMS must be capable of managing heterogeneous networks. Hence, a generic Fault Management Service must be designed and developed. Since the SATINA Trial platform consists of both an ATM network as well as an IP MPLS network, the heterogeneous functionality of the FMS will be tested on this platform.

3.3 DPE Environment

The heterogeneous nature of the distributed FMS requires an implementation environment to provide location and implementation transparency. An implementation of a DPE node that provides this transparency is *Common Object Request Broker Architecture* (CORBA) [10].

3.3.1 CORBA

CORBA defines the programming interfaces to the Object Management Architecture (OMA) ORB (Object Request Broker) component. An ORB is the basic mechanism by which objects transparently make requests to, and receive responses from, each other on the same machine or across a network. A client need not be aware of the mechanisms used to communicate with or activate an object. The client also need to be aware of how an object is implemented nor where the object is located. The ORB thus forms the foundation for building applications constructed from distributed objects and for interoperability between applications in both homogeneous and heterogeneous environments. The OMG Interface Definition Language (IDL) provides a standardized way to define the interfaces to CORBA objects. The IDL definition is the contract between the implementor of an object and the client. IDL is a strongly typed declarative language that is programming language-independent. Language mappings enable objects to be implemented and sent requests in the developer's programming language of choice in a style that is natural to that language.

In this design, CORBA is used to provide a communication bus for messages between the fault management activities. In figure 3.1, all communication from the resource adapter (RA) is DPE-based.

3.3.2 Notification Service

The traps issued by the SNMP agent needs to be distributed into the DPE environment. The OMG CORBA-based Notification Service is used for the distributing the fault alarms into the distributed fault management environment.

The CORBA-based Notification Service as described in TINA Architecture is a DPE service that enables objects to emit notifications without being aware of the set of recipient objects [11]. Similarly, the notification service enables an object to receive notifications from one or more objects without having to interact with these objects explicitly and individually. Thus, the Notification Service acts as a broker between emitters and recipients of notification. The notification channel supports both the push and pull model communications. Figure 3.2 shows a diagram of the notification service operating in the push mode.

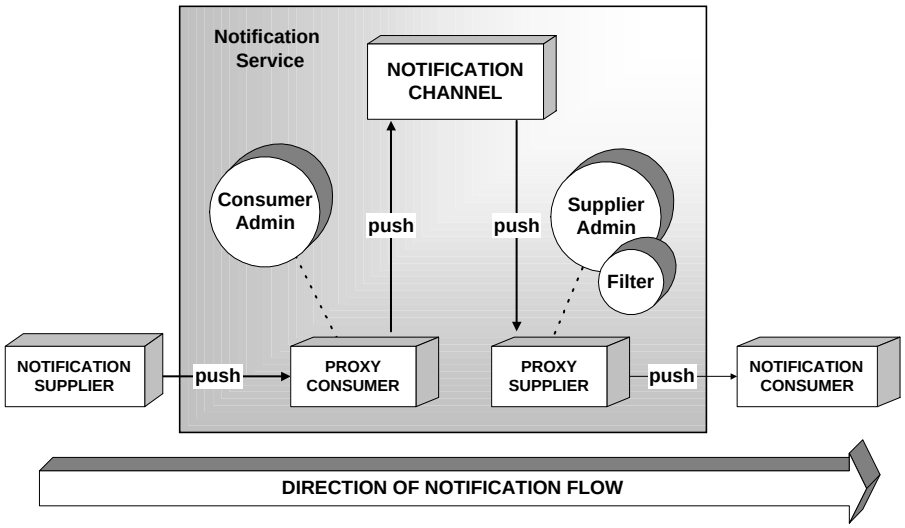

Figure 3.2: CORBA-based Notification Service Components

The CORBA-based notification service extends the existing OMG Event Service, adding to the following new capabilities to the event service [12]:

- The ability to transmit events in the form of a well-defined data structure, in addition to Anys and Typed-events supported by the existing Event Service.
- The ability for clients to specify the events they are interested in receiving, by attaching filters to each proxy in a channel.
- The ability for the event types required by all consumers of a channel to be discovered by suppliers of that channel, so that suppliers can produce events on demand, or avoid transmitting events in which no consumers have interest.

- The ability for the event types offered by suppliers to an event channel to be discovered by consumers of that channel so that consumers may subscribe to new event types as they become available.
- The ability to configure various quality of service properties on a per-channel, per-proxy, or per-event basis.

For this project, the Push-style Notification delivery model is used. With the push model, suppliers push events to the notification channel, and the channel pushes events to the consumers. In this case the suppliers are active initiators of events whereas the consumers passively wait to receive them.

Figure 3.2 shows the Notification Service components that creates the Administrator objects that allows the clients to set administrative properties. The admin interfaces are essentially a factory that creates the proxy interfaces to which clients (supplier or consumer) ultimately connect. The Consumer Admin object is used by the consumer to subscribe to only those events required by the Fault Manager CO and discards the rest. The Supplier Admin object is used by the supplier to publish the types of notifications it's sending through the notification channel. Figure 3.2 shows the Consumer and Supplier Admin objects.

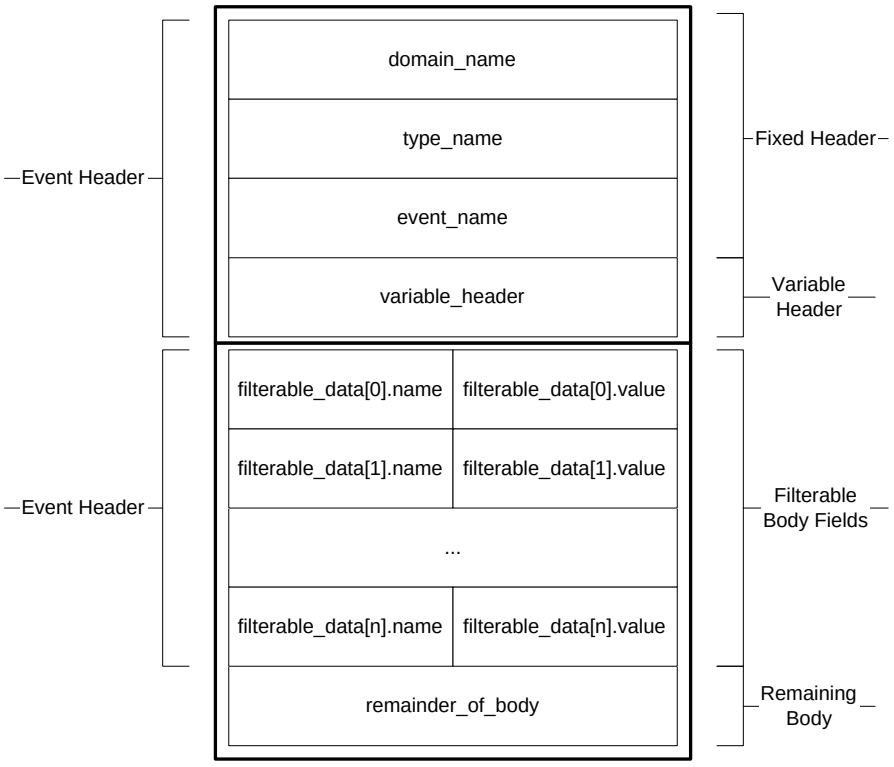

Figure 3.3: The structure of a Structured Event

The Notification Service supports event filtering on three fundamental types of events: untyped events contained within a CORBA Any, typed events as defined by the OMG Event Service, and structured events, defined in the CORBA notification specification [12]. Structured events define a well-known data structure to which many different types of events can be mapped. Figure 3.3 shows the structure of a Structured Events [12] used in this design.

Each structured event comprises of two main components: a header and a body. The header is further decomposed into a fixed portion and a variable portion. The goal of this decomposition is to minimize the size of the header which is required in every Structured Event message, thus enabling lightweight messages [12].

3.4 The Modified Fault Management Architecture

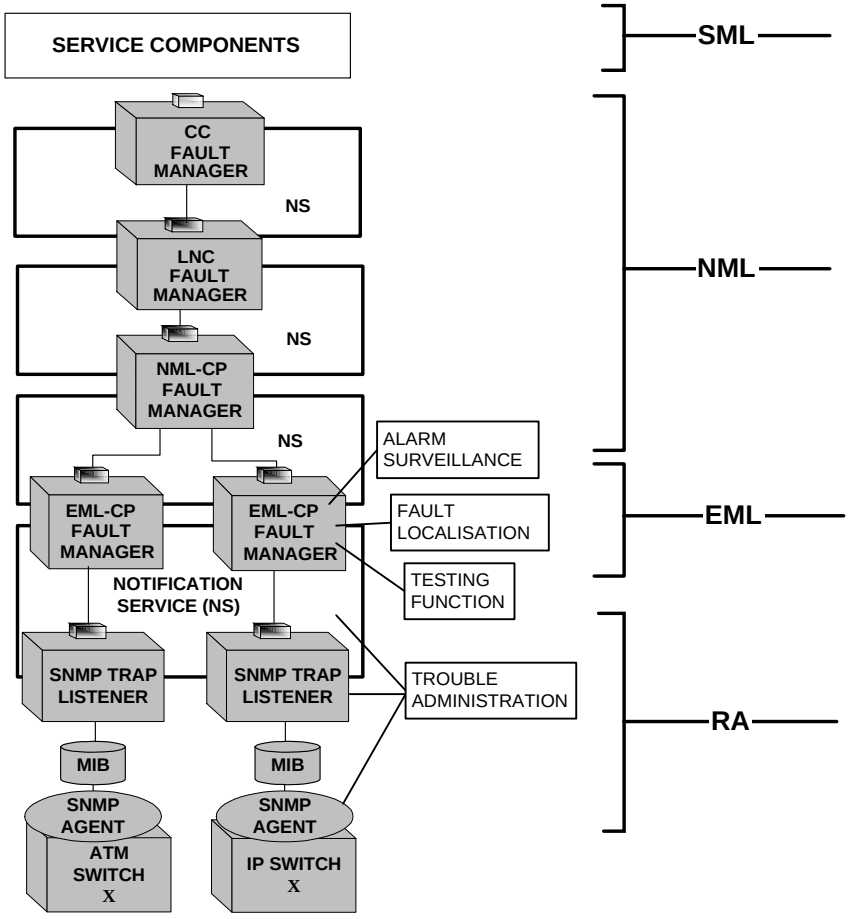

Figure 3.4: Modified Fault Management Architecture showing the allocations of Functional Activity Components to the EML and RA layer

The FMS design uses SNMP to access managed objects, uses CORBA to deal with location and implementation transparency and uses the notification service for the notification

of faults. The Fault Management Architecture presented in figure 3.1 is therefore modified to include these considerations. Figure 3.4 shows the modified version of the Fault Management Architecture.

As shown in Figure 3.4, the fault management functional activities reviewed in section 2.3.1, namely the Alarm Surveillance, Fault Localisation, Testing Function are carried out by the EML-FM component. These activities are also performed by the upper layer components, i.e the NML-FM. The two components that perform the Trouble Administration activity are the Notification Service component and the Resource Adapter component. The resource adapter component consists of an SNMP Agent, a MIB and an SNMP Trap Listener. The Notification Service is therefore positioned between the SNMP Trap Listener and the EML-FM to allow for the real-time reporting or notifying of faults to the management system. The notification service also spans adjacent management layers, as shown in Figure 3.4. In this case, the supplier of events are played by the components situated in the lower management layers and the consumer of events are played by components situated in the upper management layers.

A visual modelling technique is used to define the components that will perform the requirements for each fault management functional activity. For this project, the Unified Modelling Language (UML) is used for defining the fault management components.

3.5 A Use Case View of the Fault Management Service

A *Use Case View* models the functionality of the system as perceived by outside users, called ACTORS. A USE CASE is a coherent unit of functionality expressed as a transaction among actors and the system. The purpose of the use case view is to list the actors and the use cases and show which actors participate with each use case [13].

For this project, seven use case scenarios are presented. Each use case describes a particular functionality of the fault management service. To interact with these use cases, five actors have been identified. Table 3.1, shows all the uses cases and actors selected for this design. Each use case is discussed individually in terms of functionality and interaction with the actors. Figure 3.5 shows the use case diagram of the proposed Fault Management Service.

3.5.1 Use Case: Catch TRAP message

As a precondition to this use case, the network elements have to be configured to issue SNMP Traps to a fault manager's terminal. When an abnormal behavior occurs, the trap

UML	Fault Management Service
Actors	Network Element
	Network Administrator
	Alarm Records
	Network Setup Database
	Contact Information
Use Cases	Catch Trap message
	Enter alarm information in Alarm records
	Localise Fault alarms
	Select Test method
	Run Test method
	Contact network technician
	Update alarm records

Table 3.1: Fault Management Use Cases and Actors

messages are forwarded to the network device awaiting these traps. The network administrator needs to be notified immediately of such occurrence in the network. Once the traps are forwarded to the network manager, the alarms undergoes a filtration process where the redundant alarms are removed. After the filtration process, the alarms must be prepared to be logged into an alarm database for reference purposes. The filtered alarms are sent to the use case that enters the alarm information into an Alarm Record and to the use case that localises the fault.

3.5.2 Use Case: Enter Alarm information into Alarm Records

The filtered alarms sent from the *Catch Trap* use case is then logged into an alarm record for future references. The information entered into this database includes information obtained from the network element's SNMP agent. Typically this includes:

- Date and time of alarm
- Name or IP Address of network device that catches trap messages
- Name and IP Address of network element that issues the trap message
- Port number on which the network device listens for trap message, etc.
- The trap type, name severity, etc.
- Fault status, i.e. whether fault is activated or deactivated.

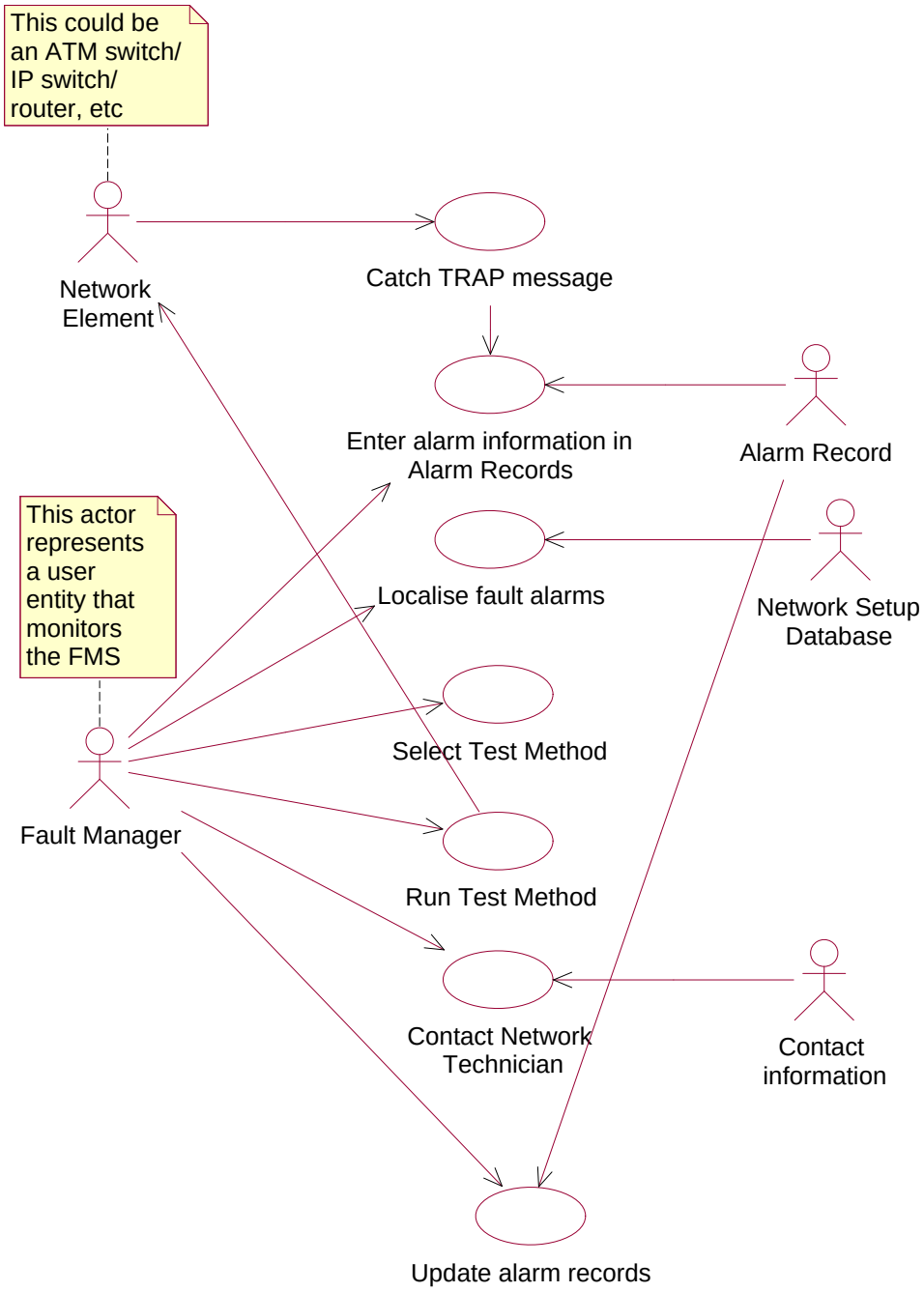

Figure 3.5: Use Case Diagram for the Fault Management Service

- Possible fault solutions.

The fault manager can also select the fault alarm from this database to be corrected.

3.5.3 Use Case: Localise Fault

The filtered alarms from sent from the *Catch Trap* undergoes a fault localisation procedure, where the fault manager tries to isolate the fault. For this procedure to take place, the fault management system must know the equipment topology. With this "knowledge" fault localisation is made possible. This use case also uses the *Run Test Method* use case to help determine the root case of the fault.

3.5.4 Use Case: Select Test Method

An appropriate test method is selected by the fault manager to assist in isolating the fault. Fault test methods are outside the scope of this work. Evaluating methods, Ping and Traceroute tests are used for demonstration. The network element for which the Test Method is selected must first be selected.

3.5.5 Use Case: Run Test Method

After the test method has been selected, some test methods may require certain parameters to be entered by the user. Parameters may include Remote Host address, required fields to be return with results, maximum ping time, etc. The fault manager then runs the selected tests. The results of the test is returned to the fault manager for analysis purposes in order to help determine the root cause of the fault.

3.5.6 Use Case: Contact Network Technician

Since no automatic fault correction is designed for this management system, faults have to be corrected by the network technician. This use case utilizes the contact information database to get contact information of the network technician to correct the fault.

3.5.7 Use Case: Update Alarm Records

Once the fault had been corrected by the network technicians, the alarm records are updated. By updating the alarm records, the fault manager changes the status of the alarm from being activated to deactivated. The network administrator also includes information regarding the cause of the fault and how the fault was corrected. The network administrator has the option of deleting alarm entries from the alarm record database.

3.5.8 Review of Use Case Scenarios

The use case scenarios discussed can be grouped together to perform fault management functional activities. Table 3.2 shows the use cases grouped to perform a specific fault management activity.

Functional Activity	**Use Case**
Alarm Surveillance	Catch Trap message Enter alarm information in alarm records Update alarm records
Fault Localisation	Localise fault alarms Contact Network Technician
Testing Function	Select Test method Run Test method

Table 3.2: Fault Management Use Cases and Actors

The following remarks can be made about Table 3.2:

- Three main fault management objects defined in section 2.3.3 perform the three management activities listed in table 3.2 . They are the Alarm Manager, Fault Coordinator and the Testing Server. The computational objects that are developed to perform these use cases are discussed in detail in section 3.6.
- These objects conform to the fault management objects defined in the TINA NRA [2] and reviewed in Section 2.3.3.
- No use cases are defined for the the Trouble Administration activity. This activity is performed by the following components:
  - SNMP Agent and MIB
  - Notification Service
  - SNMP Trap Listener

  These components are discussed in more detail in the section 3.6.1.

3.6 Component View of the Fault Management Service

The component diagram, in Figure 3.6, provide a physical view of the Fault Management Service. The component diagram shows the organizations and dependencies among software components of the FMS. Section 3.5 discusses the functionality of each component,

the Alarm Manager, Fault Coordinator, Testing Function and Notification Service, in terms of use case scenarios.

In this section, the reader is presented with the software components that perform these functional requirements. Each of these software components is discussed in terms of a Client Role and a Server Role it serves. The components is grouped in terms of the functional activity they perform. The components described hereafter, can be seen in Figure 3.6.

3.6.1 Components for Trouble Administration Activity

The components that are responsible for the reporting of faults to the management system are the Trap Listener and the notification service.

Trap Listener

This software object serves two main functions. The first, is to act as a Trap listener and the second is to act as supplier of notifications to the notification channel.

In the first instance, this component uses the "AdventNet SNMP API" [14] to listen for trap alarms originating from the managed objects. The AdventNet SNMP API is a set of Java libraries (API) used for creating cross platform Java and Web-based SNMP network management applets and applications. The package can be used to develop SNMP management applications and applets to manage SNMPv1, SNMPv2c and SNMPv3 agents. The AdventNet SNMP API talks to agent systems using any of the three versions of the SNMP protocols at the same time. The AdventNet SNMP API includes Java classes that implement [14]:

- SNMP communication for SNMPv1, SNMPv2c and SNMPv3 protocols.
- MIB support for both SMIv1 and SMIv2 formats so that Java management applications can take advantage of the information contained in the MIB files.
- SNMP Beans Components that provide enhanced functionality and for use in Java IDE tools. The Java beans components simplify management application development with SNMP-aware user interface components.
- RMI and CORBA access to SNMP API for distributed computing support. Command line tools like snmpget, snmpgetnext, snmpset, sendtrap etc.

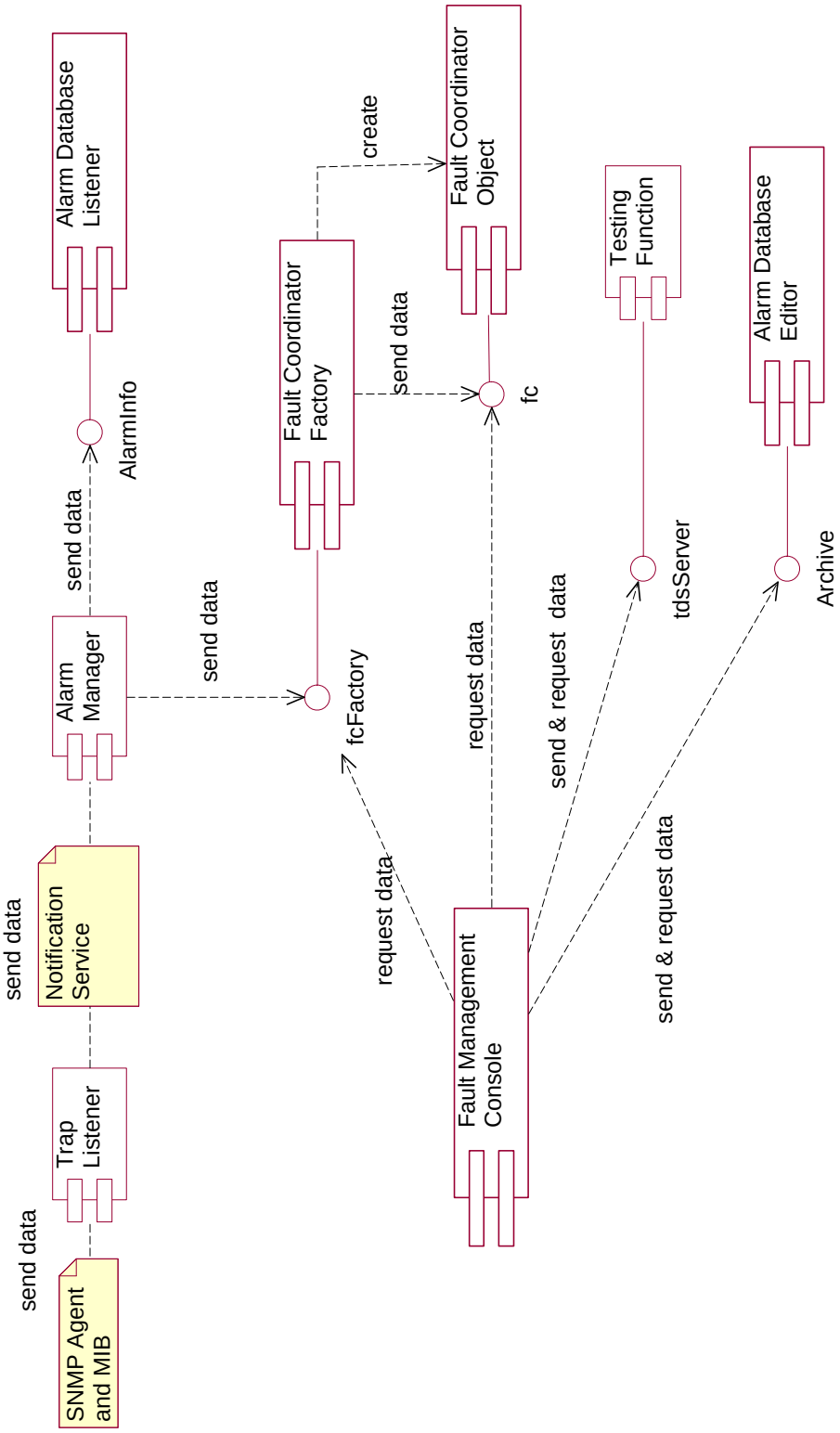

Figure 3.6: Component View of the Fault Management Service

For the Trap Listener component, the AdventNet SNMP Beans Component is used to listen for traps. The type of trap data received, includes:

- **Date and time of Alarm**
- **Source** - the host listening for traps
- **Remote Host** - the address of the device being managed
- **Port** - management applications receive trap messages in UDP port 162.
- **Community** - SNMP protocol mandates that the SNMP agents should accept request messages only if the community string in the message matches its community name. So the management application should always communicate with the agents along with the associated community name.
- **Agent Address** - this is the address of the SNMP agent which resides in the network element
- **Enterprise OID** - The enterprise field is the SNMP enterprise identifier in the trap, which is used to uniquely identify traps for a particular application.
- **Trap Variable OID** - the name of the variable in the MIB file.

The second function of the Trap Listener is to supply the trap data as notification events to the notification channel. To supply the trap data as notifications, the trap data is packaged into structured event. This structured event is sent to the notification channel. Figure 3.3 shows the structure of the structured event. The trap data forms part of the filterable body fields of the structured event. The Trap Listener then connects to the notification channel and *pushes* the event into the notification channel.

3.6.2 Components for Alarm Surveillance Activity

These components are responsible for receiving the trap notification from the notification channel, filtering this information, logging the filtered information and forwarding it to the Fault Localisation Activity.

Alarm Manager (AM)

This component behaves as a consumer of notifications as well as client to a server.

- Consumer Role

  The AM receives events from the notification channel sent by the supplier, i.e the Trap Listener. When a fault occurs, the SNMP agent issues numerous identical alarms to the Trap Listener. To remove these redundant alarms, a **LinkedList** component is used to filter these alarms before being logged into the database. The **LinkedList** component compares the trap data, such as Source, Port number, Community, Agent Address, Enterprise OID and Trap Variable OID . If the incoming alarm has the same alarm information as the alarms in the **LinkedList**, the incoming alarm information is then deleted.

- Client Role

  The AM behaves as a client to the:

  - **Alarm Database Listener** component that logs the filtered alarm information into a database. The AM uses the interface `AlarmInfo` to send the filtered alarm information to the Alarm Database Listener component.
  - **Fault Coordinator Factory** component that performs the Fault Localisation Activity. The AM uses the interface `fcFactory` to send the same alarm information, as sent to the Alarm Database Listener, to the Fault Coordinator Factory component.

Appendix A.1 and Appendix A.2 provide the full definition of `AlarmInfo` and `fcFactory` interfaces respectively.

Alarm Database Listener

The Alarm Database Listener component is primarily used for logging the alarm information received from the AM client into a database. The database management system used for this project is *MySQL (version 3.23.32)* database [15]. The databases used for this project are not complex and are not required to be stored as objects. Hence a simple relational database is used. A relational database stores data in separate tables rather than putting all the data in one big storeroom.

The following trap data is logged into this database: Date and Time of Alarm, Source, Remote Host, Port number, Community, Agent Address, Enterprise OID and Trap Variable OID.

Alarm Database Editor

The Alarm Database Editor component is used for viewing and updating the Alarm database. The alarm database editor interfaces to the Fault Management Console via the `Archive` interface. This interface is defined in Appendix A.5. The Management Console invokes the `showDB()` method to view the contents of the database where all the trap alarms are logged. The Console has the ability to update a logged alarm entry by supplying the the fault status and a solution to the fault via the `updateDB()` method. An alarm entry can also be deleted by invoking the `DeleteDB()` method.

3.6.3 Components for Fault Localisation Activity

The Fault Localisation Activity components are responsible for isolating the fault. These components interacts with a *MySQL* database to help isolate the fault. The fault localisation result is then sent to the **Fault Management Console**.

Fault Coordinator Factory (FC factory)

- Server Role
  The interface to this component is the `fcFactory` interface defined in Appendix A.2. Through this interface, the Fault Coordinator Factory receives the filtered trap data from the AM component via the `send_alarmInfo()}` method. Once the trap data is received, the fault coordinator factory creates a **Fault Coordinator Object (FC object)** i.e. for each new alarm, a new FC object is created by the FC Factory. For each new FC object created, the FC factory assigns a unique Fault Identification number (FaultID). The FC factory then stores this FaultID, with the Date and time of the alarm (received from AM) and an object reference of the newly created FC object into a list. This list can then be requested, at any time, by the Fault Management Console via the `get_faultList()` method of the `fcFactory` interface. The Fault Management Console can also select a FaultID to be deleted by invoking the `send_delete()` method. Upon receipt of the FaultID to be deleted, the FC factory deletes the FC object associated with this FaultID.

- Client Role
  The FC factory plays a client role to the FC object. Through the `send_alarmInfo()` method of the `fc` interface, the FC factory sends the trap data, received from the AM, to the FC object. The `fc` interface is defined in section A.3.

Fault Coordinator Object (FC object)

The FC object component is responsible for providing detailed information regarding the fault. The FC object component serves as a server to both the FC factory and the Fault Management Console.

The trap data is forwarded from the FC factory component via the `send_alarmInfo()` method of the `fc` interface. The Fault Coordinator object awaits an invocation of the `get_faultDetails()` method (of the `fc` interface) from the Fault Management Console. Once the invocation is made, the FC object queries the Alarm database to see if such alarms had occurred previously. If a fault with similar information fields had occurred previously and the fault was rectified, then the database will contain information regarding the possible solution to that fault. The FC object will then provide the Fault Management Console with this solution. This component then analyses the trap variable OID of the incoming alarm with that of the list of trap variable OIDs. Upon analysis, the appropriate fault management action is performed. For an example, when a "linkdown" failure is reported, the trap variable OID associated with fault alarm information is then compared to the trap variable OID's in the FC object. If there is a match, then the FC object knows what action to take to localise the fault.

The FC object then uses the trap information to query a *MySQL* database to try and localise the fault. This database contains information regarding the equipment topology of the network being managed. This information is requested when the Fault Coordinator object's `get_linkDetails()` method is invoked. The FC object also queries another database that contains contact information of the network technician in charge of fault corrections. The contact information is then requested when the Fault Management Console invokes the `get_contactInfo()` method of the `fc` interface. The `fc` interface is defined in Appendix A.3.

3.6.4 Component for Testing Function Activity

Testing Server

The Testing Server component is responsible for testing managed objects. For this design, only a PING test method was implemented. An interface to this component is the `tdsServer` which provides the Testing Server with an IP address to PING. This IP address is set by the user of the Fault Management Console. The user of the Fault Management Console then awaits the results of the Ping test. Appendix A.4 shows the full definition of the `tdsServer` interface.

Developing testing and diagnostic tools are not in the scope of this project. To implement the PING test, a PING functionality in Java was used. This functionality makes it possible to send and receive echo messages (PINGS), using ICMP (Internet Control Message Protocol) protocol, in machines with WINDOWS operating system. A time stamp (included in the sent packet, and retrieved in the received packet) is used to calculate the global ping time. The ping time is not calculated until the echo packet is received through the appropriate method. For more accurate time results, the ping reception should run in its own thread. This functionality is basically a class (`PingICMP.class`) that can be included in any java-based application [16].

3.6.5 Fault Management Console

The Fault Management Console component interfaces with the FC factory, FC object, Testing Server and the Alarm Database Editor components. The console's interactions, with the Testing Server, FC Object, FC Factory and Alarm Database Editor, have already been discussed. The Fault Management Console component provides the user with a graphical user interface (GUI) for this fault management system. The basic functionality of this GUI is to display all data that is processed in the Fault Management System. In this way, the user of the Console can monitor the status of the fault, from the time management system receives the fault to the time when the fault is corrected.

3.7 Chapter Summary

This chapter presents the overall design of the Fault Management Service. The chapter covers the functional requirements required for each fault management activity. The use of SNMP, CORBA, and the Notification Service is presented in this chapter. The chapter also presents the need for a scalable solution and how this solution is achieved using CORBA and the hierarchial arrangement of the TINA Management Architecture. The UML use case scenarios are used to define the fault management functional requirements, which is then used to define the principle fault management components. These components are grouped according to the fault management activity they perform. The functionality of these components, with their related IDL interfaces, is described in term of the client and server roles they played. A component diagram shows the interaction among the various fault management components. A graphical user interface (GUI) component is designed to monitor the functional activities of these components. The use of distributed object technology, CORBA, has enabled the Fault Management Service application accessible in the distributed environment. These components have also been designed such that they fulfill the fault management requirements in the NML.

Chapter 4

Implementation of the Fault Management Service

Chapter 3 presents an overall design of the Fault Management Service detailing software components, IDL interface and a GUI component used. This chapter shows how these software components and IDL interfaces interact with each other to provide a Fault Management Service. With the aid of UML sequence diagrams, a flow of events scenario of a typical fault management service, is presented. Details of the implementation of the Fault Management Service on the SATINA Trial platform is shown, highlighting the type of ORB's used, the different operating systems used as well as the different programming languages used to develop the components.

4.1 Deployment on the SATINA Platform

The various components described in Section 3.6 are implemented on the SATINA trial platform. The physical view of the SATINA platform is shown in Figure 4.1.

The SATINA network consists of three administrative domains, i.e the Connectivity Provider domain, Consumer Domain and a 3rd Party Domain. The Connectivity Provider domain consists of both an MPLS-enabled IP Core [17] as well as an ATM Core. The Fault Management Service forms part of the Connectivity Provider Domain and manages both the ATM Network as well as the IP MPLS Network.

The ATM network is connected to the IP MPLS Core, however, the ATM modules do not support MPLS. The ATM network consists of two *Fore Systems* ATM switches, models *LE155* and *ASX200BX*. The IP MPLS network consists of three *Cisco* Routers (model 3640) and 4 *Cisco* IP switches (model 2900XL). The workstations on the network are based on UNIX and Windows NT to support the software architecture over heterogeneous platforms. Java and C++ ORBs are use to provide a stable DPE on the SATINA platform.

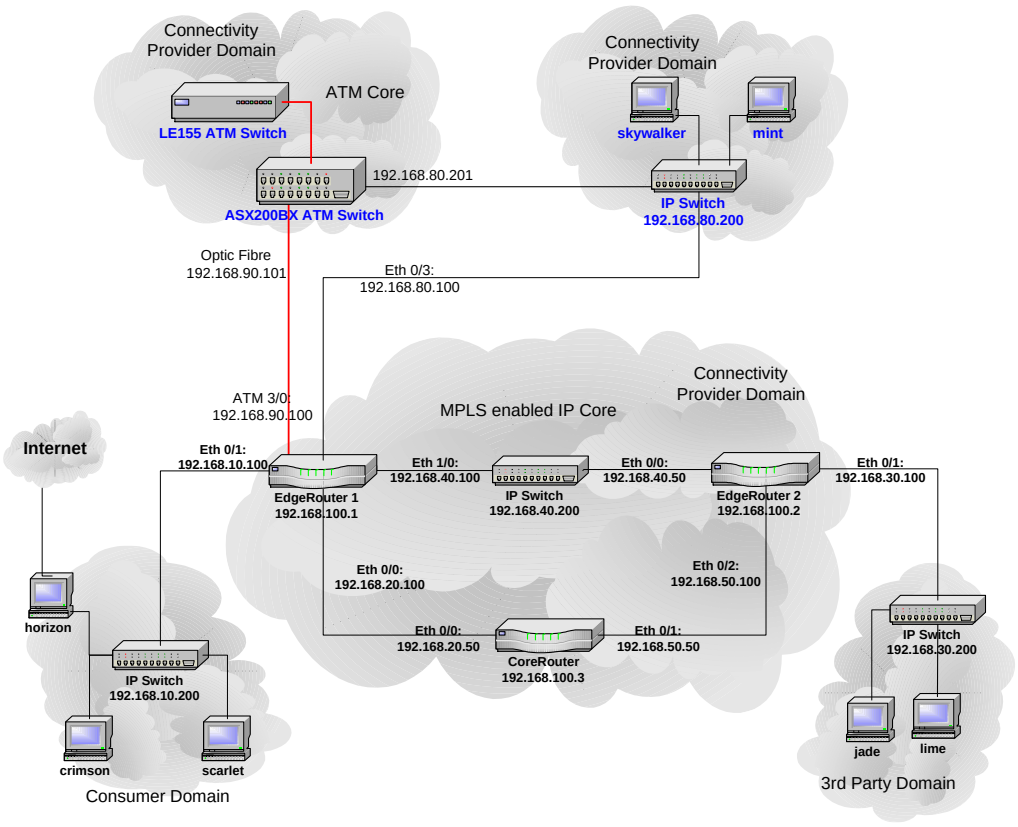

Figure 4.1: The SATINA Platform

The various components of the Fault Management Service are incorporated into the platform with the use of two different ORB implementations. The two host computers used in implementing this service are *"skywalker"* and *"mint"*. For the Fault Management Service, all routers and switches (IP and ATM) had been configured to issue traps to the host machine *skywalker*.

Figures 4.2 and 4.3 show the software configurations that implement the fault management components. The Alarm Manager, FC Factory, FC Object, Alarm Database Listener and the Alarm Database Editor components are implemented in the host machine *mint* with the The ACE ORB or TAO (version 1.1.17) implementation. Two additional components that is implemented on this host machine is the TAO Naming Service and the TAO Notification Service. TAO offers a Real-Time notification service. This service is used to create the notification channel through which the trap alarms are sent. All components that played a role in providing a Fault Management Service used the TAO Naming service to bind their references. All the components are programmed using C++.

The remaining fault management components, i.e. the Trap Listener, Testing Server and

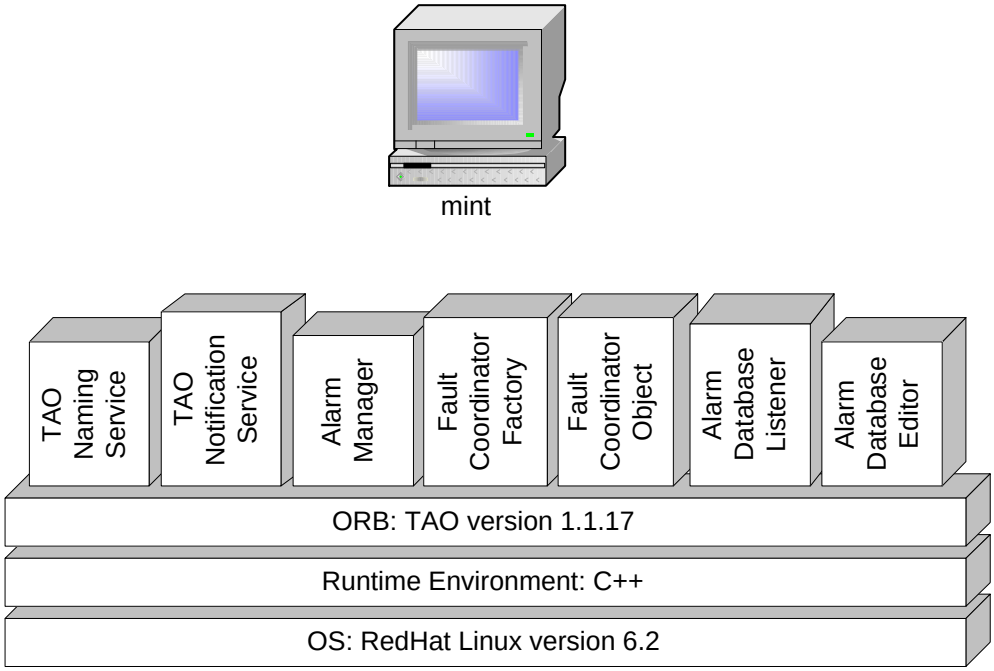

Figure 4.2: Components on Host Machine *mint*

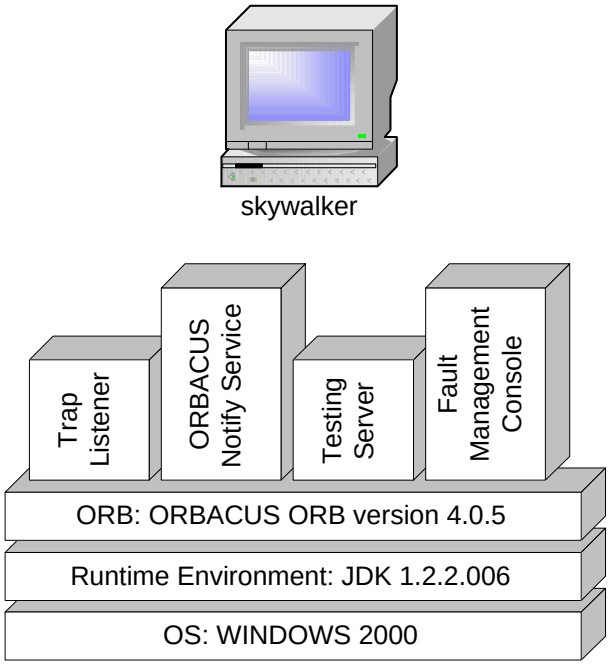

Figure 4.3: Components on Host Machine *skywalker*

the Fault Management Console are implemented on the host machine *skywalker* with the ORBACUS ORB implementation. An ORBACUS version of the notification service is also used to manage the dispatching of trap alarms from the supplier, i.e the Trap Listener component, of the notification channel. All these components use the TAO's Naming Service to bind their object references. All the components are programmed using Java.

4.2 A Sequence of Events of a Fault Management Service

This section shows, with the aid of UML sequences diagrams, a typical Fault Management Service scenario . A **"linkdown"** fault is generated to illustrate the interaction of the components in a typical fault management scenario. The link connecting the Router (IP address 192.168.100.1) to IP Switch (IP address 192.168.40.200) is disconnected in order to generate a fault. Alarms are generated by managed devices, IP Switch (IP address 192.168.80.200) and Router (IP address 192.168.100.1). The IP Switch (IP address 192.168.80.200) and Router (IP address 192.168.100.1) are hereafter referred to as *EdgeRouter1* and *Switch80* respectively. The "broken link" scenario is shown in figure 4.4.

4.2.1 Sequence Event: Capturing and Logging of Trap Alarms

Figure 4.5 deals mainly with trapping the alarm information dispatched by the managed device. The alarms are then sent through the notification channel where the alarms are received by the consumer of the notification channel. The consumer filters and logs the alarm in a database.

1 In sequence diagram Figure 4.5, the damaged link causes the network elements, in this case an *Switch80* and *EdgeRouter1*, to issue trap alarms to the host machine *skywalker*. Both the elements send Trap information with the following fields: Date, Source, Remote Host, Port, Community, Agent Address, Enterprise OID, and Trap Variable OID.

2-5 The Trap Listener component is configured to receive these trap alarms. Upon receiving the trap alarms, the fault Trap Listener prepares these trap information to be sent through the notification channel. The Trap Listener accomplishes this by packaging the trap information into a structured data format (see Figure 3.3). The trap information is then sent immediately into the notification channel. The Trap Listener does not know the recipient of this notification.

5-8 The Alarm Manager component subscribes to receive notifications from the channel. It receives these trap notifications from the channel. Since the network element

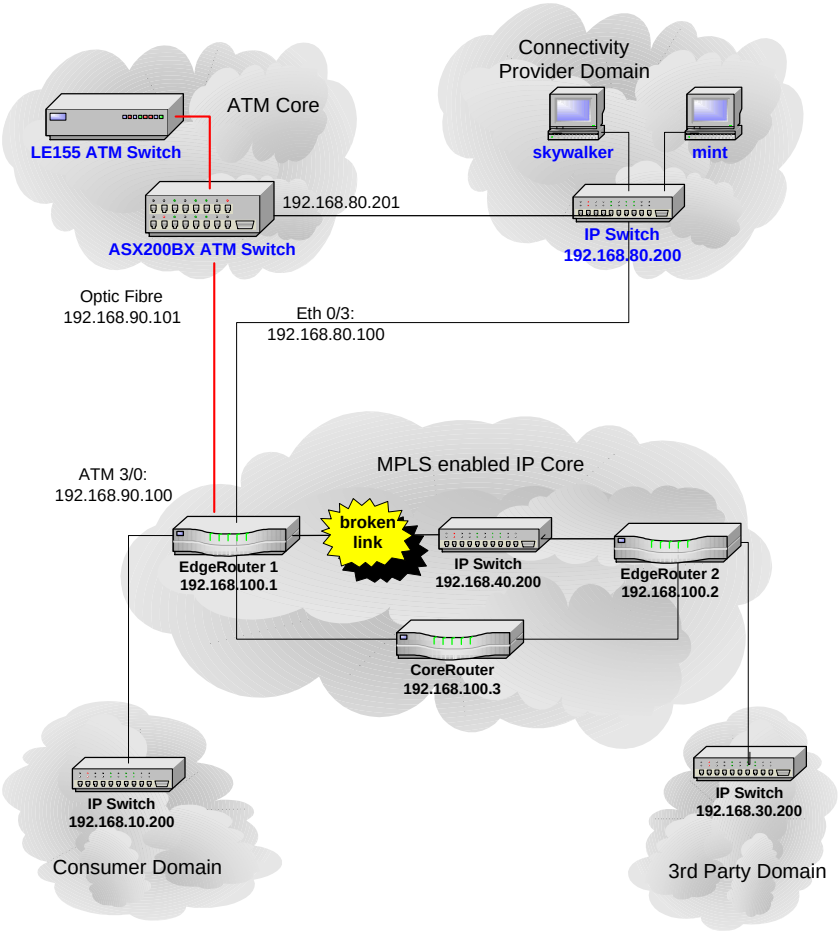

Figure 4.4: *"Damaged Link"* Fault Scenario

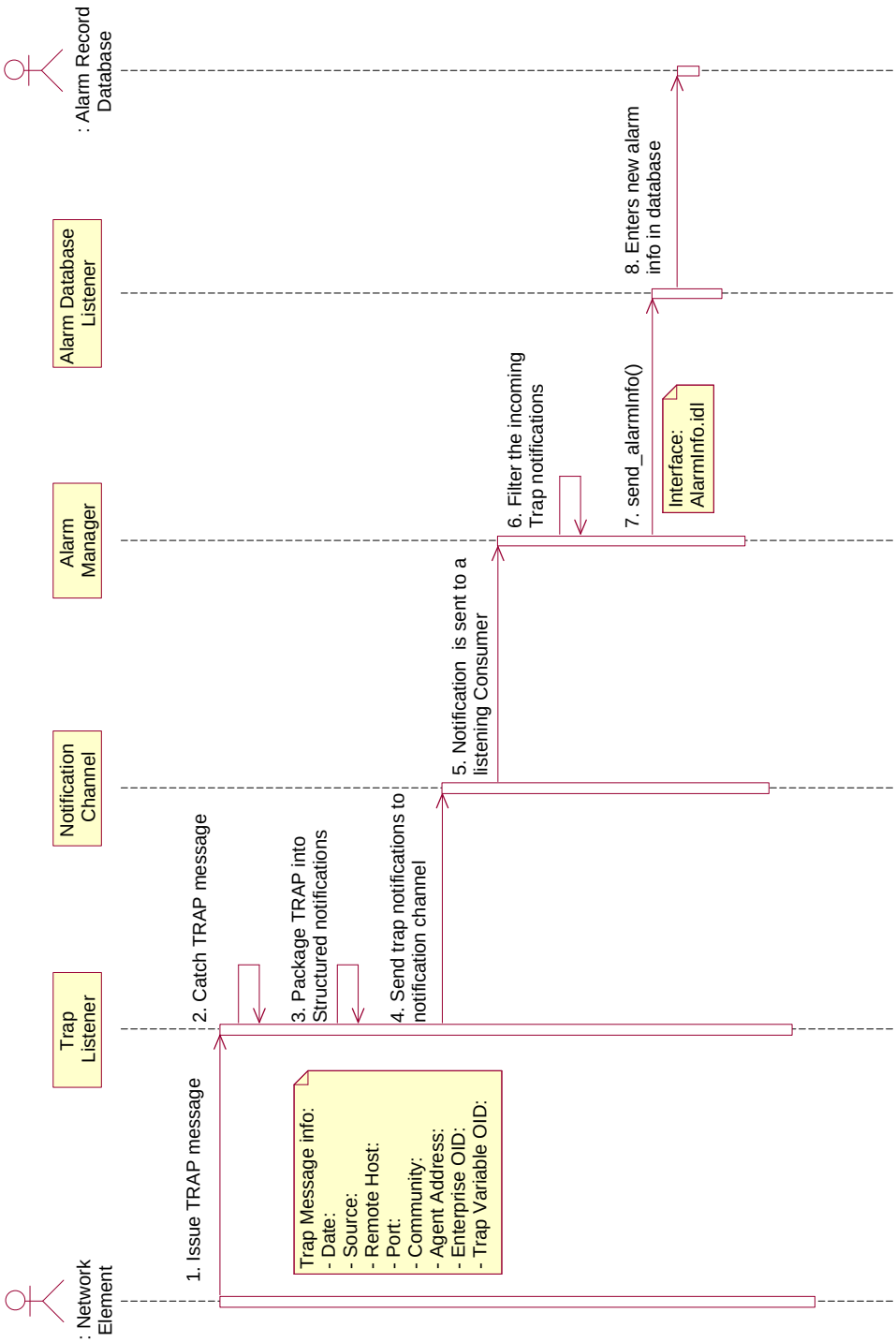

Figure 4.5: Sequence Event: Capturing and Logging of Trap Alarms

continues to issue identical trap alarms until the fault is corrected, the Alarm Manager then filters and removes redundant alarms. After the filteration process is completed, the trap alarm is then sent to the Alarm Database Listener component via the `AlarmInfo` interface. The Alarm Database Listener then inserts this alarm information into a database.

4.2.2 Sequence Event: Forwarding Trap Alarms to Fault Coordinator Components

Figure 4.6 follows from Figure 4.5 where the Alarm Manager forwards the alarm information to the Alarm Database Listener. This diagram deals mainly with forwarding the trap alarm to the Fault Coordinator components, where fault localisation takes place.

9 The Alarm Manager also forwards the alarm information to the Fault Coordinator Factory (FC factory) via the `fcFactory` interface.

10 - 11 The FC Factory then receives the alarm information. FC Factory creates a Fault Coordinator Object (FC object) for that particular alarm. For each new alarm received, the FC factory creates a new FC object to handle the alarm.

12 Once the FC object is created, the FC factory sends the alarm information to the FC object via the `fc` interface.

13 The FC factory then assigns a Fault identification number (FaultID) to the FC object it created. The FaultID together with its corresponding FC object reference and the date when the object was created is stored in a list. The FC factory then awaits for a request for the lists from the Fault Management Console.

14 The Fault Management Console requests the fault list, containing the FaultID, Date and FC object reference, from the FC factory via the `fcFactory` interface. Figure 4.7 shows the screen capture of Fault Management Console GUI after the FC factory has been requested for the fault list.

4.2.3 Sequence Event: Fault Localisation Process

Figure 4.8 shows the sequence of events that takes place during the fault localisation process. This diagram follows Figure 4.6.

15 The user of the Console selects a FaultID from the alarm list. When the selection is done, the corresponding FC object reference is automatically selected. The user

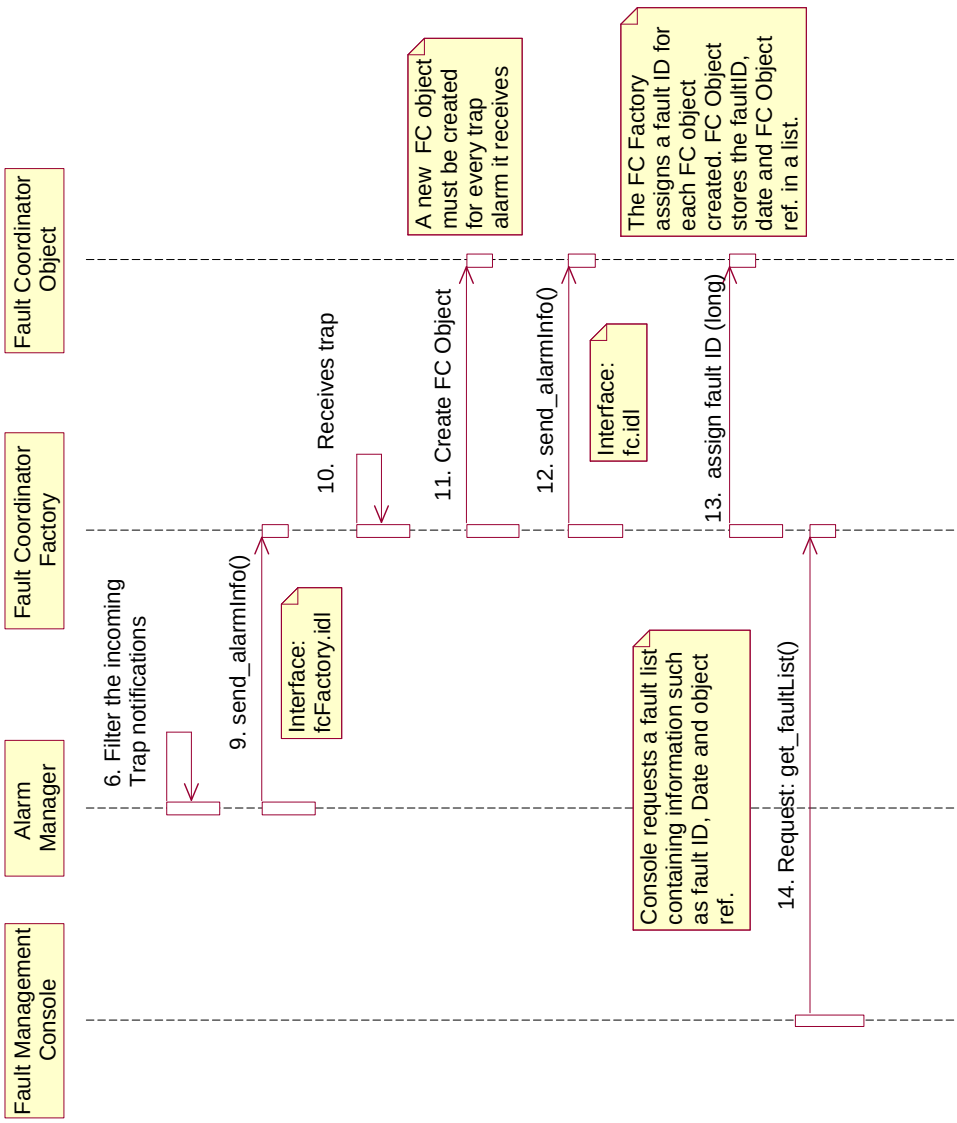

Figure 4.6: Sequence Event: Forwarding Trap Alarms to Fault Coordinator Components

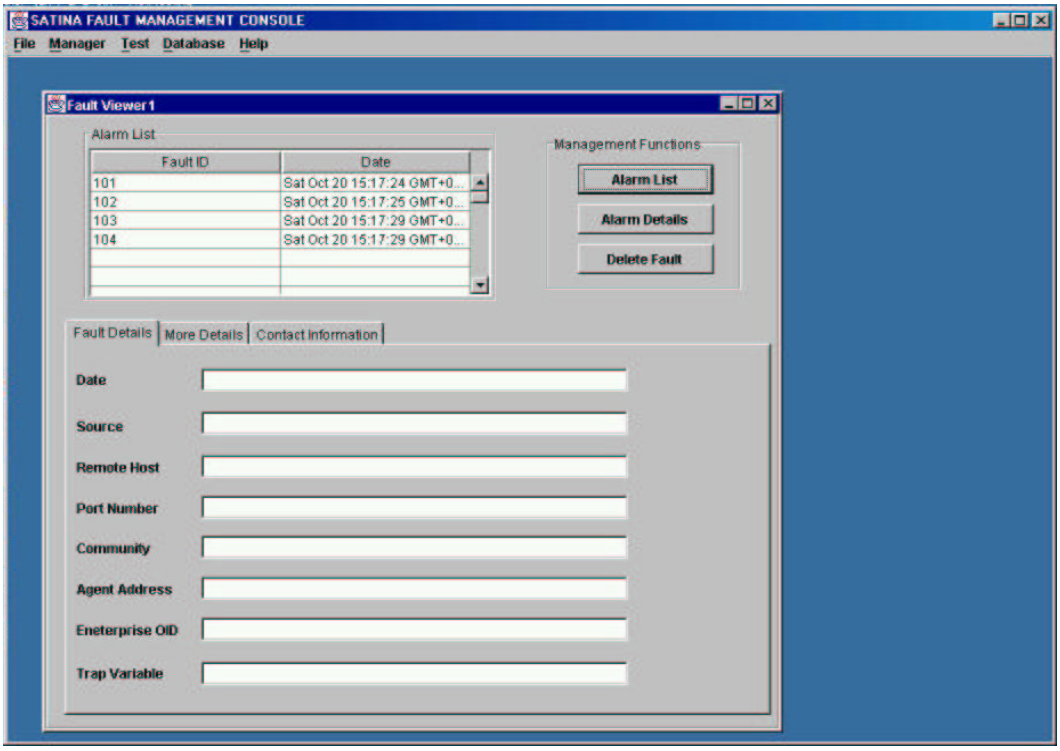

Figure 4.7: Screen Capture of the Fault Management Console GUI showing the Alarm Lists

of the Console then request for alarm details using this FC object reference. Figure 4.9 shows a screen capture of the Fault Management Console GUI showing the alarm details of the FC object selected.

17 - 18 Using the same FC object reference as in *Step 15*, another request is made to get more details relating to the alarm. Upon receipt of this request, the FC object queries the equipment topology database to help localise the fault. The results of this query are then displayed on the Fault Management Console. The results are shown in Figure 4.10

19 - 20 At the same time as the above request is made, the console requests the network technician's contact details in order to correct the fault. The FC object then queries the Contact Information database for the information. This information is then sent to the Console where it is displayed. Figure 4.11 shows a GUI with the database results. A functionality not yet implemented is Short Message Service, where the network administrator sends an SMS message to the technician's mobile celluar phone, informing him/her of the fault details. See Figure 4.11.

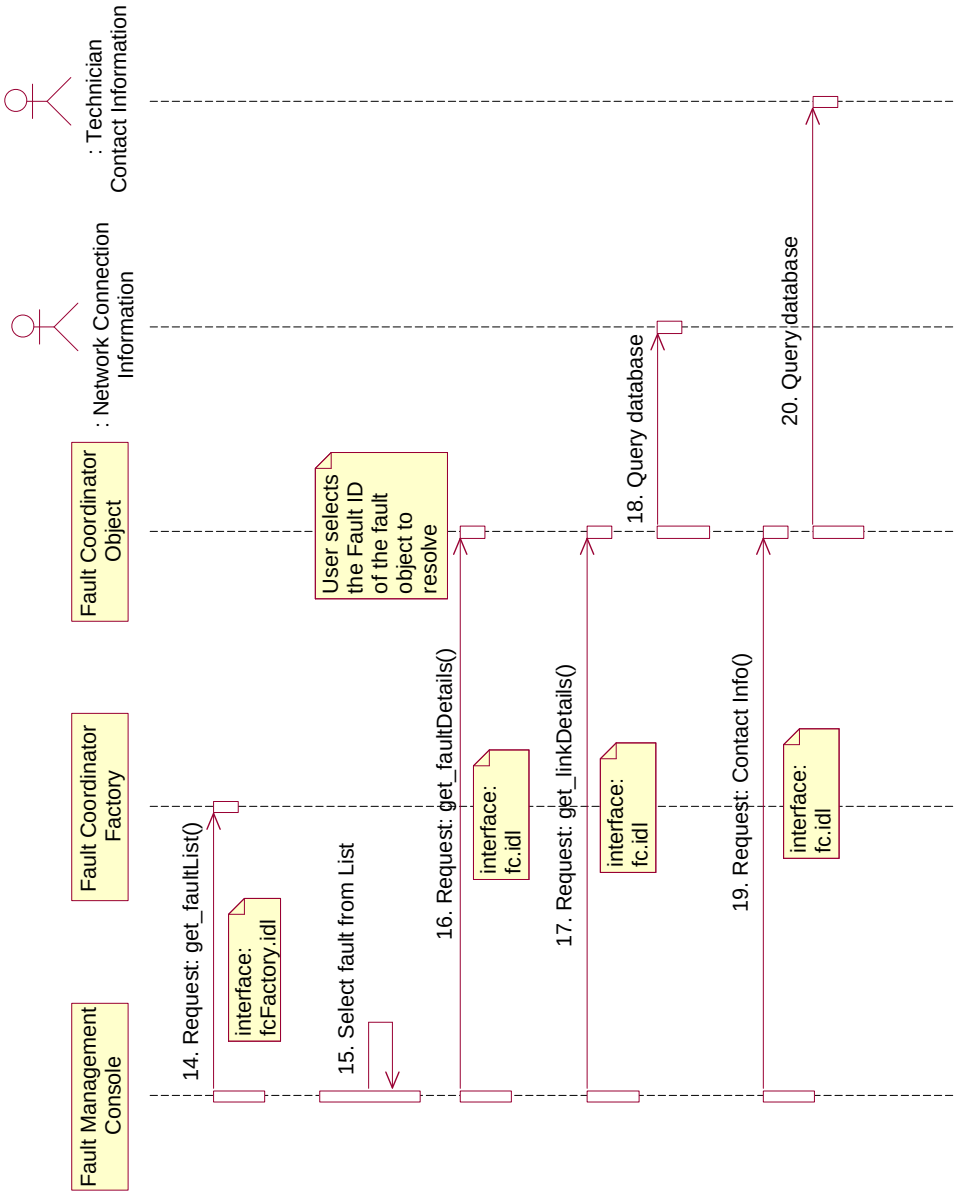

Figure 4.8: Sequence Event: Fault Localisation Process

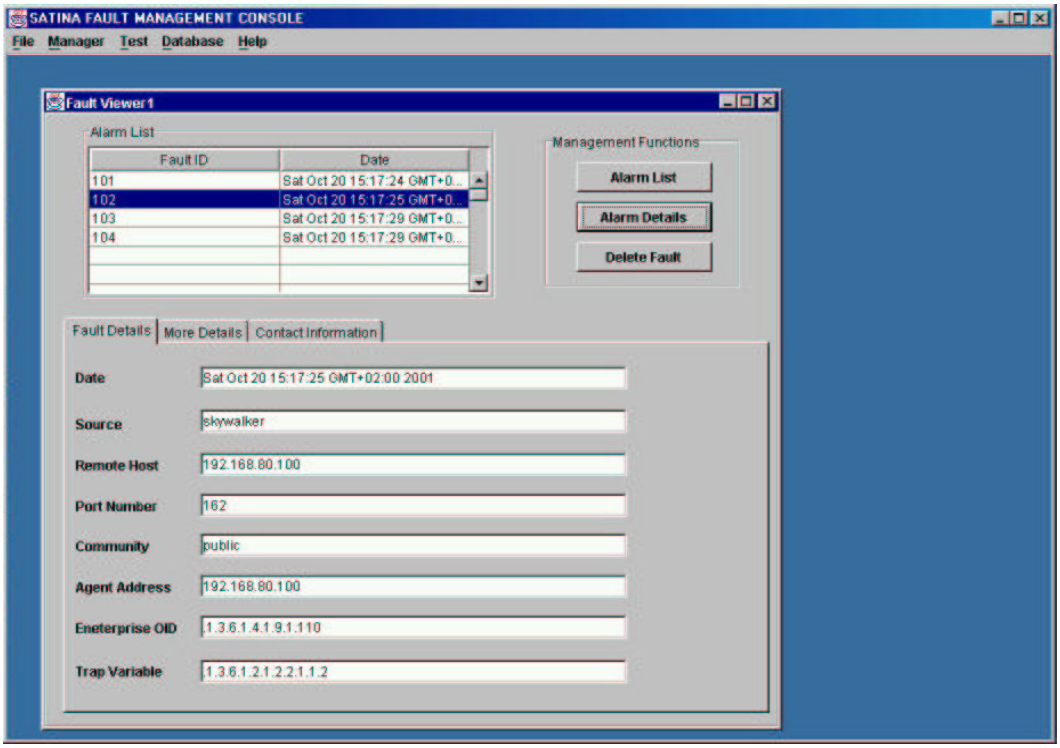

Figure 4.9: Screen capture of the Fault Management Console GUI showing the alarm details

4.2.4 Sequence Event: Testing Functionality

The sequence diagram in Figure 4.12 shows the sequence of events of the Testing functionality of the Fault Management Service.

21 The IP addresses received form the Fault localisation process, is used in this testing function. An IP address of a network element is sent via the `tdsServer` interface to the Testing Server component.

22 - 23 The Testing Server component than runs the PING test on the network element with that IP address. The results of the PING test are then sent to the Fault Management Console. These results are shown in Figure 4.13.

4.2.5 Sequence Event: Updating the Alarm Database

Figure 4.14 shows the sequence of events when the fault manager manages the alarm record database. Management of the alarm database includes the viewing of the database, updating the database as well as deleting alarm entries from the database.

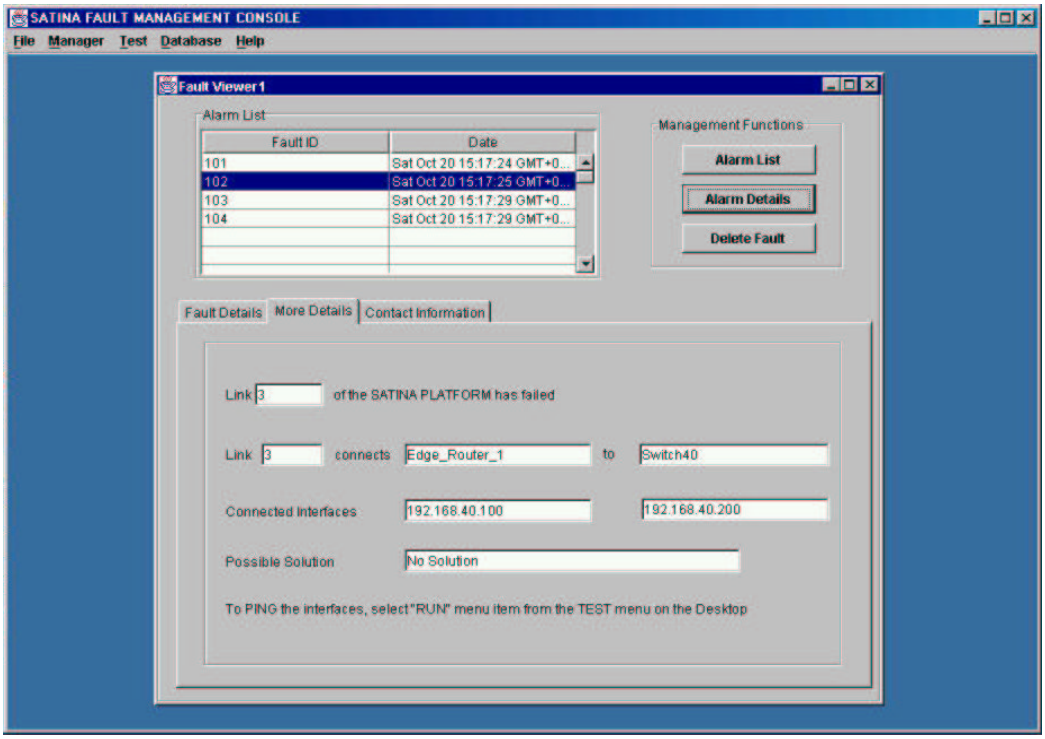

Figure 4.10: Screen capture of the Fault Management Console GUI showing the database query results of the Trap alarm

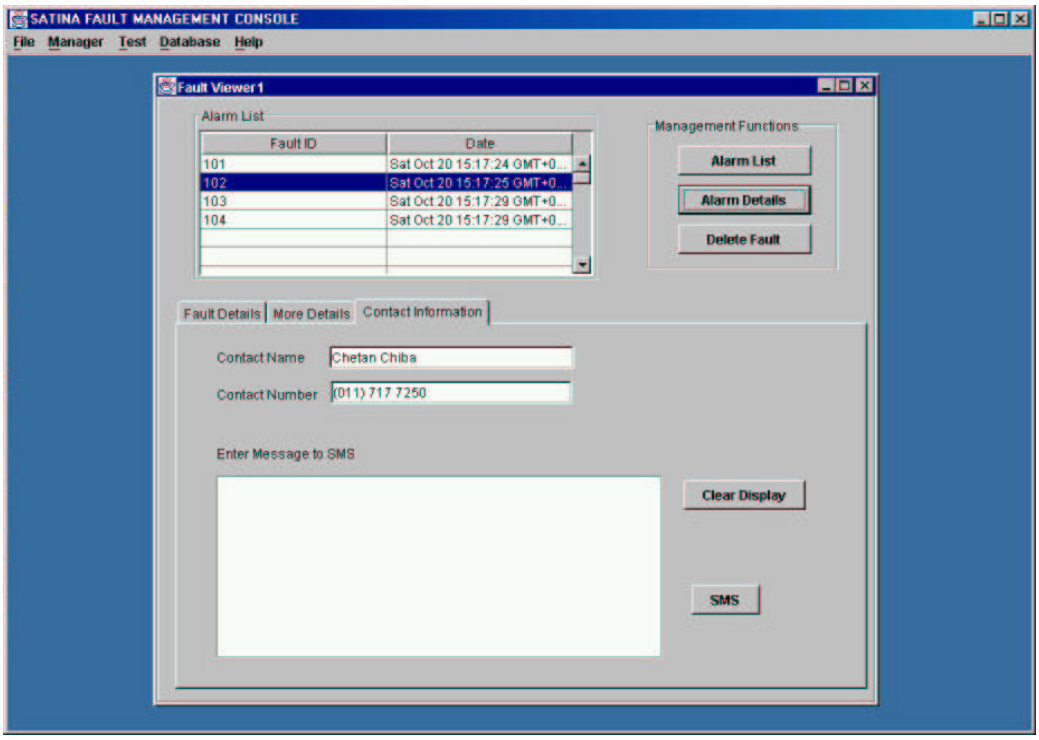

Figure 4.11: Screen capture of the Fault Management Console GUI showing the Network Technician's Contact information

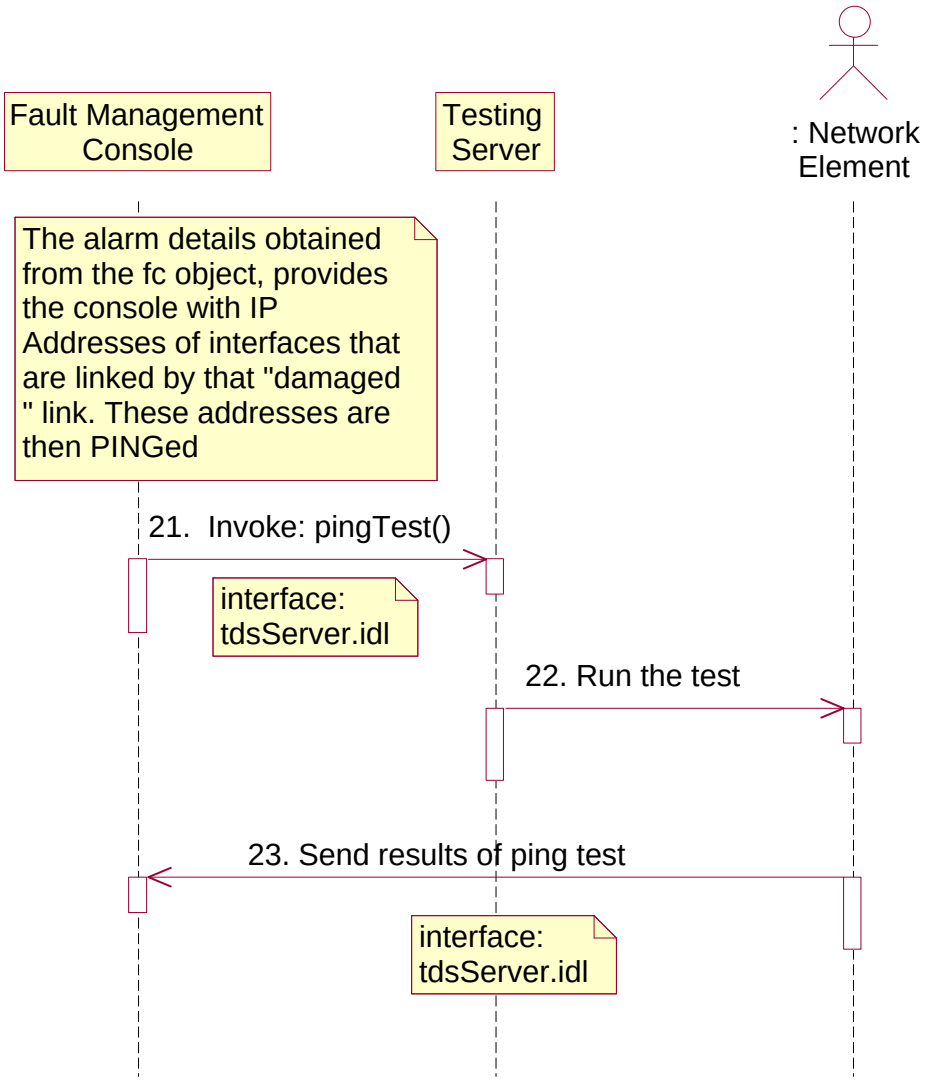

Figure 4.12: Sequence Event: Testing Functionality

24 - 25 The fault manager requests to view the alarm information entries in the Alarm Database. The fault manager request an alarm list from the Alarm Database Editor component, via the interface `Archive` interface. This component then queries the Alarm Record database. The results of the query are then displayed on the Fault Management Console, i.e. Figure 4.15.

26 The fault manager then selects the alarm entry fields to view. This is shown in Figure 4.16. With this selection, the network administrator has an option either to update the fields in the entry or delete the entire alarm entry.

27a - 28a When the fault manager wishes to Update a field entry in the database, he/she selects

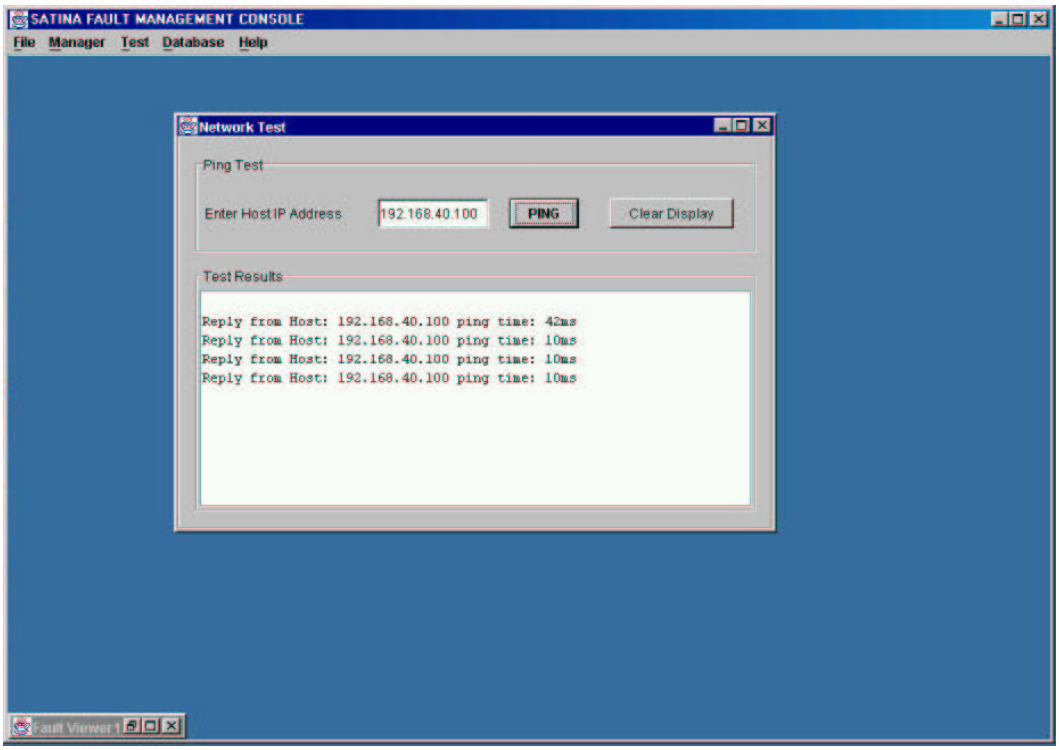

Figure 4.13: Screen capture of the Fault Management Console GUI showing the Testing Functionality

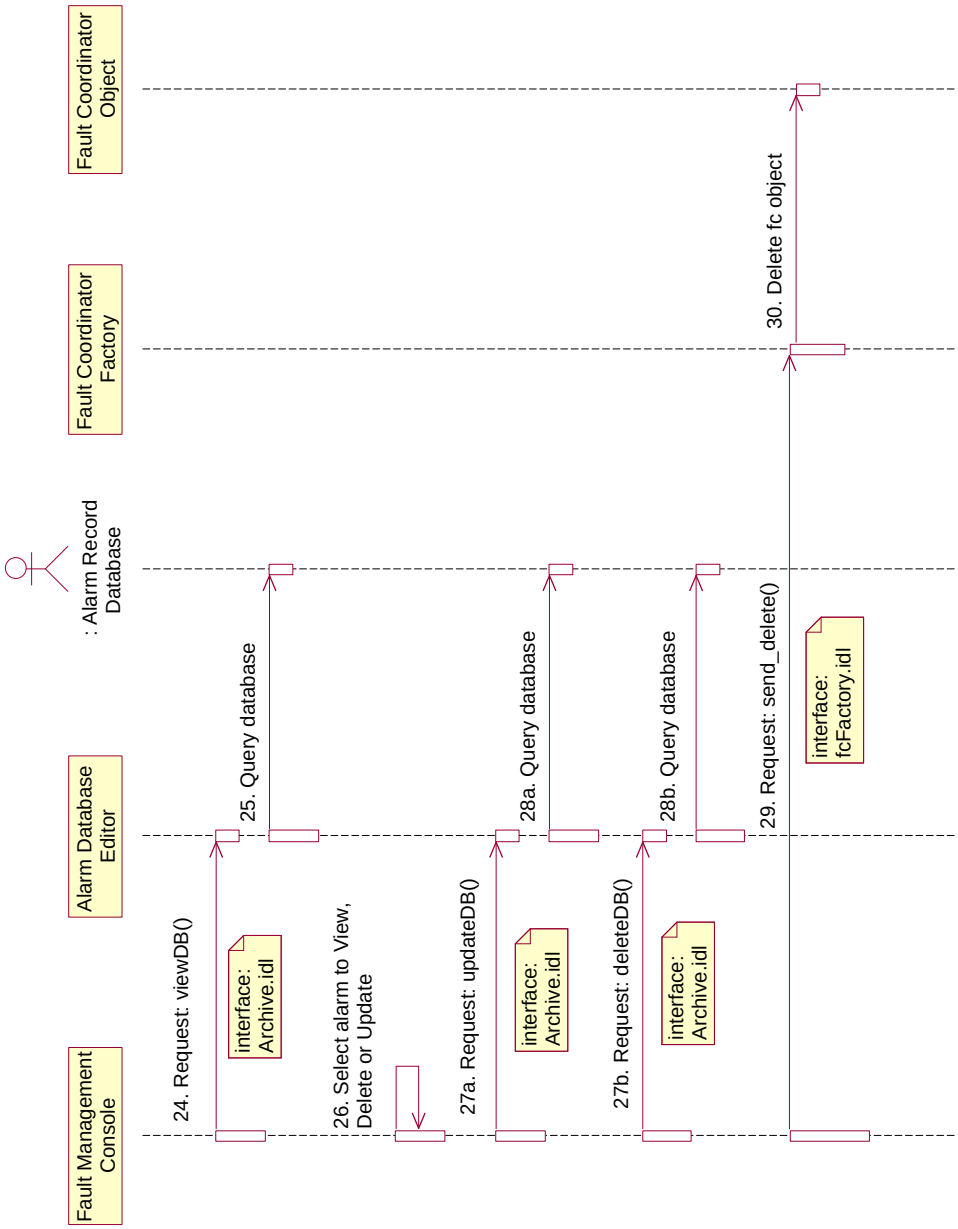

Figure 4.14: Sequence Event: Updating the Alarm Database

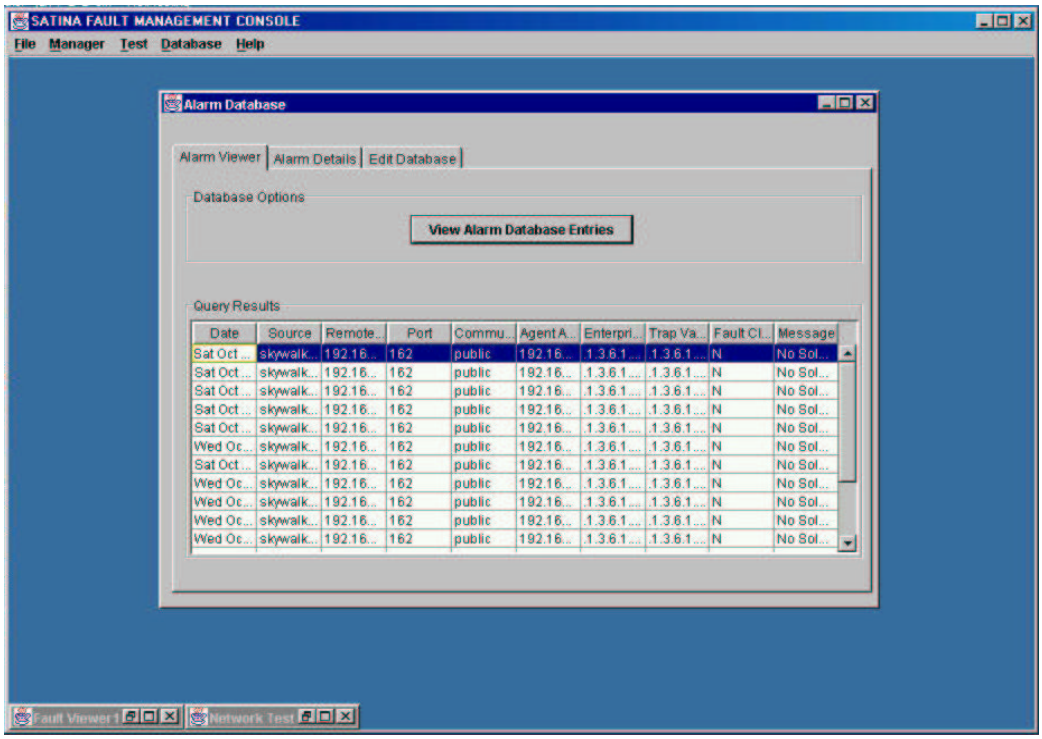

Figure 4.15: Screen capture of the Fault Management Console GUI showing the Alarm Record database entries

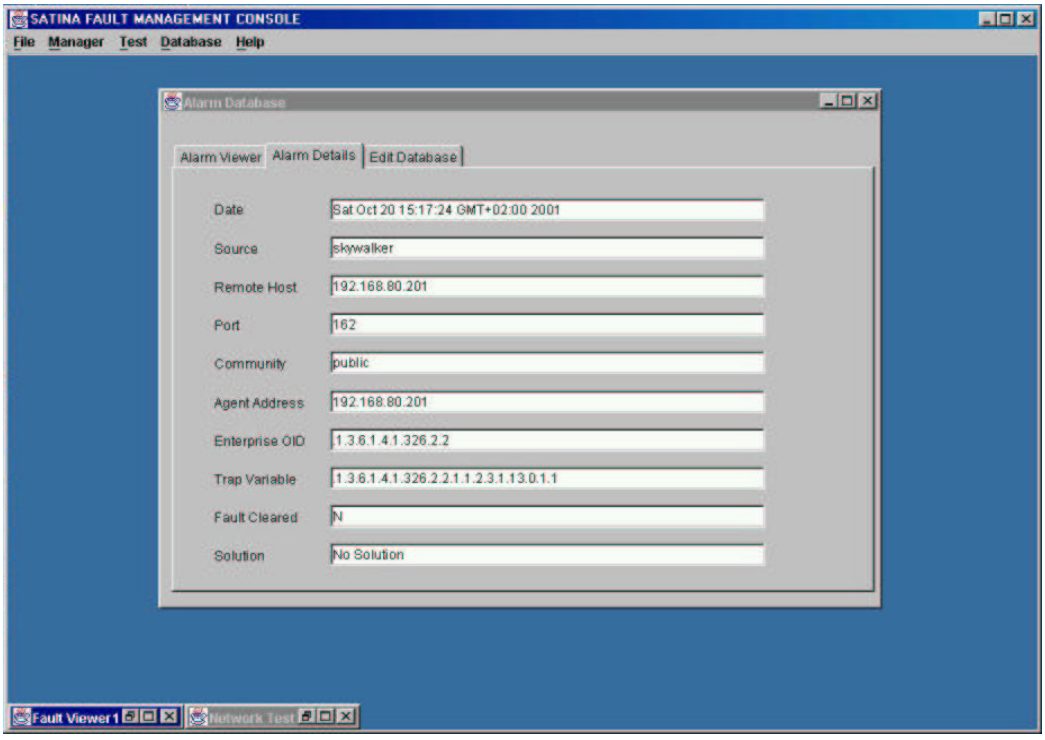

Figure 4.16: Screen capture of the Fault Management Console GUI showing the Alarm Record database entries

the alarm entry, as in Step 26. Once the selection is made, the corresponding Date field includes the date of the alarm to be updated. The fault manager then updates the required fields. Once this is done, the fault manager sends the data to Alarm Database Editor component. This Editor component then queries the database and updates the required fields in the database. These fields are shown in Figures 4.17.

27b - 28b When the administrator wishes to Delete an entry in the database, he/she selects the alarm entry, as in Step 26. Once the selection is made, the corresponding Date field includes the date of the alarm to be deleted. This field is shown in Figures 4.17. The administrator then presses the delete button.

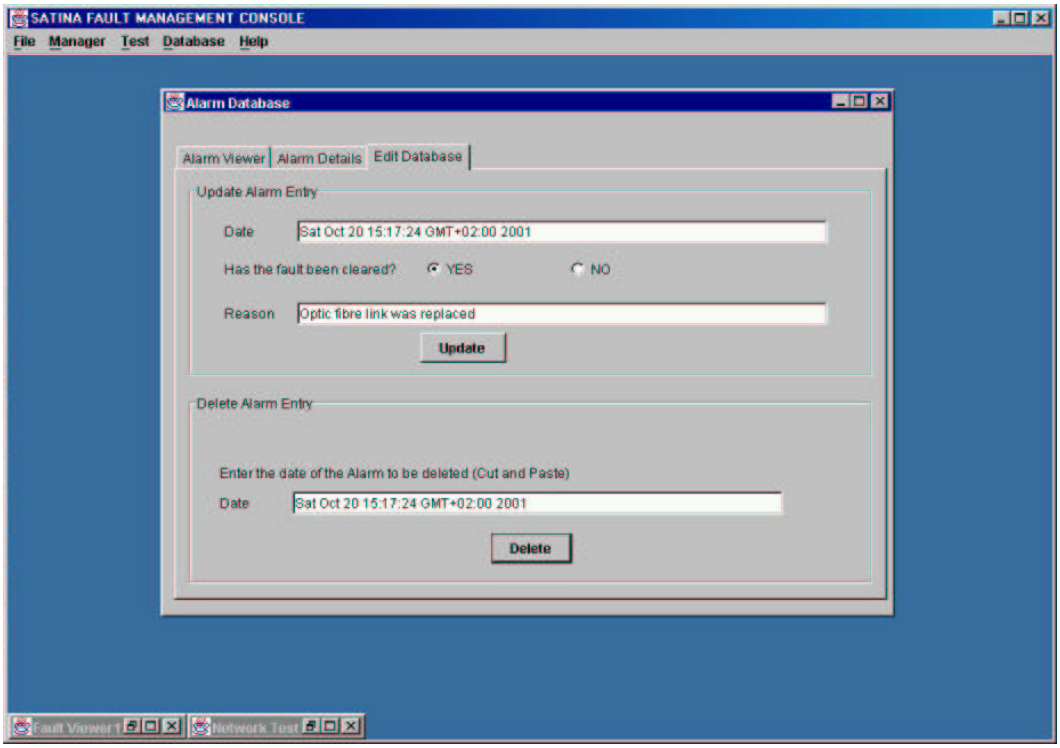

Figure 4.17: Screen capture of the Fault Management Console GUI showing the Update and Delete Procedure of an Alarm entry

29 - 30 When the alarm has been analysed, localised, corrected by the network technician, and the alarm database updated, the administrator has to delete the FC object. To do this, the administrator selects the FaultID to be deleted, as shown in the Figure 4.7 and Step 15. The Console sends this Delete request via the `fcFactory` interface, to the FC factory which deletes this FC object.

.

4.3 Limitations of this Design

The Fault Management Service design has certain limitations. These limitations are described below.

4.3.1 No Fault Correction

The focus of this project was to monitor, detect and localise faults across heterogeneous networks. Hence, this service does not perform fault correction. The fault correction is performed by the network technician. This design informs the network administrator of the fault conditions and performs processes to isolate the fault. With this information, the administrator has the knowledge of the possible causes of the fault. This information is then given to the a network technicians who rectifies the fault.

4.3.2 Limited fault types tested

The only fault types that were tested with the Fault Management Service were the *"linkdown"* and *"login"* faults. The fault localisation procedure could only localise faults of this nature. This design focused on fault detection and isolation in a TINA environment, and the above two fault are used solely for testing these functionalities. The true performance of this system had not been tested due to the limited fault conditions tested.

4.3.3 Need for a generic MIB

Since this service had to be designed to manage heterogeneous devices, a generic or a vendor-independent MIB is required. For this design, the MIB RFC 1213 [18] is used as it provided a limited number of generic variables used by all network devices. However, more management information is required to be made available from these network devices in order for the FMS to manage the network efficiently. To retrieve this information, vendor dependent MIB's are required. Hence a need for a generic MIB is required, for this design, in order to retrieve more management information from any network device.

4.4 Chapter Summary

This chapter presents the deployment of the FMS on to the SATINA trial platform. The different ORB implementations, operating systems, network configurations and run-time environments available on the SATINA trial platform provided a heterogeneous environment

to design, implement and test this Fault Management Service. A *linkdown* fault scenario is used to evaluate the performance of this Fault Management Service. UML sequence diagrams together with screen-captured pictures of the Fault Management Service GUI, is used to provide a detailed explanation of a typical Fault Management Service scenario. This UML sequence diagrams shows the operation of various components in providing a Fault Management Service in a TINA network transport environment. Finally, this chapter also highlighted the functional limitations of this design.

Chapter 5

Conclusion

5.1 Discussion

Chapter 1 of this report discusses the importance of implementing a fault management system to manage large and complex communication networks. The issue of the rapid growth of telecommunications brings into question the scalability of fault management systems. The importance of scalable fault management systems are further stressed with the movement of telecommunication network to a distributed processing environment. The issues of implementing a scalable fault management system for distributed networks are the reasons for the development of this distributed fault management service.

This report has detailed the design, implementation and evaluation of the scalable distributed fault management service to be used for growing heterogeneous networks. One of the major motivations for a distributed fault management service was to develop a single fault management service that manages heterogeneous networks. The TINA NRA and Management Architecture has been used to develop this fault management service since it provides a generic architecture for the development of such a fault management functionality. Chapter 2 provides a detailed view of Fault Management as defined by the TINA NRA and Management Architecture. The view identifies fault management functional activities, functional requirements and computational objects that perform a fault management service. The view serves as the basis on which the Fault Management Service was developed.

Chapter 3 provides the design methodology used to develop this fault management service. This methodology uses the TINA NRA and Management Architecture specifications to design and develop this fault management service. Being constrained to use SNMP to access MOs, the methodology describes how SNMP is incorporated into the design. CORBA and the CORBA-based Notification service was incorporated into the fault management service design. CORBA provided an implementation environment that facilitate location and implementation transparency. The CORBA notification service played an important role in

reporting faults to the management service. The principle software components together with their related CORBA interfaces, are developed to implement a fault management service. Since both EML and NML have similar functional requirements, the components developed are applicable to both management layers.

Chapter 4 describes the deployment of this Fault Management Service onto the SATINA trial platform. The different ORB implementations, operating systems, network configurations and run-time environments available on the SATINA trial platform provided a heterogeneous environment to design, implement and evaluate this Fault Management Service. To evaluate the viability of the proposed fault management service, a *"linkdown"* fault was generated on the SATINA platform. The fault management service reported, located and detected the fault. The fault management service tracked the status of the fault with the use of a GUI. The functionality of the fault management service is limited in the sense that the service cannot perform automatic fault correction. Since only one fault type was tested, the true performance of the fault management service is not known.

The findings in the study and conclusions drawn are discussed in section 5.2. Recommendations for future work regarding the implemented fault management service are suggested in section 5.3.

5.2 Conclusions

This section concludes the report on the development of a distributed fault management service for the SATINA Trial project, presenting the findings of the work.

5.2.1 The Need for a Distributed Fault Management Service

The developed Fault Management Service defines a distributed management functionality that is capable of providing fault management support across heterogeneous networks. The generic fault management service developed demonstrates that with the use of both the CORBA-enabled computational objects (CO), SNMP traps can be processed in large-scale heterogeneous networks. The combination of CORBA-based elements and the hierarchial arrangement of the TINA Management Architecture, allows the distributed fault management service to be scalable.

The distribution of management functionality in the lower management layers (i.e the EML) allowed for the complete management of local faults. Managing local faults locally is more efficient in time and bandwidth. Only faults that span multiple networks are handled by the fault management service in the upper management layers (i.e. the NML). Hence, the

distributed architecture of the fault management service provides a flexible and scalable management architecture.

In summary, the Fault Management Service implementation validates the TINA NRA and Management Architecture by showing that TINA's concepts and principles can be implemented.

5.2.2 Demonstration on the SATINA platform

The proposed Distributed Fault Management Service outlined in the report was implemented on the SATINA Trial project. The different ORB implementations, operating systems, network configurations and run-time environments available on the SATINA trial platform provided a heterogeneous and distributed environment to evaluate the Fault Management Service.

The implementation of the various components, used to perform the fault management service, are evaluated by generating a *linkdown* fault on both the ATM core network as well as the MPLS-core network. The *linkdown* failure test performed on the SATINA platform concludes that the Fault Management Service is applicable to any connection-orientated network that is modeled using the TINA NRA specification and principles. The demonstration of the service proved the capability of the fault management service to monitors, locate and detects faults encountered in the network. The true performance of the fault management system is unknown as the service could not be tested in a large scale distributed environment where thousands of elements are required to be managed.

In conclusion, the TINA NRA and Management Architecture specification have been used to design, implement and evaluate a Distributed Fault Management Service for the management of the SATINA transport network. The project has shown the proposed Fault Management Service to be capable of delivering the functionality that the design set out to provide.

5.3 Recommendations for Future Work

5.3.1 Fault Correction

The implemented Fault Management Service did not provide a fault correction process. In an ideal scenario, the fault corrective action forms part of a typical fault management system. Hence a fault correction action should be designed and incorporated into this Fault Management Service in order to automatically correct faults. The corrective process should

determine what analysis, testing, or repair activity is required to be performed. The corrective process should isolate the unit with the fault and report on automatic restoration process within the NE.

5.3.2 Integrated Fault Management

The term "fault" is usually taken to mean the same as "failure", which means component (hardware or software) malfunctions, e.g. sensor failures, broken links or software malfunctions [11]. Such faults are called *"hard"* faults and can be solved by replacing hardware elements or software debugging and/or re-initialization. The diagnosis of the *"hard"* faults is called "re-active" diagnosis in the sense that it consists of basically the reactions to the actual failures. In communication networks, however, there are other important kinds of faults that need to be considered. For example, the performance of a switch is degrading or there exists congestion on one of the links. Since there might not be a failure in any of the components, such faults are referred to as *"soft"* faults. *"Soft"* faults are in many cases indications of some serious problems and for this reason, the diagnosis of such faults is called "pro-active" diagnosis. By early attention and diagnosis, such pro-active management will sense and prevent disastrous failures and thus can increase the survivability and efficiency of the networks.

For this design, only *"hard"* faults are tested. To test the functionality of the Fault Management Service, the service should also be capable of detecting, isolating and correcting *"soft"* faults. This can only be achieved if vendor-dependent MIB’s are used, since these MIBs allows more management information to be retrieved from the network devices. In this case, vendor-independent MIB can be created in order to get more management information from the managed device.

References

[1] A. Sahai and C. Morin, "Towards Distributed and Dynamic Network Management," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, (New Orleans, USA), pp. 15–20, February 1998. http: citeseer.nj.nec.com/sahai98towards.html.

[2] C. Abarca and J. Forslow, *Network Resource Architecture*. TINA Consortium, http://www.tinac.com, 10 February 1997.

[3] M. Chapman and M. Stefano, *Overall Concepts and Principles of TINA*. http://www.tinac.com, 17 February 1995.

[4] R. A. Achterberg and H. E. Hanrahan, "The South African TINA Trial: SATINA - Project Status and Vision," January 1999. url: http://satina.ee.wits.ac.za.

[5] L. Fuente and T. Walles, *Management Architecture*. TINA Consortium, http://www.tinac.com, December 1994.

[6] ITU-T Recommendation M.3010, *Principles for a Telecommunications Management Network*, 1992.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin, " A Simple Network Management Protocol," tech. rep., Network Working Group, May 1990. RFC 1157.

[8] W. Stallings, "SNMPv3: A Security Enhancement for SNMP," *IEEE Communications Surveys*, vol. 1, no. 1, 1998. url: http://www.comsoc.org/pubs/surveys.

[9] G. Goldszmidt, Y. Yemini, K. Meyer, and M. Erlinger, "Decentalizing Control and Intelligence in Network Management," in *4th International Symposium on Integrated Network Management*, May 1995. url: http://netman.cit.buffalo.edu/Doc/Papers/gol9505.ps.

[10] "CORBA." url: http://www.corba.org/.

[11] J. Baras, H. Li, and G. Mykoniatis, "Integrated, Distributed Fault Management for Communication Networks," April 1998. url: http://www.isr.umd.edu/CSHCN/.

[12] OMG, *Notification Service Specification*, June 2000. url: www.omg.org.

[13] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[14] AdventNet, Inc., *AdventNet SNMP API Release 3.2: Help Manual*, March 2001. url: http://www.adventnet.com.

[15] "MySQL." url: http://www.mysql.com.

[16] M. I. Garcia and O. Fernandez. http://www.geocities.com/SiliconValley/Bit/5716/ping/.

[17] H. M. Morar, "Development of a Distributed Connection Management System for the Provision of QoS in IP Networks," Master's thesis, University of the Witwatersrand, School of Electrical and Information Engineering, Johannesburg, South Africa, November 2001.

[18] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB II," March 1991. url: ftp://ftp.rfc-editor.org/in-notes/rfc1213.txt.

Appendix A

IDL interface specifications

A.1 AlarmInfo Interface

```
// File: AlarmInfo.idl

interface AlarmInfo {
    void send_alarmInfo( in string date,
                         in string source,
                         in string remote_Host,
                         in string port,
                         in string community,
                         in string agent_Address,
                         in string enterprise,
                         in string trapVariable);
};
```

A.2 The fcFactory Interface

```
// File: fcFactory.idl
typedef sequence<Object> ObjList;

struct facGui {
    long FaultID;
    string Date;
    Object ObjRef;
};
```

```
typedef sequence<facGui> facGuiSeq;

interface fcFactory {

    void send_alarmInfo( in string date,
                         in string source,
                         in string remote_Host,
                         in string port,
                         in string community,
                         in string agent_Address,
                         in string enterprise,
                         in string trapVariable);

    void get_faultList( out facGuiSeq faultList);

    void send_delete(in long fault_id);
};
```

A.3 The fc Interface

```
// File: fc.idl

interface fc {

    void send_alarmInfo( in string date,
                         in string source,
                         in string remote_Host,
                         in string port,
                         in string community,
                         in string agent_Address,
                         in string enterprise,
                         in string trapVariable);


    void get_faultDetails(  out string gDate,
                            out string gSource,
                            out string gRemoteHost,
                            out string gPort,
```

```
                              out string gComm,
                              out string gAgAddress,
                              out string gEnterP,
                              out string gtVariable);

    void get_linkDetails( out string solution,
                          out string linkNum,
                          out string equip1,
                          out string equip2,
                          out string ipAdd_1,
                          out string ipAdd_2);

    void get_contactInfo( out string contactName,
                          out string contactNumber);
};
```

A.4 The tdsServer Interface

```
// File: tdsServer.idl

typedef sequence<string> pingResult;

typedef sequence<pingResult> pingResultSeq;

interface tdsServer {

      void pingTest(    in string aAddress,
                        out pingResultSeq results);
};
```

A.5 The Archive Interface

```
//File: Archive.idl

typedef sequence<string> dbInfo;
```

```
typedef sequence<dbInfo> dbInfoSeq;

interface Archive {

    void showDB(out dbInfoSeq list);

    void updateDB(  in string uDate,
                    in string response,
                    in string message);

    void DeleteDB(in string dDate);

};
```

Appendix B

CD-ROM Guide

The attached CD-ROM contains the documentation, source code (compiled and not compiled) and other software components referred to in the report. The CD-ROM is organised as follows.

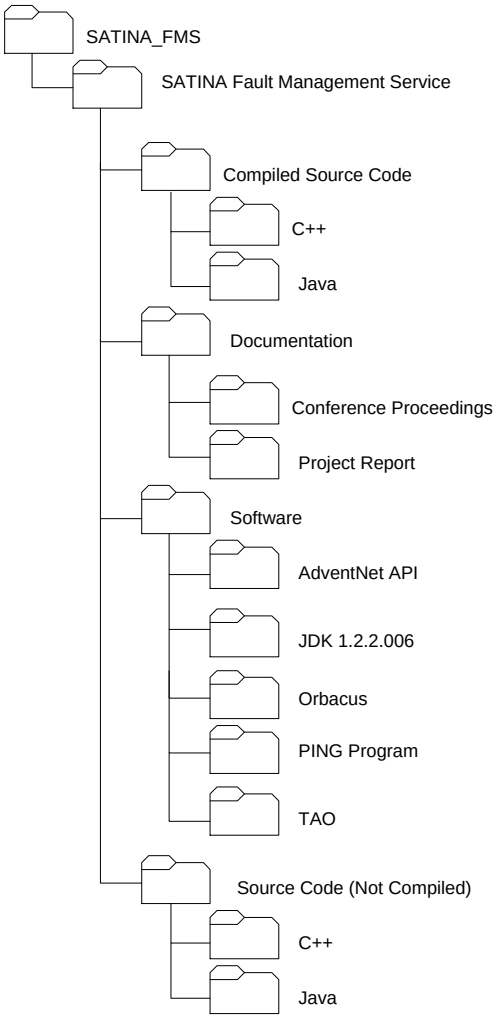

Figure B.1: Directory Structure of CD-ROM

The fully compiled source code for the Fault Management Service is stored in the *Compiled Source Code* folder. The compiled source code is divided into C++ and Java source code. The user of the Fault Management Service must use the same directory structure as used for the CD-ROM. If a different directory structure is used, the "Makefile" files must be changed to include the new paths for any shared libraries and IDL files. The folder also contains *README.txt* files for more information regarding the sub folders.

Under the *Documentation* folder, an electronic copy of the project report as well as a conference proceeding is provided. The documents are in pdf and postscript format.

The software used to implement this Fault Management System is provided in the *Software* folder. These components include the AdventNet API, Java Runtime Environment, ORBACUS, the PING functionality and TAO. In order to run the Fault Management Service, the following Software must first be installed.

The *Source Code (Not Compiled)* folder provides the developed components for this Fault Management Service. The components are the same as in the *Compiled Source Code* folder. These components are not compiled. The user is required to compile the files using the "Makefile" files. Please see the README.txt files under this folder to see compilation procedures.