HELSINKI UNIVERSITY OF TECHNOLOGY

Faculty of Electronics, Communicatios and Automation
Department of Communications and Networking

Ville Ryhänen

Converting CORBA Based Fault Management to SNMP

Master's Thesis submitted in partial fulfillment of requirements fo the degree of Master of Science in Technology in Espoo, Finland on May 15th 2009

Supervisor: Professor Jörg Ott

Instructor: M.Sc. Kaisa Kettunen

HELSINKI UNIVERSITY OF TECHNOLOGY **ABSTRACT OF THE MASTER'S THESIS**

Author: Ville Ryhänen

Name of the thesis: Converting CORBA Based Fault Management to SNMP

Date: 15.5.2009 **Number of pages:** 10+63

Faculty: Faculty of Electronics, Communications and Automation,
Department of Communications and Networking

Professorship: Networking Technology **Code:** S-38

Supervisor: Professor Jörg Ott

Instructor: M.Sc. Kaisa Kettunen

Fault management involves tasks to enable the detection, isolation and correction of abnormal operation of the telecommunication network. Telecommunications management network architecture of ITU-T consists of five layers, of which the bottom two, the element management layer and the network element layer, are focused on the management of network elements. For fault management tasks at these layers, several protocols have been utilised.

CORBA based fault management has been common in network elements and element management systems utilising solution sets of 3GPP. But as the telecommunications industry moves towards an all-IP world, SNMP has yet again become the predominant protocol for monitoring network elements. A network operator looking into unifying the fault management of its network could consider converting the CORBA based network element to using SNMP. This thesis studies the requirements and details of this kind of conversion. With a literary study, aspects of fault management and comparison of CORBA and SNMP are scoped for designing a CORBA-SNMP converter. A proof of concept for the conversion is obtained with a simplified implementation.

The implementation shows that a converter is quite easy to construct, and the converter can perform with operating principle of either CORBA or SNMP. The converter is also able to provide fault management unification without adding considerable delay, strain on bandwidth usage or consuming memory resources. The results of this thesis give grounds for studying the proposed concepts further and also broaden the converter to cover configuration management. Though for configuration management SNMP may not be the preferred protocol.

Keywords: Fault management, CORBA, SNMP, protocol conversion, converter

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä: Ville Ryhänen

Työn nimi: CORBA-pohjaisen vianhallinnan konversio SNMP:lle

Päivämäärä: 15.5.2009 **Sivumäärä:** 10+63

Laitos: Elektroniikan, tietoliikenteen ja automaation tiedekunta,
Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka **Koodi:** S-38

Työn valvoja: Professori Jörg Ott

Työn ohjaaja: DI Kaisa Kettunen

Vianhallinta pitää sisällään toimia, joilla havaitaan, eristetään ja korjataan tietoliikenneverkon epänormaaleja toimintoja. ITU-T:n tietoliikenteen hallintaverkkoarkkitehtuuri koostuu viidestä tasosta, joista kaksi alimmaista, elementinhallintataso sekä verkkoelementtitaso, keskittyvät verkkoelementtien hallintaan. Useita protokollia on hyödynnetty vianhallintaan näillä tasoilla.

CORBA-pohjainen vianhallinta on ollut yleinen 3GPP:n ratkaisusarjaa hyödyntävissä verkkoelementeissä ja elementinhallintajärjestelmissä. Mutta tietoliikenneteollisuuden siirtyessä kohti all-IP-maailmaa, SNMP on jälleen uudelleen vahvistamassa asemaansa hallitsevana verkkoelementtien monitorointiprotokollana. Täten verkkonsa vianhallinnan yhdenmukaistamista tutkiva verkko-operaattori voisi harkita CORBA-pohjaisten verkkoelementtiensä konvertointia käyttämään SNMP:tä. Tämä työ tutkii tämänkaltaisen konversion edellytyksiä ja yksityiskohtia. Kirjallisuustutkimuksella otetaan selvää vianhallinnan eri puolista ja vertaillaan CORBA:a ja SNMP:tä konvertterin suunnittelua varten. Todiste konversiokonseptin toimivuudesta saadaan yksinkertaistetun implementaation avulla.

Implementaatio osoittaa konvertterin olevan melko helposti rakennettavissa, ja että konvertteri voi toimia joko CORBA:n tai SNMP:n toimintaperiaatteella. Konvertteri mahdollistaa vianhallinnan yhdenmukaistamisen lisäämättä huomattavaa viivettä, rasitusta kaistaleveyskäytölle tai kuluttamatta runsaasti muistiresursseja. Tämän työn tulokset antavat aihetta esitettyjen konseptien laajemmalle tutkimukselle, sekä konvertterin laajentamiselle kattamaan konfiguraation hallinnan. Tähän tosin SNMP ei ehkä ole suositelluin protokolla.

Avainsanat: Vianhallinta, CORBA, SNMP, protokollakonversio, konvertteri

Preface

This thesis was written at Oy L M Ericsson Ab, Finland. At first finding a suitable research subject took a little time, but at the end I got a subject I felt real interest in. For this and for the very extensive guidance and support throughout my writing process, I want to thank my instructors Kaisa Kettunen and Leena Pitkäranta.

In the beginning, and at times during my writing, the completion of this thesis seemed like an insurmountable task, but going one step at a time and with rigorous proofreading by my instructors got me to the finish. Even though the writing took more time than originally planned, the end result from the added time was beneficial for the end result and helped the work of my supervisor, Professor Jörg Ott.

With this master's thesis I end the longest, and to this point the most significant part of my life – my university years. This time would certainly have been briefer without all the activities I partook, but the time would also have been many times less memorable. I will especially remember the many bus trips across the land, and would like to thank my travelling companions for the many gained memories and stories. Other big thanks go to my Driving School's headmaster Veikko Sompa and all his colourful students; you kept my driving skills up to date and with your invites to play board games aided me in extending my student years a little further.

Finally thanks to my parents for giving my support through my whole life, and to all the friends I've gained since my freshman year, for helping me to get my mind of my troubles when I felt stressed.

Espoo, May 15th 2009

Ville Ryhänen

Table of Contents

List of Figures

List of Tables

Abbreviations and Definitions

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
BML	Business Management Layer
CBC	Cipher Block Chaining
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DES	Data Encryption Standard
DN	Distinguished Name
DO	Distributed Object
EML	Element Management Layer
EMS	Element Management System
ESIOP	Environment Specific Inter-ORB protocol
FCAPS	Fault, Configuration, Accounting, Performance, Security
FM	Fault Management
HMAC	Hash Message Authentication Code
IDL	Interface Definition Language
IEC	International Engineering Consortium
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ITU-T	International Telecommunications Unions Telecommunications Standardization Sector

IRTF	Internet Research Task Force
MD5	Message-Digest algorithm 5
MIB	Management Information Base
MO	Managed Object
NE	Network Element
NEL	Network Element layer
NML	Network Management Layer
NMS	Network Management System
OID	Object Identifier
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
OS	Operations System
PDU	Protocol Data Unit
RAS	Reliability, Availability and Survivability
RDN	Relative Distinguished Name
RFC	IETF's Request For Comments
SHA-1	Secure Hash Algorithm 1
SMI	Structure of Management Information
SML	Service Management Layer
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
TL1	Transaction Language 1
TMN	Telecommunications Management Network

UDP	User Datagram Protocol
USM	Used-based Security Model
VACM	View Access Control Model
XML	Extensible Markup Language

1 Introduction

Telecommunications networks consist of interconnected network elements that communicate with various protocols and transport information with various transmission paths. Over the last decade the telecommunications networks have been in transition. Old networks were primarily designed for circuit-switched voice traffic and were relatively simple. They were based on copper loops for subscriber access and on a network of telephone exchanges to process calls. These networks have evolved into transporting voice, high-speed data, video, and every possible combination of these; they are now based on a variety of complex technologies.

The International Engineering Consortium describes an element management system (EMS) as a system that manages one or more of a specific type of telecommunications network element (NE) [IEC]. Typically, the EMS manages the functions and capabilities within each NE but does not manage the traffic between different NEs in the network. To support management of the traffic between itself and other NEs, the EMS communicates upward to higher-level network management systems (NMS) as described in the International Telecommunications Unions Telecommunications Standardization Sectors (ITU-T) telecommunications management network (TMN) layered model. In addition to the layered structure, the general management functionality splits into the five key areas of fault, configuration, accounting, performance, and security comprising the so-called FCAPS reference model.

The machine-to-machine communication protocol between the NE and its EMS varies from NE to NE and ranges from vendor-proprietary solutions to open standards such as CORBA or SNMP. Defined by the Object Management Group (OMG), the Common Object Request Broker Architecture (CORBA) is distributed middleware that can be used to manage a network element; it is for example used by the 3rd Generation Partnership Project (3GPP). Simple Network Management Protocol (SNMP) is part of the Internet Engineering Task Force's (IETF) internet protocol suite used in network management systems to monitor network devices for conditions that warrant administrative attention. Other widely used management interface protocols have been

Transaction Language 1 (TL1), Common Management Information Protocol (CMIP), and nowadays Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP). As telecommunications move toward an all-IP world and networks converge, the environment around NE may change and a need to change the used protocol may arise.

1.1 Objectives and Scope

This thesis will study how the conversion between a CORBA based management and an SNMP management can be done. Management of a network element involves all aspects depicted by the FCAPS model, but this thesis will concentrate on fault aspects, as they are important in all kinds of network elements. With this framing, the basic problem can be addressed simply, but the results can be broadened to cover the whole aspect of management of network elements and networks.

The thesis will find out which problems arise from interworking of two different ways of management implementation and how to solve the problems that occur. The study will show if this kind of conversion is practical and useful, and if the subject is case for further study. What benefits this kind of conversion brings will also be studied, and if these benefits are beneficial in other types of conversions or if they will become disadvantages.

1.2 Structure of the Thesis

This thesis uses literature study and constructive research by designing, implementing and testing a CORBA-SNMP converter. The first half is mainly based on literature study of books, articles and technical specifications.

The latter part deals with the design and implementation of the converter. The intended method for the conversion is to translate SNMP messages to a form that a CORBA based element will understand and also translate the elements responses and notifications to SNMP messages. The goal of the thesis is to study the possibilities to convert CORBA based control to SNMP so that the network elements inner behaviour need not to be changed and that the conversion can be done without completely

unpacking and repacking all management messages. The conversion will operate in both directions between a SNMP based managing system and CORBA based network element. The study will evaluate if this kind of conversion is economic and robust enough for service providers to use.

This thesis is structured as follows: Chapter 2 describes what management of network element entails, the CORBA architecture and the SNMP protocol. Chapter 3 explores the reasons for a conversion from CORBA to SNMP, and compares the differences and similarities of CORBA and SNMP fault management. In Chapter 4 the design and implementation of the converter is explained with testing and results. Chapter 5, as the last chapter, sums up the findings and includes discussion about the topics of future research.

2 Management of Network Elements

Network elements typically do not remain static through their life cycle in the way they are set up, nor does the network around them. Operators must be able to reconfigure their network nodes if they decide to make changes or in case a fault or some other event occurs. Management must be applied in these cases and this refers to the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems and elements [Raman-1].

ITU-T has defined a Telecommunications Management Network (TMN) in recommendations M.3000, M.3010, M.3200 and M3400 for managing open systems in a communications network. A TMN may provide management functions and offer communications both between Operations Systems (OS) themselves, and between OSs and the various parts of the telecommunications network [ITU3010]. A TMN may also provide management functions and offer communications to another TMN or TMN-like entities in order to support the management of international and national telecommunications networks. A telecommunications network consists of many types of analogue and digital telecommunications equipment and associated support equipment, such as transmission systems, switching systems, multiplexes, signalling terminals, front-end processors, mainframes, cluster controllers, and file servers. When managed, such equipment is generically referred to as network elements (NE).

The TMN architecture is a reference model for hierarchical telecommunications management approach. Its purpose is to partition the functional areas of management into layers. The key benefit of this architecture is to identify five functional levels of telecommunication management as depicted in Figure 1: business management layer (BML), service management layer (SML), network management layer (NML), element management layer (EML), and NEs in the network element layer (NEL). This distribution of management responsibilities makes it possible to spread these functions or applications over the multiple disciplines of a service provider and use different operating systems, different databases, and different programming languages [IEC].

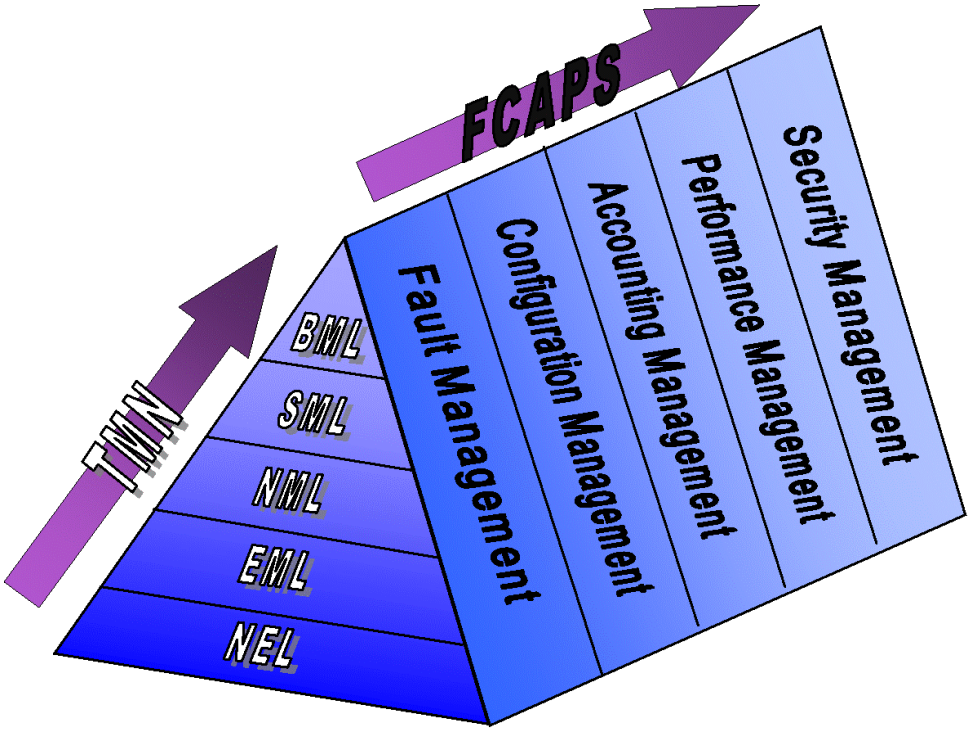

Figure 1: The Five-layer TMN architecture and FCAPS

As Figure 1 illustrates, ITU-T also splits the general-management functionality into five areas: Fault management, Configuration management, Accounting management, Performance management and Security management (FCAPS). This categorization is a functional one and stems directly from ITU-T recommendations describing the five different types of information handled by management systems. Portions of each of the functionalities will be performed at different layers of the TMN architecture. A subset of the FCAPS functionality is listed in Table 1.

Table 1: A Subset of the FCAPS Functionality [IEC]

Fault Management	Configuration Management	Accounting Management	Performance Management	Security Management
alarm handling	system turn-up	track service usage	data collection	control NE access
trouble detection	network provisioning	bill for services	report generation	enable NE functions
trouble correction	autodiscovery		data analysis	access logs
test and acceptance	back up and restore			
network recovery	database handling			

2.1.1 Fault Management

The TMN M.3400 recommendation describes fault management (FM) as a set of functions which enables the detection, isolation and correction of abnormal operation of the telecommunication network and its environment [TMN3400]. The concept of a fault is central to the definition of fault management. A fault is usually indicated by failure to operate correctly or by excessive errors. There could be various types of faults related to a NE: faulty hardware; software failures such as software related bugs, incompatibility with hardware; congestion problems such as overload and threshold condition at the NE; or communication failure between the NE and EMS [Agrawal]. A fault can also be defined by its difference from an error; a fault is an abnormal condition that requires management attention to repair, whereas an error is a single event [Stallings-1].

Fault management includes the following function set groups in the TMN M.3400 recommendation:

- Reliability, Availability and Survivability (RAS) Quality Assurance
- Alarm Surveillance
- Fault Localization
- Fault Correction
- Testing
- Trouble Administration

As can be seen in Table 1, other divisions also exist, but they cover basically the same areas.

RAS quality assurance establishes the reliability criteria that guide the design policy for redundant equipment for managing availability and outage reporting [TMN3400]. With it, an operator can determine how much redundancy is needed in the managed network nodes and what the network must report when a node or service goes offline.

Alarm surveillance or fault detection is about monitoring NE failures in near-real time. When such a failure occurs, an indication is made available by the NE. Based on this a service provider (SP) determines the nature and severity of the fault. Alarm information can be reported at the time of occurrence, and/or logged for future access. An alarm may also cause further management actions within the NE that lead to the generation of other fault management data [TMN3400].

Where the initial failure information is insufficient for fault localization, it has to be augmented with information obtained by additional failure localization routines employed by service providers tests [TMN3400]. Fault localization requires communication between nodes to determine where the failure occurred. Based on the fault information received, fault diagnosis is done to determine the root cause of the failure [Hanemann]. Fault correction transfers data concerning the repair of a fault. To replace equipment or facilities that have failed service provider utilises procedures that put redundant resources in use.

In addition to passive failure detection, a service provider can also perform proactive tests. These tests can either deal with resources or can assume the role of virtual

customer and test a service by performing interaction at the service access point [Hanemann]. Testing dealing with resources can be carried out in one of two ways [TMN3400]. In one case, a service provider directs a given NE to carry out analysis of circuit or equipment characteristics. Processing is executed entirely within the NE and the results are automatically reported to the service provider, either immediately or on a regular delay. Another method is to carry out the analysis within the SP side. In this case the SP merely requests that the NE provide access to the circuit or equipment of interest and no other messages are exchanged with the NE.

Trouble administration transfers trouble reports originated by customers and trouble tickets originated by proactive failure detection checks. The aim is for the service provider to identify and react to problems in its offered services before a customer notices them [Hanemann]. The probing can be done from a customer point of view or by testing the resources which are part of the service.

In essence fault management contains facilities that enable the detection, isolation and correction of network problems. Furthermore, it may use trend analysis to predict faults so that the availability of the network is maximized. This can be established by monitoring networks for abnormal behaviour. When a fault or event occurs, a network component will send a notification to the network operator using a management protocol. The TMN M.3010 recommendation allows for the use of multiple protocols to be used for management. This means that open standard such as SNMP and CORBA are consistent with the TMN framework as is its initial management protocol CMIP [IEC]. Each protocol executes the management in its own way but the principles remain the same.

2.2 Element Management System

An element management system (EMS) is used to manage one device or a set of devices connected in a network. The EMS communicates with network devices directly using network management protocols – an efficient EMS communicates with its NE using whatever protocol is native to the NE. Functions usually expected from an EMS can be listed as [Misra]:

- provide common information about the NE, such as system up-time and system name,
- monitor the operational state of the NE, such as start-up and shut-down,
- monitor the occurrence of NE malfunctions,
- gather and process NE performance information,
- enable configuration of the NE,
- carry out remote operations on the NE.

The EMS is a critical piece in the total telecommunications management solution [IEC]. Only the EMS can access the complete management information content of all the NEs in its management domain. As Figure 2 shows, the EMSs are the media that transmit the network elements' management information and control to the network management layer and to the network management systems.

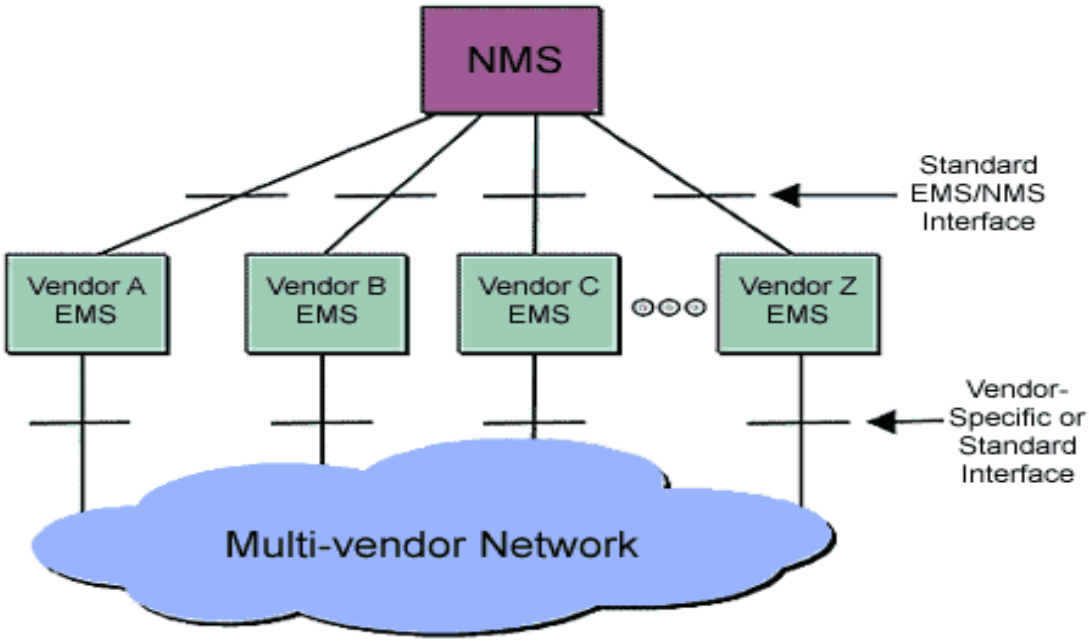

Figure 2: Position of the EMSs in the Telecommunications Network [IEC]

According to the OSI Management architecture – which is the basis for many modern network management systems [Stallings-1] – EMSs and NEs reside in a management domain [Klerer]. A management domain may be decomposed into one or more management systems, and zero or more managed elements. An object represents an abstraction of a bundle of data and instructions in object-oriented world. A managed

element, such as a network element, may be decomposed into one or more managed objects (MO). MOs depict network element's management information that represents resources and their activities. A managed object presents a view of the resource to the management system with properties that are manageable [Raman-2]. A resource such as an ATM board provides interfaces for ATM traffic, however, only some aspects of the board are manageable by a management system. These properties are reflected in the managed object that represents the board. A management system, such as EMS illustrated in Figure 3, is an application process within a management domain witch effects monitoring and control functions on managed objects [Klerer].

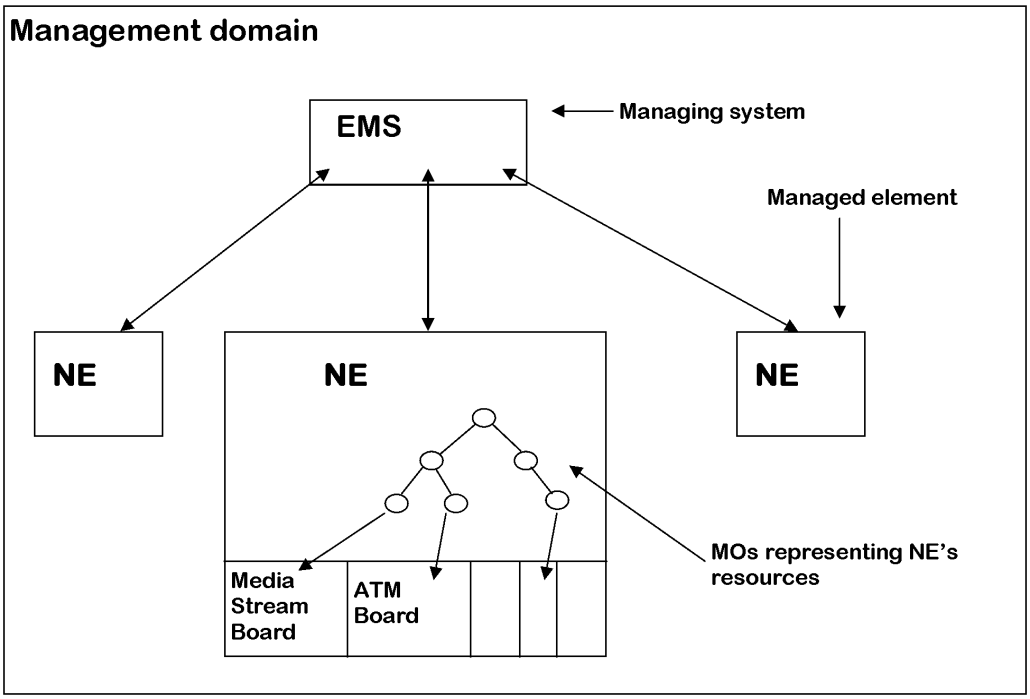

Figure 3: Relationships of managed objects, managed elements and managing system

2.2.1 Fault Management at the Element Management Layer

In the TMN model an EMS is placed on the element management layer. Fault management at the EML is about logging each discrete alarm or event in detail. Most faults will be detected by the NE and reported to the EMS as notifications or alarms. By periodically polling an NE an EMS can detect the communication failure with the NE, in which case a notification is generated at the EMS. The EMS filters the alarms and

forwards them to an NMS that performs alarm correlation across multiple nodes and technologies to perform root-cause analysis [IEC]. EMS stores the list of currently active alarms at NEs, and EMS removes the alarm from active list whenever a notification to clear the alarm is received. EMS also maintains history of the alarms.

Fault management at the EML can be executed by a number of protocols. The following sections detail two of the most prominent ones: CORBA and SNMP. A brief view of other fault management methods is also included.

2.3 CORBA

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables distributed software components in a network to work together. CORBA allows applications to talk to each other even if they are on different computers, on different operating systems, on different CPU types, or implemented with different programming languages [McHale].

The components of CORBA are Object Management Architecture (OMA), Object Request Broker (ORB) to support interaction between objects, and object services. The object interfaces are specified in a notion called Interface Definition Language (IDL). CORBA 2.0 introduced protocols for building interoperable ORBs to the architecture. The General Inter-ORB Protocol (GIOP) is connection-oriented. A specialization of this protocol for use with TCP/IP Internet Suite has been defined in IIOP. Environment Specific Inter-ORB protocols (ESIOP) have been defined for interfacing with platforms that do not support CORBA. [Raman]

2.3.1 The Concept of an Object in CORBA

A central concept in CORBA is its version of the managed object. A CORBA managed object is not quite identical to an object in a programming language; they generally share characteristics even though all sets of objects do not intersect [Pope]. The type of a CORBA object is called an interface, which is similar in concept to a C++ class or a Java interface. An object has methods, state, and a characteristic behaviour. Objects are

a physical manifestation of a class. A CORBA object is an instance of a class encapsulating operations, attributes, and exceptions. CORBA also admits the possibility of types that are not objects and types that the OMG documents call pseudo-objects. An object is a basic computational unit consisting of a defined behaviour and perhaps some attributes. The attributes retain the effect of behaviour. Requests made on an object are messages or methods. The visible part of an object is its interface. An interface to an object is the combined sum of the messaging protocols used to request services.

As distributed middleware CORBA envisions distributed objects (DO) as the union of concepts from two paradigms – distributed computing and object-orientation, with some explicit differences [Pope]:

- A client knows an object by its interface.
- Objects are not always local with respect to their clients.
- Dynamic composition may compose objects into new applications.
- Objects hide many of the underlying differences in architecture through encapsulation.

In summary, distributed objects offer benefits of object-orientation and client-server: the ability to distribute risk, rightsizing system development with small combinable subtasks, and having looser coupling with well-defined integration [Pope].

3GPP describes a Managed Object (MO) as a software object that encapsulates the manageable characteristics and behaviour of a particular network resource and uses this description in its CORBA solution set [3GPP-1068]. MOs are organised in hierarchical Managed Object Model similar to that of a file system, where an MO that contains another one is referred to as the superior (parent), whereas the contained MO is referred to as the subordinate (child). More of this will be covered in Chapter 3.3.

2.3.2 Object Management Architecture (OMA)

In CORBA, a managed object is an object that is subject to system wide administration and control. It is a client of services, such as activation, installation, or dynamic behaviour [Pope]. These managed objects are manifested either as an application object, an object facility, or an object service.

The object management architecture is composed of two aspects and uses object-oriented design concepts: a core model that describes the principles for defining objects along with their properties and interfaces; and a reference model with four components: object request broker, object services, common facilities, and application objects. The last two components use the object services as building blocks. The common facilities are higher level services that may be used by several applications. [Raman]

2.3.3 Object Request Broker (ORB)

The foundation of CORBA is the object request broker which is the mechanism for objects to interact with each other. Figure 5 in Paragraph 2.3.6 illustrates a request sent by a client to an object implementation. The client is the entity that wishes to perform an operation on the object and the object implementation is the code and data that actually implements the object [OMG3.0.3].

When an object in the client role invokes an operation, the request is processed by an ORB to identify the server object to perform the request [Raman]. The client is not aware of either the location or implementation details of the server object. The client makes the request using the object reference. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request [OMG3.1]. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect that is not reflected in the interface of the object. In the client-server model an object plays roles of a client and a server and an object may assume both client and server roles for different operations.

When considering security aspects of CORBA, the ORB, by itself, offers only trivial security without the aid of an underlying secure infrastructure. It enables only a very minimal security of being able to check that parameters meet requirements and that the correct target receives a request. The design of security is flexible to the point of being

able to support a wide variety of security mechanisms from the network and hardware.

2.3.4 Object Services

The core part of CORBA is of limited use by itself, in the same way that a programming language stripped of its standardized library is of limited use. What greatly enhances the power of CORBA is a standardized collection of object services – called CORBA Services – that provide functionality useful for the development of a wide variety of distributed applications [McHale]. The CORBA Services have APIs that are defined in IDL. The CORBA Services are a distributed, standardized class library. The object services identified and standardized include [Raman]: Event Service, Life Cycle Service, Naming Service, Persistence Service, Concurrence Control Service, Externalization Service, Relationship Service, Transaction Service, Security Service, Time Service, Query Service, Licensing Service, Trader Service, Change Management Service, Start-up Service, Properties Service, and Topology Service. For example, Security Service is defined in a way that is independent of any particular security technology and developers are given freedom from underlying security technologies used in a given system without needing to redesign the applications. The architecture allows new services to be identified if desired.

2.3.5 Interface Definition Language (IDL)

The object interfaces in CORBA are defined using a language called IDL. The structure of IDL is simple and the method of defining data types and interfaces is very similar to writing programs in terms of data declarations and function calls [Raman]. IDL is an object contract language, but IDL is not a complete programming language, it has no iterators or flow control [Pope]. It is primarily a language in which one can express complex interfaces, but it does not provide implementation for interfaces.

IDL defines data types such as integer, character string, and enumerated. It is also possible to define new types using the struct, union, and sequence constructs. The operations interfaces include name, parameters, result, and exceptions. The IDL definitions including type definitions, constant deceleration, and interface definitions

may be combined into one or more modules. [Raman]

IDL is used to define the public API that is exposed by objects in a server application. IDL defines this API in a way that is independent of any particular programming language. However, for CORBA to be useful there must be a mapping from IDL to one or a number of particular programming languages. Clients are not written in OMG IDL, which is purely a descriptive language, but in languages for which mappings from OMG IDL concepts have been defined. The CORBA standard currently defines mappings from IDL to the following programming languages [McHale]: C, C++, Java, Ada, Smalltalk, COBOL, PL/I, LISP, Python and IDLScript. These officially-endorsed language mappings provide source-code portability of applications across different CORBA products. There are also proprietary mappings, but they will not guarantee source-code portability to other CORBA vendor products [McHale]. Typically, a CORBA implementation comes with a tool called an IDL compiler which converts the user's IDL code into some language-specific generated code. A language mapping requires the developer to create some IDL code that represents the interfaces to his objects. A traditional compiler then compiles the generated code to create the linkable-object files for the application. Figure 4 illustrates how the generated code is used within the CORBA infrastructure.

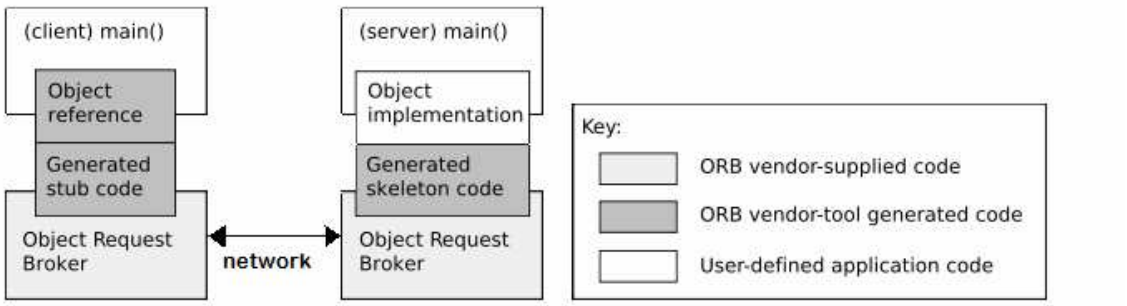

Figure 4: Autogeneration of the infrastructure code from an interface defined using the IDL

2.3.6 Interoperability and General Inter-ORB Protocol (GIOP)

ORB interoperability specifies a comprehensive, flexible approach to supporting networks of objects that are distributed across and managed by multiple, heterogeneous

CORBA-compliant ORBs. General Inter-ORB Protocol (GIOP) is the abstract protocol by which ORBs communicate. This protocol defines the different message types – such as request and reply messages – that can be exchanged between client and server applications and also specifies a binary format for the on-the-wire representation of IDL types.

GIOP does not specify the actual networking technology that is used to transmit messages between clients and servers. For example, GIOP does not specify if messages should be transmitted over TCP/IP, X.25, ATM or some other transport. Instead, the choice of transport mechanism is decided in a specialization of GIOP. The most well-known GIOP specialization is the Internet Inter-ORB Protocol (IIOP), which is for use on TCP/IP networks. All CORBA products are obliged to support IIOP, but they may optionally also support other GIOP-based protocols or environment specific inter-ORB protocols (ESIOP). Figure 5 illustrates the inter-ORB protocol relationship. An interoperable object reference (IOR) contains the contact details for all the protocols that clients can use to communicate with an object in a server. [McHale]

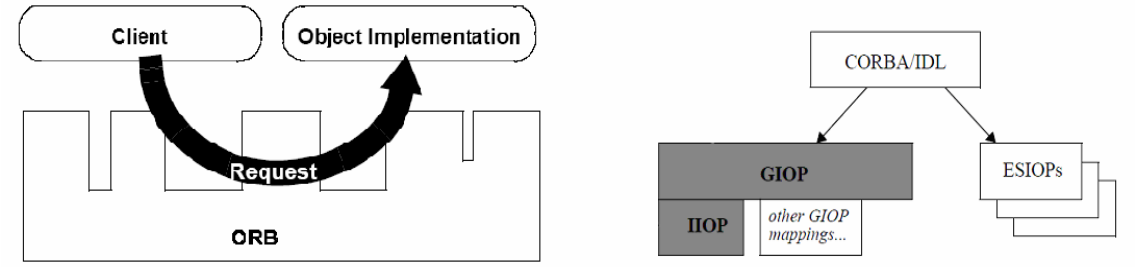

Figure 5: A request being sent through the ORB and Inter-ORB protocol relationship [OMG3.0.3]

The GIOP specifies eight message types that can be transmitted between client and server applications. The GIOP message types are: Request, Reply, Fragment, CancelRequest, CloseConnection, MessageError, LocateRequest and LocateReply.

Table 2: GIOP Messages

TypeName	Originator	Notes
Request	Client	
Reply	Server (i.e. Implementation)	
Fragment	both	remaining pieces of a large Request/Reply
CancelRequest	Client	sent when client gets a timeout exception
CloseConnection	Server	sent before closing the socket connection
MessageError	both	normally sent to non-CORBA applications
LocateRequest	Client	similar to “ping”, “is the object here?”
LocateReply	Server	reply to “ping”

Figure 6 describes an example the GIOP request message and its headers. The fields in the GIOP header can be described as follows:

- The four characters "GIOP" serve to identify the protocol.
- The GIOP version number (major and minor) is used to create the message.
- A flag byte is currently only used to indicate the byte ordering.
- An integer is used to indicate the message type.
- The message size (excluding the GIOP header itself).

This summarises all information which is sent to the GIOP header. The request message consists of a Request header followed by a Request body. The header consists of the following fields:

- The `service_contexts` field allows service specific context information to be passed along with a Request. Intended for use in conjunction with the CORBA services to carry extra information along with the Request, the service contexts are not needed in the core specification of CORBA.

- The `request_id` field is used to uniquely identify a Request emanating from a client so that the client can later match a received Reply with its corresponding Request (the corresponding Reply is tagged with the same `request_id`).

- The `response_expected` flag is used to indicate whether the Request is one-way or not. A normal Request has `response_expected` set equal to TRUE.

- The next field is an array of three bytes reserved for future use.

- The `object_key` field is used at the server end to identify the object which is being invoked.

- The operation field is simply a string giving the name of the operation being invoked.

- The `requesting_principal` field identifies the user making the request. That is, it is simply the user name of the person running the client.

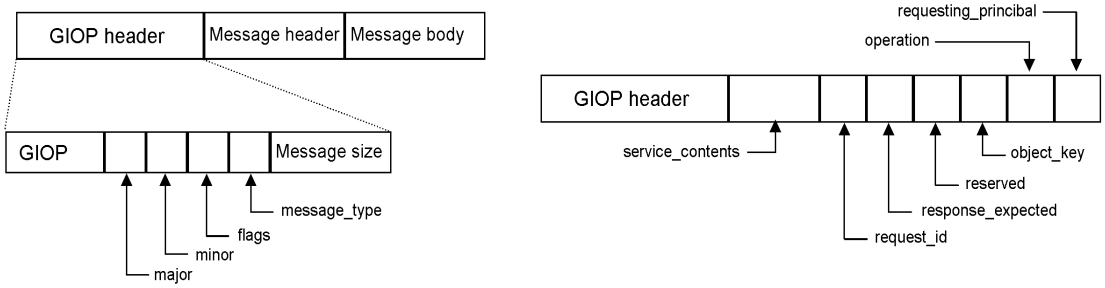

Figure 6: The format of a GIOP message and message header, and the format of a Request message header

2.3.7 CORBA and Fault Management

According to OMG many different kinds of applications, developed by the members of the OMG and the users of CORBA, have a need for fault tolerance [OMG3.0.3]. The standard for Fault Tolerant CORBA aims to provide robust support for applications that require a high level of reliability, including applications that require more reliability than can be provided by a single backup server. The standard requires that there shall be no single point of failure.

In CORBA fault tolerance depends on entity redundancy, fault detection, and recovery. The entity redundancy means the replication of objects. This strategy allows greater flexibility in configuration management of the number of replicas, and of their assignment to different hosts, compared to server replication. Replicated objects can invoke the methods of other replicated objects without regard to the physical location of those objects. Support for redundancy in time is provided by allowing clients to make repeated requests on the server, using the same or alternative transport paths.

In a fault-tolerant CORBA system, fault management encompasses the following activities [OMG3.0.3]:

- Fault detection - detecting the presence of a fault in the system and generating a fault report.
- Fault notification - propagating fault reports to entities that have registered for such notifications.
- Fault analysis/diagnosis - analysing a (potentially large) number of related fault reports and generating condensed or summary reports.

In the fault tolerance infrastructure, fault detectors detect faults in the objects, and report faults to the fault notifier. The fault notifier receives fault reports from the fault detectors, filters the reports, and propagates the filtered reports as fault event notifications to consumers that have subscribed for them. The fault analyser reasons about the received fault reports, and produces aggregate or summary fault reports. It propagates these reports back to the fault notifier for dissemination to other consumers.

Typically, there are several fault detectors, including those provided by the

infrastructure to monitor objects, and other fault detectors provided by the infrastructure or the application. Each fault detector belongs to a particular fault tolerance domain, and is not shared across fault tolerance domains. Most implementations of fault detectors are based on time-outs, and use either pull- or push-based monitoring. There can be also one or more fault analysers.

A problem with the CORBA fault notification is the potential for a large number of notifications to be generated by a single fault [OMG3.0.3]. This problem is addressed by filtering within the fault notifier, by fault analysers, and by the `FaultMonitoringGranularity` interface.

Figure 7 illustrates the architecture of fault tolerance infrastructure - the interaction between the fault detectors, fault notifier, fault analyzer, and replication manager in a relatively simple system.

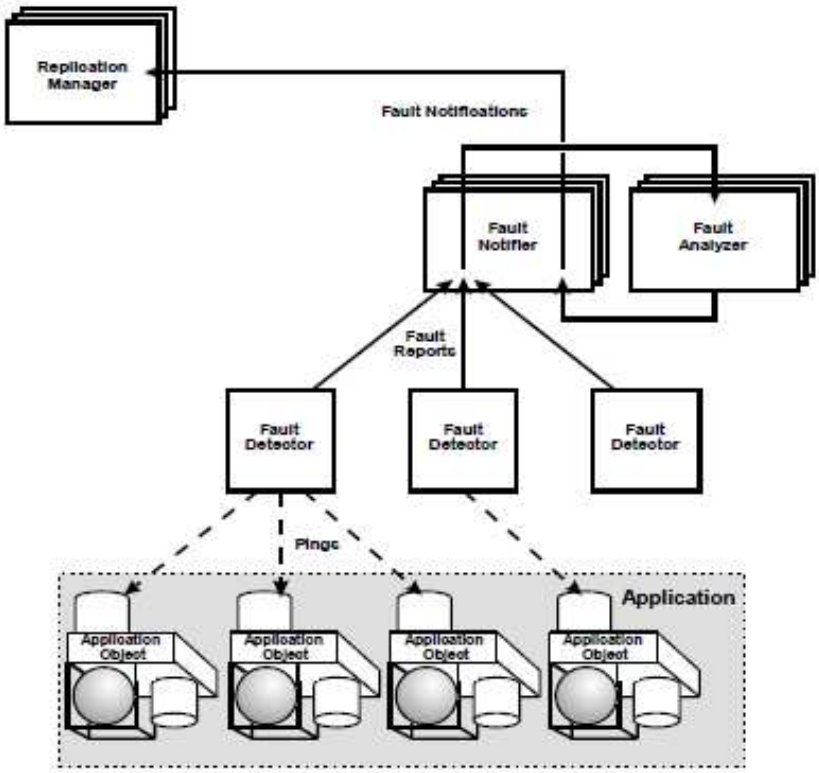

Figure 7: Interaction between the fault detector, fault notifier, fault analyzer and replication manager [OMG3.0.3]

2.4 SNMP

SNMP stands for Simple Network Management Protocol and is currently in its third version – although versions one and two are also still in some use. SNMP is standardized by the Internet Engineering Task Force (IETF) and it is the standard network management protocol in the IP realm. Each version of SNMP is specified by one or more IETF Request for Comments (RFC): RFC 1157 defines SNMP Version 1 (SNMPv1), RFCs from 3416 to 3418 define SNMP Version 2 (SNMPv2) and RFCs from 3410 to 3418 and RFC 2576 define SNMP Version 3 (SNMPv3).

The initial version of SNMP protocol is nowadays a historical IETF standard; although SNMPv1 is historical, it is still widely supported by many vendors. The security of SNMPv1 is based on communities, which are nothing more than passwords. SNMPv2 expanded the functionality of SNMP and broadened its applicability from only TCP/IP-based networks to also include OSI-based networks. The key enhancements that SNMPv2 provides to version 1 fall into three following categories: structure of management information (SMI), manager-to-manager capability, and protocol operations. SNMPv3 addresses the security problems of the previous versions, but no other essential changes are made to the protocol [Mauro]. There are several new textual conventions, but these are really just more precise ways of interpreting the data types defined in previous versions.

For gathering management information SNMP uses simple messages called protocol data units (PDU). SNMP uses user datagram protocol (UDP) to transmit PDUs. The SMI provides a way to define the managed objects of SNMP and their behaviour, and the management information base (MIB) can be thought as a database of managed objects.

2.4.1 Basic SNMP Concepts

In SNMPv1 and SNMPv2 there are two kinds of entities: managers and agents. A manager is a server running a software system handling management tasks for a

network. An agent is a piece of software that runs on the managed network device. It can be a separate program or it can be incorporated into the operating system; most IP devices today have some kind of SNMP agent build in [Mauro]. The agent is responsible for:

- Collecting and maintaining information about its local environment
- Providing that information to a manager, either in response to a request or in an unsolicited fashion when something noteworthy happens
- Responding to manager commands to alter the local configuration or operating parameters

The manager station generally provides a user interface so that a human network manager can control and observe the management process. This interface allows the user to issue commands (for example deactivate a link, collect statistics on performance) and provides logic for summarizing and formatting information collected by the system. SNMPv3 makes a big change to the basic concepts and abandons the notion of managers and agents, both are now called SNMP entities. Chapter 2.4.2 elaborates this change in more detail.

2.4.1.1 The Structure of Management Information (SMI)

The structure of management information defines the general framework how managed objects and their resources are named and represented and specifies their associated data types. The definition of managed object in SMI can be broken down into three attributes [Mauro]:

- **Name:** the name, or object identifier (OID), uniquely defines an MO. Names commonly appear in two forms: numeric and "human readable." In either case, the names are long and inconvenient. In SNMP applications, a lot of work goes into helping the user to navigate through the namespace conveniently.
- **Type and syntax:** A data type of an MO is defined using a subset of Abstract Syntax Notation One (ASN.1). ASN.1 is a way of specifying how data is represented and transmitted between entities, within the concept of SNMP. The benefit of ASN.1 is that it is machine independent and different machines can communicate without

worrying about such as byte ordering.

- **Encoding:** A single instance of an MO is encoded into a string of octets using Basic Encoding Rules (BER). BER defines how the objects are encoded and decoded so that they can be transmitted over a transport medium such as Ethernet.

2.4.1.2 Management Information Base (MIB)

All MOs in the SNMP environment are arranged in a hierarchical or tree structure. The leaf objects of the leaf are the actual MOs, each of which represents some resource, activity, or related information that is to be managed. The tree structure itself defines a grouping of objects into logically related sets. Figure 8 illustrates the MIB-II sub tree, which is a very important management group because every device that supports SNMP must also support MIB-II [Mauro]. The MIB-II standard defines variables for things such as interface statistics (interface speeds, maximum transmission unit, octets sent, octets received, and so on) as well as various other things pertaining to the system itself (system location, system contact, and so on). The main goal of MIB-II is to provide general TCP/IP management information; it does not cover every possible item a vendor may want to manage within its particular device.

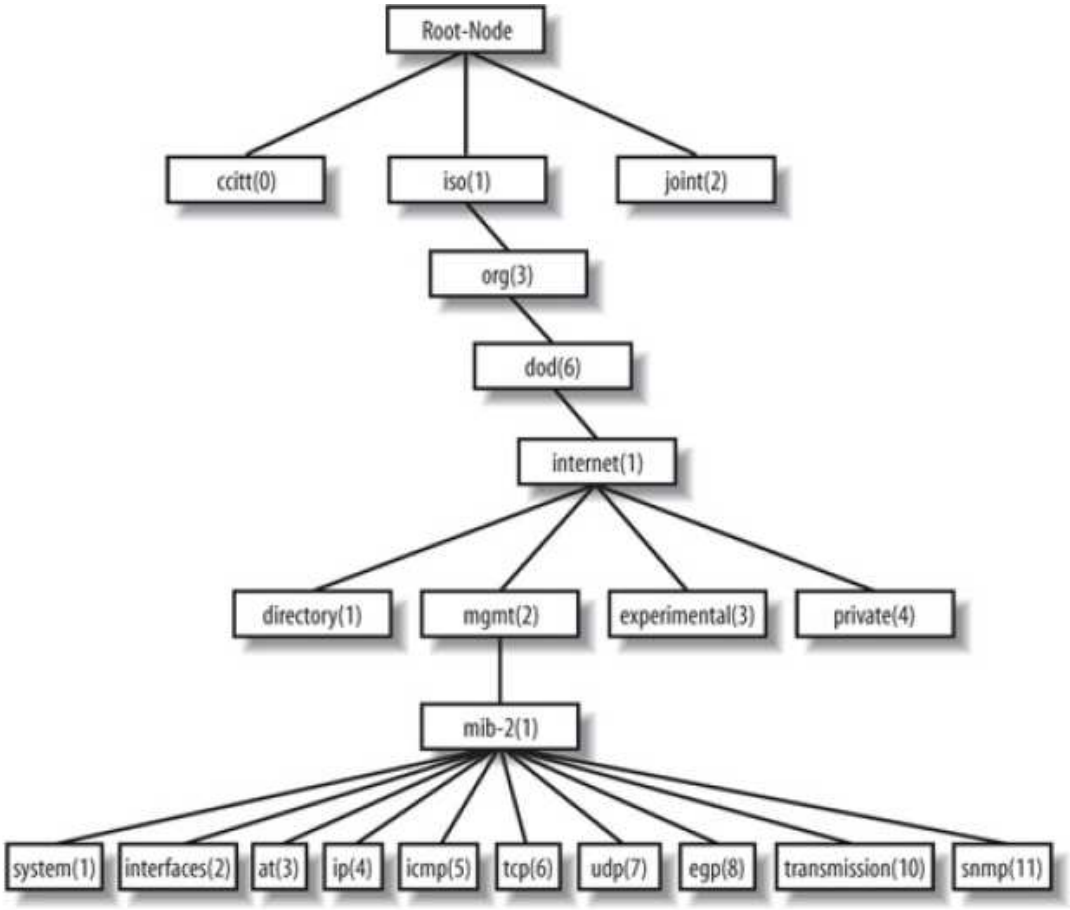

Figure 8: MIB-II sub tree of the mgmt branch [Mauro]

What gives SNMP its power is the extensive set of standardized MIB structures that has been defined [Stallings-2]. The MIB at an agent-entity dictates what information that agent will collect and store. For example, there are a number of variables in the basic MIB that relate to the operation of the underlying TCP and IP protocols, including number of packets sent and received, packets in error, and so on. Since all agents maintain the same set of data variables, applications can be written at the management station to exploit this information.

2.4.1.3 SNMP Operations

SNMP is designed to be easy to implement and to consume minimal processor and network resources. It is therefore a tool for building a bare-bones management facility. Entities send and receive information by PDUs. Each of the following SNMP operations

has a standard PDU format:

- Get: Used by a manager to retrieve an item from an agent's MIB.
- GetNext: Used by a manager to traverse a MIB sub-tree in lexicographic order.
- GetBulk: Used by a manager to retrieve a large section of a table from an agent's MIB at once.
- Set: Used by a manager to set a value in an agent's MIB.
- GetResponse: Used by an agent to respond to manager’s get, getnext or getbulk operation.
- Trap: Used by an agent to send an alert to a manager.
- Notification: Used to standardize the PDU format of traps.
- Inform: Used by a manager to send an alert to another manager.
- Report: Allows SNMP engines to communicate with each other.

2.4.2 Protocol Architecture

Each SNMP entity consists of an SNMP engine and one or more SNMP applications. The revised concepts are important because they define an architecture rather than simply a set of messages; the architecture helps separate different pieces of the SNMP system making a secure implementation possible [Mauro].

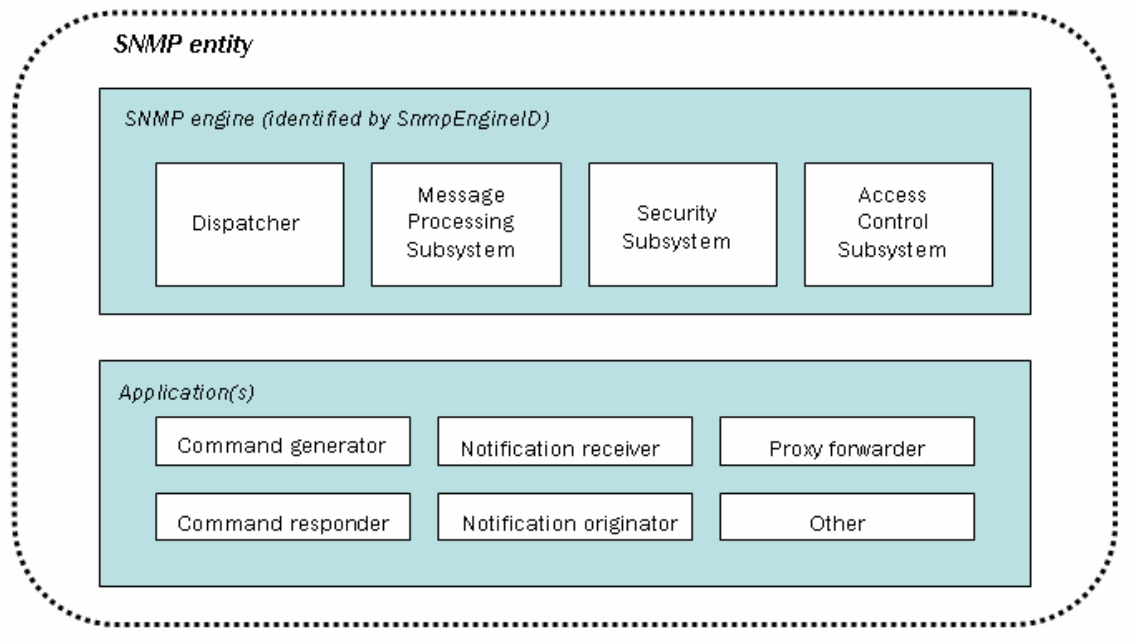

Figure 9: SNMPv3 entity [Mauro]

2.4.2.1 The SNMPv3 Engine

The engine is composed of four pieces: the dispatcher, the message processing subsystem, the security subsystem, and the access control subsystem. An SNMP engine implements functions for sending and receiving messages, authenticating and encrypting or decrypting messages, and controlling access to managed objects [Stallings-1].

The job of the dispatcher is to send and receive messages. The dispatcher determines the version of each received message and hands it to the message processing subsystem. The dispatcher also sends SNMP messages to other entities. The message processing subsystem prepares messages to be sent and extracts data from received messages. An implementation of the message processing subsystem may support a single message format corresponding to a single version of SNMP, or it may contain a number of modules, each supporting a different version of SNMP. The security subsystem provides authentication and privacy services. Each outgoing message is passed to the security subsystem from the message processing subsystem. Depending on the services required, the security subsystem may encrypt the enclosed message, and it may generate

an authentication code and insert it into the message header. The processed message is then returned to the message processing subsystem. Similarly the security subsystem checks incoming messages for authentication code and performs decryption. Authentication uses either community strings (SNMPv1 and v2) or SNMPv3 user-based authentication, that uses the MD5 or SHA algorithms to authenticate without sending a password in the clear. The privacy service uses the DES algorithm to encrypt and decrypt SNMP messages. The access control subsystem is responsible for controlling access to management information base objects. [Mauro]

2.4.2.2 The SNMPv3 Applications

SNMPv3 divides the traditional manager and agent roles of previous SNMP version into a number of applications presented in Table 3. All applications make use of the services provided by the SNMP engine of an entity. RFC 3411 allows additional applications to be defined over time.

Table 3: SNMPv3 Applications

Application	**Description**	**Traditional role**
Command generator	Generates get, getnext, getbulk, and set requests and processes the responses. Implemented by an NMS.	manager
Command responder	Receives and responds to get, getnext, getbulk, and set requests.	agent
Notification originator	Monitors a system for particular events or conditions, and generates SNMP traps and notifications. A notification originator must have a mechanism for determining where to send messages, and which SNMP version and security parameters to use.	manager and agent

Notification receiver	Listens for notification messages, receives traps and inform messages, and generates response messages to them.	manager
Proxy forwarder	Forwards messages between entities.	agent

2.4.3 Messages

In all versions of SNMP information is exchanged between SNMP entities with messages. Each message includes a message header and a PDU. SNMPv3 message format is illustrated in Figure 10.

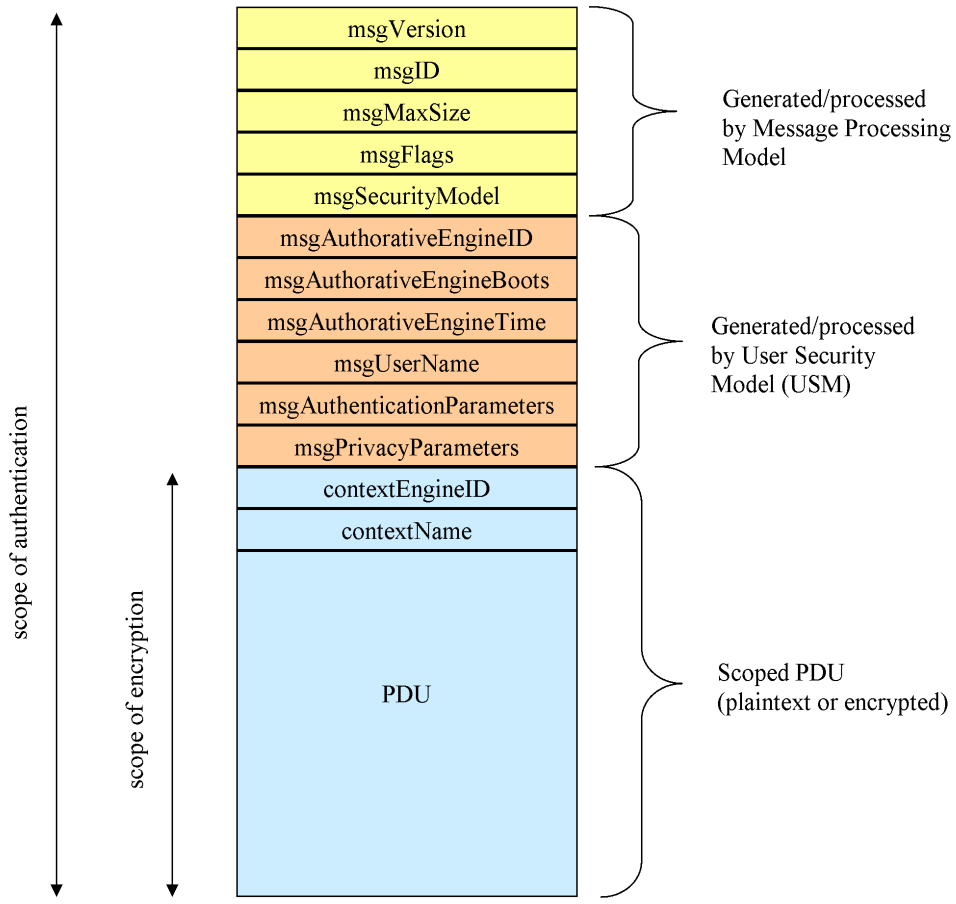

Figure 10: SNMPv3 message format with USM

The user-based security model (USM) and the view access control model (VACM) together detail the security enhancements added with SNMPv3. RFC 2274 defines the USM and this specification encompasses [Stallings-1]:

- Authentication: Provides data integrity and data origin authentication. The message authentication code HMAC, with either the hash function MD5 or SHA-1, provides authentication.
- Timeliness: Protects against message delay or replay.
- Privacy: Protects against disclosure of message payload. The cipher block chaining (CBC) mode of DES is used for encryption.
- Message format: Defines format of `msgSecurityParameters` field, which supports the functions of authentication, timeliness, and privacy.
- Discovery: Defines procedures by which one SNMP engine obtains information about another SNMP engine.
- Key management: Defines procedures for key generation, update, and use.

Specifically, USM is designed to secure against the following principal threats [Stallings-2]:

- Modification of information: An entity could alter an in-transit message generated by an authorized entity in such a way as to effect unauthorized management operations, including the setting of object values. The essence of this threat is that an unauthorized entity could change any management parameter, including those related to configuration, operations, and accounting.
- Masquerade: Management operations that are not authorized for some entity may be attempted by that entity by assuming the identity of an authorized entity.
- Message stream modification: SNMP is designed to operate over a connectionless transport protocol. There is a threat that SNMP messages could be reordered, delayed, or replayed (duplicated) to effect unauthorized management operations. For example, a message to reboot a device could be copied and replayed later.
- Disclosure: An entity could observe exchanges between a manager and an agent

and thereby learn the values of managed objects and learn of notifyable events. For example, the observation of a set command that changes passwords would enable an attacker to learn the new passwords.

USM does not secure against denial of service: an attacker may prevent exchanges between SNMP entities, or traffic analysis: an attacker may observe the general pattern of traffic between entities.

In any message transmission, either the transmitter or receiver is designated as the authorative SNMP engine. When an SNMP message contains a payload, which expects a response (for example, a Get-, GetNext-, GetBulk-, Set-, or Inform-PDU), then the receiver of such messages is authoritative; when an SNMP message contains a payload, which does not expect a response (for example, an SNMPv2-Trap-, Response-, or Report-PDU), then the sender of such a message is authoritative. This designation serves two purposes [Stallings-2]:

1. The timeliness of a message is determined with respect to a clock maintained by the authoritative engine. When an authoritative engine sends a message (Trap, Response, Report), it contains the current value of its clock, so that the non-authoritative recipient can synchronize on that clock. When a non-authoritative engine sends a message (Get, GetNext, GetBulk, Set, Inform), it includes its current estimate of the time value at the destination, allowing the destination to assess the timeliness of the message.

2. A key localization process enables a single principal to own keys stored in multiple engines; these keys are localized to the authoritative engine in such a way that the principal is responsible for a single key but avoids the security risk of storing multiple copies of the same key in a distributed network.

Access control is a security function performed at the PDU level. An access control document defines mechanisms for determining whether access to a managed object in a local MIB by a remote principal (which may be an individual or an application or a group of individuals or applications) should be allowed [Stallings-2]. The view-based

access control model is defined in RFC 2275 and used to control access to managed objects in a MIB or MIBs. VACM makes use of a MIB that defines the access policy for this entity, and makes it possible for remote configuration to be used. VACM implements the services required for the access control subsystem. VACM makes an access control decision on basis of the principal asking for access, the security model and security level used for communicating the request of the principal, the context to which access is requested, the type of access requested (read, write, notify), and the actual object for which access is requested [Stallings-1].

2.5 Other Methods of Fault Management

Despite the current strong presence of CORBA and SNMP, there are also other methods for fault management in the telecommunications field. The nowadays somewhat outdated common management information protocol (CMIP) is still used to manage telecommunications devices that do not use the TCP/IP stack. 3GPP includes CMIP in its solution sets for fault management.

Looking at the wider area of network management there are other methods than CORBA and SNMP available from different organisations and vendors. Large device manufacturers such as Cisco and Juniper have offered their own command line interface for managing their devices, but they do not scale well to managing a large group of devices. Large manufacturers have also developed their solutions to XML-based agents and utilised embedded web servers (EWS) on their devices to provide web based configuration management [Choi-et-al]. These vendor specific solutions do however mainly configuration management. IETF has started its own project, called Netconf, to standardise the vendor specific XML-based solutions [IETF-N]. It also though only concentrates on configuration management. The Internet Research Task Force (IRTF) – a sister group of the IETF – does research on future ways of fault management, and network management as a whole. At the moment it has improvement for SNMP, but no completely new management ways.

Configuration management has been seen as the weakness of SNMP, and therefore new

management solutions have concentrated on that. SNMP has been left on the side of these solutions to handle fault management. But nothing prevents from developing a complete XML-based management system as described in an article from Korean POSTECH [Choi-et-al]. Management information can be defined by XML Schema and transferred using HTTP over TCP. 3GPP has also done a feasibility study of XML-based telecommunications management and introduced a solution set to replace their CORBA solution set [3GPP-XML]. In this solution simple object access protocol (SOAP) is used to as XML messaging and invocation protocol. With the wide and ever widening usage of XML, XML/SOAP solutions can be seen as very viable candidates in the future of fault management.

3 Conversion between CORBA and SNMP Fault Management

The previous section described CORBA and SNMP – the two prominent methods for fault management. This section focuses on conversion between these two solutions and presents reasons why a service provider would want to conduct such a conversion.

3.1 Incentives for a CORBA-SNMP Converter

CORBA was taken to fault management use in late 1990s, when CMIP and SNMP were seen as hard to learn and implement [Deri&Ban]. Also the advent of Java gave wave to CORBA as it provided language mapping to Java [Henning]. This and other language mappings promised developers a tool that would allow them to build heterogeneous distributed applications and their management interfaces with relative ease. Particularly GSM and UMTS NEs were provided to service providers with almost solely CORBA management. However, SNMP remained in use and its usability and different implementations evolved over the years.

The biggest impact of CORBA as a management protocol was in the telecommunications sector, but after its initial success years, the concept of an all-IP world started gaining ground also in telecommunications. In this environment SNMP, as part of the Internet Protocol Suite, became once again the most popular fault management mode because of its merits, such as ease of implementation and great interoperability [Yoon-et-al]. CORBA on the other hand was being seen as complex, inconsistent and downright arcane [Henning]. The bursting of the Internet bubble in 2002 did also not help CORBA as its development wound down, because several vendors and software companies refocused their efforts. Also according to Henning, the standardization process of OMG hindered development of CORBA so that it is presently almost static in the management field.

While SNMP is not without its drawbacks, it is still the strongest choice for monitoring

NEs and for fault management on moderate size networks. Having been designed to be simple it has limitations on scalability, though this affects configuration management more than fault management. Because most network devices used worldwide are equipped with an SNMP agent, the best choice for a service provider wanting to manage all its NEs with a single protocol is SNMP. For an SP operating, both CORBA using and SNMP using NEs, the solution is to convert the CORBA based fault management to using SNMP.

3.2 General Aspects of Protocol Conversion

There has been only limited amount of research carried out on protocol conversion concerning element management or network management. In the late 90s, the trend in research was converting existing SNMP managed devices to CORBA management [Mazumbar], [Aschemann], [Deri&Ban], and in the early 2000s studies on SNMP-XML conversion have been made [Yoon], [Klie], [Choi-et-al]. From these studies similar general aspects for management protocol conversion can be gathered.

In most cases illustrated in research articles, protocol conversion includes two main parts, a translator and a gateway [Mazumbar], [Aschemann], [Yoon]. Usually different protocols have different presentations of information models and to convert this information a translation algorithm is needed. The translator's algorithm translates data type definitions; object references; variable, attribute, and other names between the protocols involved in the conversion. The loss of information is possible in these translations and should be taken into consideration when designing the algorithm. Some loss of less important information may though be justified for the functionality reasons. In some cases the original protocol has information and functions not needed by the desired new protocol, so translation of these functions is not always mandatory. The translator may have to convert one type, definition or name of one protocol to many on the other protocol or vice versa.

The gateway is needed to convert and map operations, methods and messages from one protocol to the other. This gateway conversion may also be similar to the conversion in

the translator, as one message in the other protocol may require two or more messages in the other. The gateway may also need to generate messages and monitor entities on its each side.

Some metrics are needed for the conversion evaluation. The conversion should be able to convert all desired functions, not add significant load compared to the original protocol, and not be noticeable to the user – the user will not notice the underlying protocol.

3.3 Comparison of Fault Management with CORBA and SNMP

As Section 2 illustrated, fault management can be done by using both CORBA and SNMP, but they have differences in how they execute fault management. There are differences in protocol stacks, data types, MO presentations, and messages.

The protocol stacks of CORBA and SNMP are presented side by side in Figure 11. Both SNMP and CORBA protocols – comprising of IIOP, GIOP and a stub or a skeleton – preside on the application layer. SNMP and CORBA use different protocols on transport layer; CORBA utilises the more reliable TCP while SNMP uses lighter and speedier UDP.

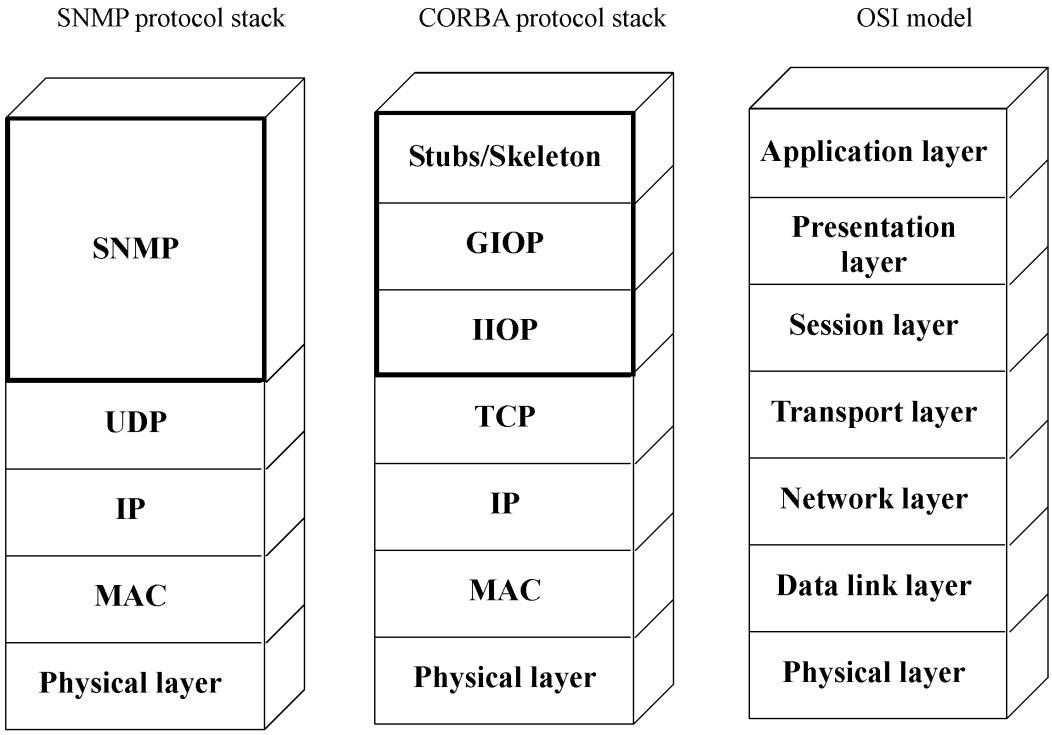

Figure 11: Protocol stacks of SNMP and CORBA

The messages of SNMP and CORBA used in fault management also differ. In CORBA all is done basically with two GIOP message types: request and reply. A request can also be cancelled and both message types can be fragmented. These messages are still versatile as they can convey different CORBA operations. The operations the EMS can perform on the NE via these messages and the CORBA services that provide these operations are the real essence of CORBA fault management. CORBA communications generally are also more complex than SNMP and have to cope with many aspects such as encoding of operation calls, or different parameter semantics. Contrary to CORBA, in SNMP the essence of fault management are the messages the EMS and the NE send to each other. SNMP uses eight different message types for fault management as described in Paragraph 2.4.1.3., and communication with these messages is defined very precisely. In addition to some general information, each SNMP message contains a list of name-value pairs, a so-called variable binding list. Especially the trap and notification messages are important as they alert the manager entity of alarms and other

events in the NE. The CORBA equivalents to SNMP traps are alarm and event operations. Since an SNMP message PDU can specify more than one variable that may span multiple tables and rows, each SNMP PDU can equal more than one CORBA operation [Mazumdar]. In describing data SNMP uses ASN.1 types in the PDUs while CORBA uses IDL types.

The MO presentation is quite similar, but there are differences in the details such as SNMP information modules being SMI documents while those of CORBA are IDL modules. As explained in Paragraphs 2.4.1.1 and 2.4.1.2 managed objects in SNMP are organised in a treelike hierarchy defined by unique object identifiers (illustrated in Figure 8). An object ID is made up of a series of integers based on the nodes in the tree, separated by dots. Each managed object has a numerical OID and an associated textual name; the OID textual name pairs pertaining to a specific network element are grouped in a MIB. In 3GPP CORBA fault management, managed objects are also organised in treelike hierarchy called name space [3GPP-1068]. Whereas SNMP MOs are similar to the pairs that IP addresses and URLs make, managed objects of CORBA are more like objects in object oriented programming languages. A distinguished name (DN) is used to uniquely identify a CORBA MO within a name space. The distinguished name is constructed from a series of name components referred to as relative distinguished names (RDN) such as the MO's type name and identity number; the full distinguished name contains the path from the MO to the global root MO similarly as for example file names in Unix file system. ITU-T Recommendation X.500 defines the concepts of DN and RDN in detail. The CORBA MOs have also attributes that reference what is their parent MO and what context does the MO belong.

Table 4 summarises the main differences in fault management of CORBA and SNMP.

Table 4: Differences of CORBA and SNMP Fault Management

	CORBA	**SNMP**	**Main difference**
Transport protocol	TCP	UDP	Retransmissions in TCP
Messages	2 formal types, many different operations, name-value pairs	8 formal types, OID-value pairs	Message structure: CORBA has fixed fields versus SNMP having free variable bindings
MO presentation	IDL modules, objects with attributes	SMI modules, OID-textual name pairs	Attributes within MO versus attributes as branches of MO
Data types	IDL types	ASN.1 types	
Manager functions	Several different operations for modifying and accessing MO information	Set-operation for variable-binding manipulation, get-operation for viewing variable-bindings	CORBA-operations are a lot like programming language functions, SNMP-operations simple and rigid
Security	Security Service (CORBASEC)	USM and VACM	

4 Converter

As described in Chapter 1 the objective of the thesis is to study how a network operator – having network elements using 3GPP's implementation of CORBA fault management (FM), while most other of the operator's NEs as well as the operator's EMS are using SNMP for fault management – can unify the fault management of its network and thus convert the CORBA based NEs into understanding SNMP FM. In order to do this a CORBA-SNMP converter is needed and designed to carry out the protocol conversion from one to another. The converter can be added into the NEs or between the EMS and the NE.

4.1 Structure of a CORBA-SNMP Converter

The structural design of the converter can be divided into five sections representing the different tasks of the CORBA SNMP conversion. These sections – protocol mapping, message mapping, managed object mapping, security mapping, receiving and forwarding messages – are combined to form the whole converter.

4.1.1 Protocol Mapping

To convey messages between the EMS and NE, the converter has to take care of the different protocol stacks of SNMP and CORBA. The stacks differ on top of the IP layer and operate on different styles of connections that have to be taken into consideration. Stallings [Stallings-1] describes a concept of an SNMP proxy agent, where an SNMP agent acts on behalf of one or more devices that do not implement SNMP. The CORBA-SNMP converter works similarly and Figure 12 indicates the protocol architecture. The converter must connect to the EMS according to the SNMP protocol stack and at the same time connect to the NE according to the CORBA protocol stack. Once both connections from the converter are set up, the connection between the EMS and the NE is complete.

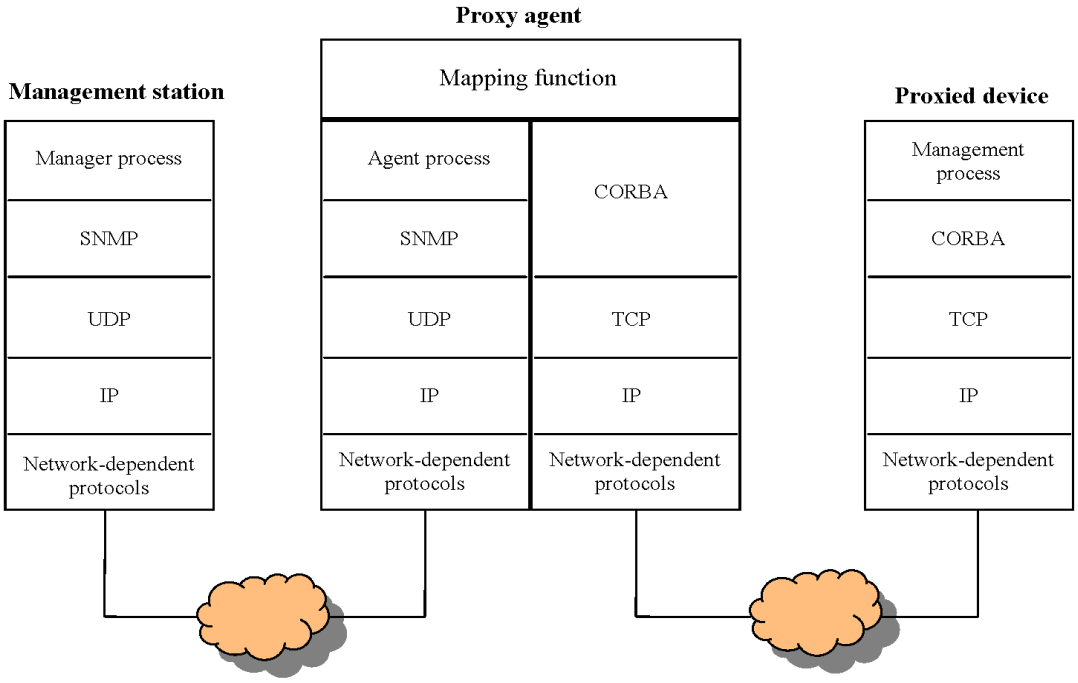

Figure 12: Proxy configuration of the converter [Stallings-1]

4.1.2 Message Mapping

In both CORBA and SNMP, the essential part of fault management is the NE detecting faults and other important events and reporting them to the EMS with messages. The other essential part is the EMS setting filters on the NE – also with messages – which determine what notifications it wants to receive. In essence, a CORBA-SNMP converter is a gateway that translates these FM messages that EMSs and NEs send each other. To the EMS the converter makes the CORBA managing interface of the NE look like a SNMP agent entity; at the same time the CORBA fault management remains unchanged for the NE.

As described of the CORBA based fault management in Paragraph 2.3.7, the faults occurring in CORBA based fault management systems are detected by a fault detector and reported to a fault notifier. The fault notifier filters these reports and propagates the filtered reports to consumers that have subscribed to them. The reporting is done by sending an alarm message or an event message to the EMS. The task of the CORBA-

SNMP converter is to translate these alarm or event messages into SNMP trap messages. The NE maintains a subscription list that also describes, according to filters applied to some of the subscriptions, which alarms and events are sent to which managing systems. The EMS can manage the subscription list with SNMP set messages. The converter needs to map the SNMP set and get messages to CORBA set requests, as well as the NE responses from CORBA to SNMP. Figure 13 illustrates the tasks and components of the converter.

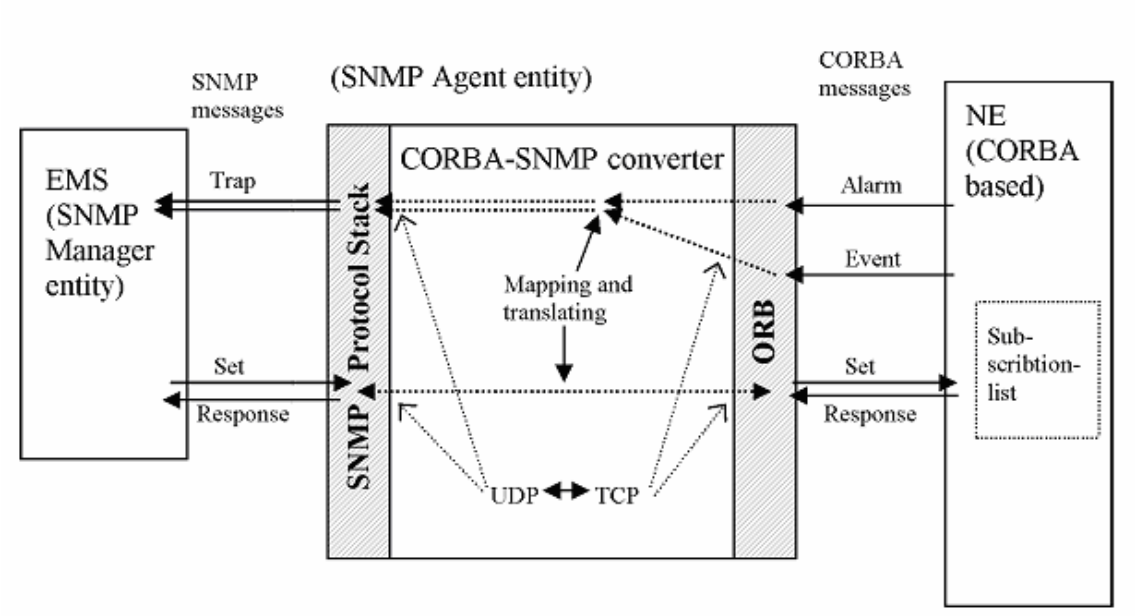

Figure 13: The interaction between SNMP manager and CORBA based NE using the CORBA-SNMP converter

SNMP Trap- and Set-PDUs contain only few mandatory fields or mandatory variable-bindings. Most information is in variable-bindings that are defined by specific MIB files. In order for the manager entity to understand information coming from the NE trough the converter, a MIB file corresponding to the CORBA based NEs must be defined and loaded to the converter and the EMS. The MIB file defines object types that correspond to the information fields of CORBA alarm and event notifications. It also defines the OIDs of each of these object types.

In an SNMP set message every variable-binding field must be valid in order for the

intended changes to be made [Stallings-1]. In addition to describing the CORBA based NE, a MIB file must also define what CORBA operations a SNMP set message can invoke on the NE. The converter maps these allowed variable-bindings to CORBA operations and operation parameters, and invokes the operation on the NE's ORB through its own ORB. The MIB file must similarly define what information an SNMP get messages can obtain from the NE.

4.1.3 Managed Object Mapping

The converter needs a MIB file that defines the NE information and characteristics. The MIB file needs to include – in addition to the traditional SNMP OID-textual name pairs – additional mapping of SNMP OID and CORBA distinguished name pairings. When the converter receives a notification from the NE it needs to map the DN of the MO causing the notification into a corresponding OID. The DN is a line containing every MO RDN in a direct succession between the root MO and the leaf MO causing the notification (for example a DN can look like "`DC=CompanyXYZ,Net=DS3BackBone,Station=TMR,Node=1,Port=3`"), and the mapping function needs to map each part separated by a comma to a part of an OID and append them to a full OID of the MO.

4.1.4 Security Mapping

In addition to protocol stacks, the mapping function has to convert the security functions of CORBA and SNMPv3. Since the security functions of SNMP and CORBA are mostly not similar, the converter cannot map security with exactly same parameters between the EMS and the NE. The most important part of the security mapping is trying to maintain the same level of security from the EMS through the converter to the NE and vice versa. The maker of the converter or the operator using the converter needs to decide what CORBASEC level and features correspond to the desired SNMP USM and VACM features.

4.1.5 Receiving and Forwarding Messages

In SNMPv3 the manager and agent tasks are represented by applications. A normal

SNMP (agent) entity utilises a proxy forwarder application to forward messages from other SNMP entities. As the converter also has to map and translate messages from the NE to the EMS and vice versa, messages cannot simply be forwarded by proxy. Each message needs to be read and a new message needs to be formed according to it. As a basic initialization, the converter should be configured to listen to the address and port of the EMS and replace the information of the EMS with its own in the subscription list of the NE.

One important aspect of message transfer that needs to be considered in the converter's design is message blocking. SNMP and CORBA messages can be sent synchronously (blocking) and asynchronously (non-blocking). If entities are in different threads of control, they are asynchronous and sending a request in which the sender expects a response does not halt the operations of the sender while waiting for the response. With the addition of a converter between the EMS and NE, synchronous message exchange may block the operations of the managing system for much longer than an asynchronous straight SNMP exchange. If the converter blocks all other messages while waiting for a response to its first received request, it can become a bottleneck and impede performance of the network element. Message receiving and handling should therefore be threaded by the converter. Figure 14 and Figure 15 illustrate the difference between blocking and non-blocking message exchange with simplified examples. Blocked requests from the EMS caused by blocking or other causes are not transmitted automatically unless the EMS application itself takes care of this. If the EMS is polling the NE periodically, the loss of some of the management data is acceptable. SNMP over TCP would help with some loss of management data, but would also increase network congestion with its retransmissions in situations for which SNMP is designed for, when networks are in trouble and something has gone wrong in network elements [Mauro].

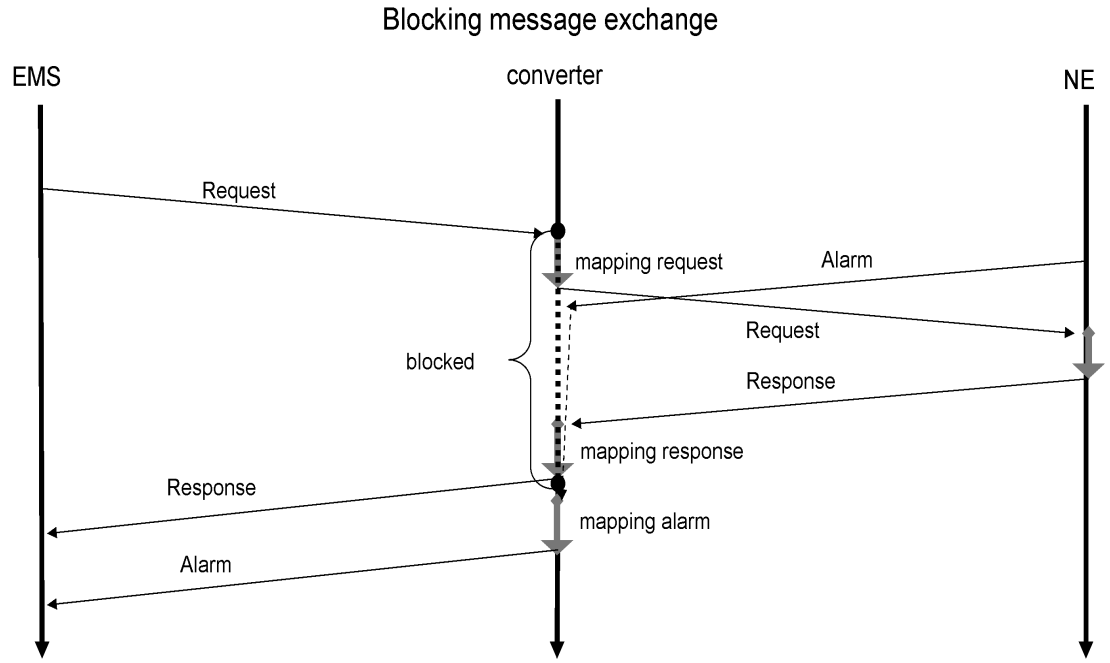

Figure 14: Synchronous message exchange through the converter

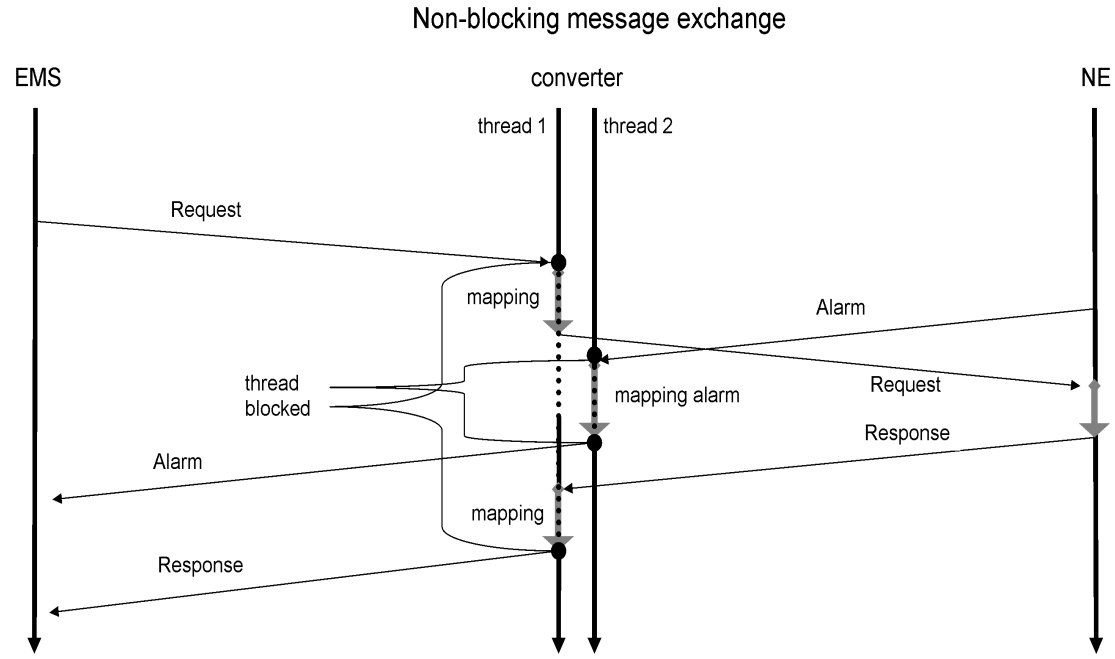

Figure 15: Asynchronous message exchange through the converter

In its most robust configuration, the converter handles each task in a different thread of control. Receiving messages from the NE, sending messages to the NE, receiving

messages from the EMS, and sending messages to the EMS are all handled with a separate thread. As the communication between the converter and the EMS and between the converter and the NE is done with different transport protocols, using different threads of control the converter will not block other functions, if a message times out.

4.2 Design of a CORBA-SNMP Converter

The intent of the converter implementation is to prove – as described in the previous sections – that the concept of converting a network element's CORBA fault management to use SNMP via a converter is functional. To make this proof of concept, the following kind of CORBA to SNMP converter was designed and implemented.

The converter implementation is written in Java programming language; the fault management of an NE in this thesis utilises Ericsson's Java implementation of IDL specifications for Alarm and Notification services defined by 3GPP's CORBA Solution Sets [3GPP-1063], [3GPP-1113]. For implementing SNMP managing system functionality and the SNMP functionality of the converter, an open source SNMP application programming interface (API) for Java is used [SNMP4J]. Although the implementation utilises an Ericsson NE, the converter was designed for interoperability with network elements and managing systems of other manufacturers; the design follows both CORBA and SNMP standards.

The operating principle of this example implementation is to manage CORBA fault management operations with SNMP messages and the following restrictions apply. The implementation only covers fault management conversion; as such no changes to the MOs or MO models can be made or reconfigurations performed. Also the operating principle could have been to operate as much as possible like a normal SNMP agent entity, but as the difference between these operating principles is not of great relevance to the presented conversion concept, this principle was chosen for testing purposes.

The implementation comprises of five Java classes: a class for handling the SNMP

connection and functions (`ConverterAgent`), a class to function as the converter's ORB and for handling the CORBA connection and operations (`ConverterORB`), a class that handles the mapping and translation functions (`MessageMapping`), a class that represents the converter's MIB file containing OID textual name DN pairings of every MO of the network element (`SnmpCorbaConstants`), and finally the class that combines the other classes and utilises them to make the converter functional (`Converter`). These classes also utilise the previously mentioned SNMP and CORBA Java libraries. The MIB also contains the CORBA operation parameter OID pairs needed for the mapping functions. To avoid message blocking the converter object runs in its own thread as well as running both its agent and ORB objects in their own separate threads.

To start up the implementation, several tasks are performed. First the converter implementation sets up both the CORBA connection toward the NE and the SNMP connection toward the EMS. The converter initiates its ORB to start up the CORBA runtime system; the converter needs to use CORBA naming service for resolving the needed CORBA objects from the NE. But before the converter can use the naming service, it must get a reference to the naming service itself. This is done by fetching an IOR file that has a stored stringified object reference to the root naming context. Next, with the resolved naming service, the converter imports references to AlarmIRP and NotificationIRP CORBA objects; these handle the fault management operations on the NE. Once the converter has a reference to a CORBA object in the NE, the converter can invoke operations upon it. For receiving and properly handling notifications from the ORB of the NE, a sequence push consumer must be connected to the converter's ORB. The converter's ORB can be put to listen for notifications in the start up, or later with a specific function (`attach_push`).

To set up the SNMP connection, the converter starts SNMP agent entity processes: multithreaded message dispatcher, UDP transport mapping with IPv4, message processing model MPv3, command responder, and security models. The address and port number of the EMS for sending, and the address and port number of the converter

agent for listening are also configured – this information also needs be configured in the managing system, for communicating with the NE trough the converter. After finishing its installation the agent starts listening for messages. Figure 16 illustrates the components and start up processes of the converter implementation.

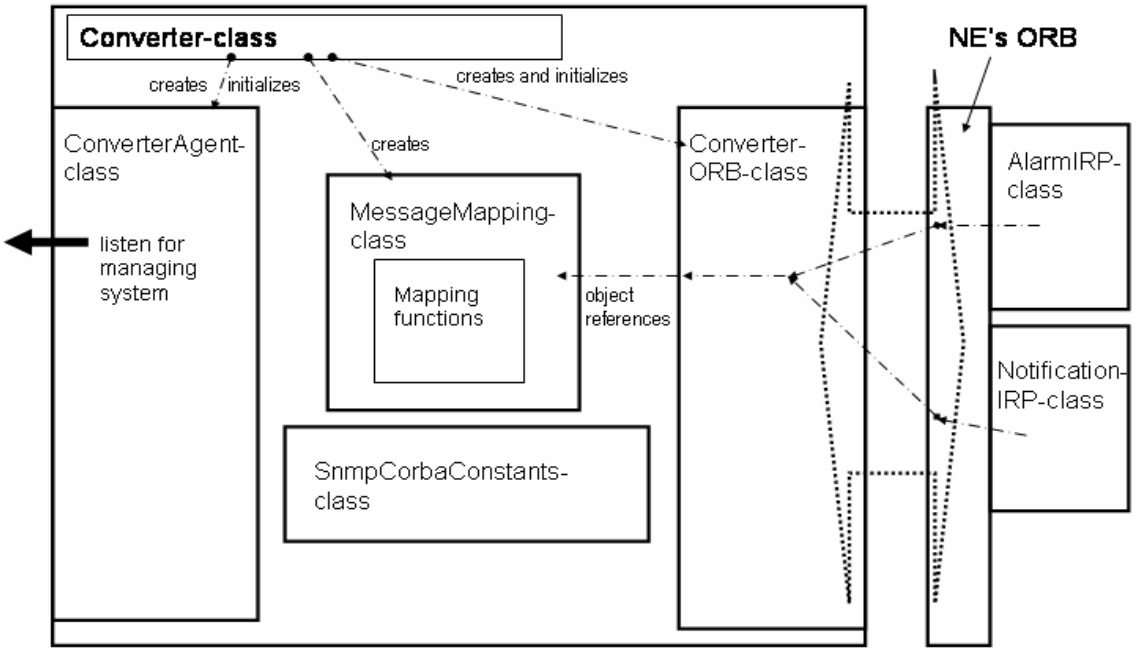

Figure 16: Converter implementation at start up

Also at the start up of the converter, both the converter agent and the converter ORB set up security features. Security mapping, or more accurately applying the same level of security between the EMS and NE, is set up. The agent is set up with USM security model and VACM access control and in the ORB CORBASEC is set up from an IOR file.

4.2.1 Protocol Mapping Implementation

A UDP connection between the EMS and the converter is established with the setting up of the converter's SNMP agent entity. In setting up the converter's CORBA ORB, a TCP connection with IPv4 between the converter and the NE is established. As the messages that the converter needs to convey between the EMS and the NE require more than just encapsulating the message payload with new transport protocol headers, no

direct protocol mapping between the transportation protocols is done. The converter receives an SNMP message, reads its contents and based on the information, uses mapping functions to invoke a new CORBA operation through the ORB. When the ORB of the converter gets a notification form the NE, a SNMP message is formed and mapped from the information of the notification. Because of this different philosophy of working in SNMP and CORBA, the converter maintains two separate transport connections, instead of directly mapping the transport protocols.

4.2.2 Message Mapping Implementation

In the 3GPP implementation of CORBA fault management alarm and event notifications are transported in a message called structured event [3GPP-1063]. A structured event is comprised of three struct constructs: header, filterable data, and the remainder of body. The header can be further decomposed into a fixed portion and a variable portion, and the fixed portion of the header divided into event type and event name. Still further in details the event type has two fields: domain name and type name. The domain name identifies the particular vertical industry domain in which the event type is defined (for example telecommunications, finance, or health care); the type name categorizes the type of the event uniquely within the domain. For both alarm and event notifications the header part of the structured event is similar, but the number of fields in filterable data differs to some extent. Filterable data comprises of name-value pairs called properties. Both alarm and event have fields for managed object class, managed object instance, event time, notification ID, specific event or problem, system distinguished name, and additional text and info. The alarm notification has also fields for probable cause, perceived severity, alarm ID and acknowledgement handling. The format of these name-value pairs is mostly string-string with a few string-integer and string-long integer pairs.

The task of the converter is to read the received structured event and construct an SNMP Trap-PDU. The converter fills the request-id field and variable-bindings of the PDU based on the information it reads from the structured event. Mandatory value-name pairs in the Trap-PDU are `sysUpTime.0` and `snmpTrapOID.0`. The field for

`sysUpTime` is filled with system uptime OID and the converter's own time counter value; the field for `snmpTrapOID` is filled with an OID indicating an enterprise SNMP trap and an OID corresponding to the type name field of the structured event. Other information fields of the structured event are mapped to an OID corresponding to the name of the field. The values are mapped as is, except for the managed object instance field containing the MO's DN, which will be mapped to the corresponding OID of the MO, as the following paragraph explains. The converter looks up the OIDs from its MIB file. After executing mapping functions, the converter sends the new constructed trap message to the EMS.

The 3GPP specification TS 32.106-8 gives an example of a managed object model and its corresponding relative distinguished names seen here in Figure 17 [3GPP-1068]. In this figure the bottom object of NS-B right branch has a DN of "`DC=se.ericsson.lmc,A=9,F=1,G=1,H=2`". The first part represents the DN prefix of the maker of the MO and will be mapped to the enterprise OID prefix of the maker (1.3.6.1.4.1.193 for Ericsson). The local DN "`A=9,F=1,G=1,H=2`" will be mapped as "9.1.1.2" with each RDN number part appended to the OID; so the full DN of the example MO will map to an OID of 1.3.6.1.4.1.193.9.1.1.2.

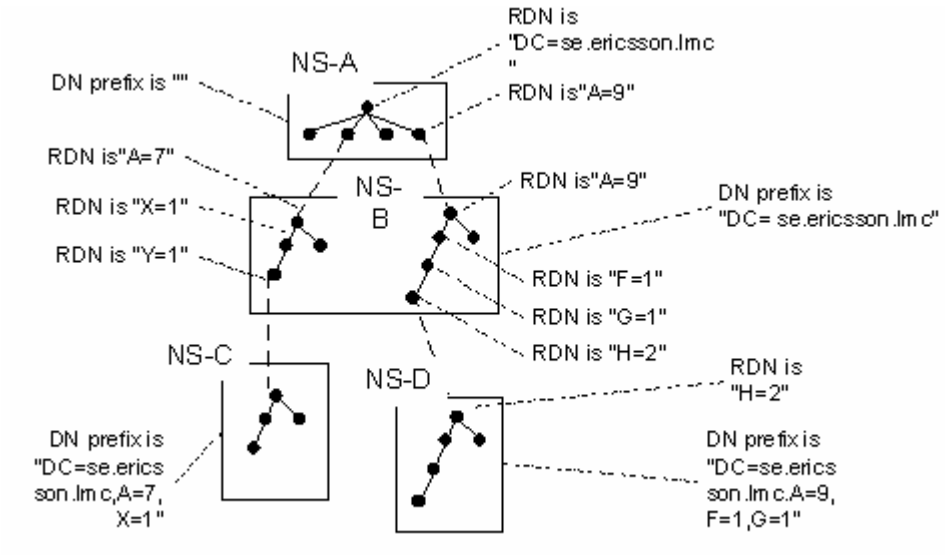

Figure 17: Managed Object Model and Name Space Partitions [3GPP-1068]

Upon receiving a get or set message from the EMS, the converter performs the mapping in the opposite direction. In the 3GPP CORBA FM implementation, the managing system can perform a set of different configuration operations on the NE. These include subscribing to notifications, removing a subscription, setting or changing a filter for subscribed notifications, and setting or removing configuration for the severity of a specific problem. The converter needs to map variable-bindings in an SNMP set message to these operations and respond with an error response, if mapping does not match correctly. The CORBA operations can have several parameters and the parameters can also be objects with their own parameters. A list of all these parameters and sub-parameters needs to be in a MIB file, and a corresponding OID and variable-binding assigned to each of them. In case the set operation is successful, the managing system gets a response that no errors occurred.

The managing system can retrieve information from the NE with several “get” CORBA operations. These include retrieving the subscription status (this also keeps the subscription active), the versions of the notification and alarm services, list of active alarms, number of active alarms, and a list of modified alarms. Similar to set messages, the converter needs to map variable-bindings in an SNMP get message to these operations and the parameters of these operations. The converter generates and maps a response message from the response information of the NE and sends it to the managing system.

4.3 Design Evaluation

To test the functionality of the converter implementation, the designed Java classes were compiled, and three different types of test case sets were run: cases involving SNMP set messages; cases involving SNMP get messages; and cases involving SNMP trap messages. During the test environment set up, some modifications to the complete designed operating environment of the converter were made in order to cope with the complexity of the complete system. For simplicity, simulated Alarm and Notification services of the NE – run on the same computer as the converter – were used and the CORBA security functions turned off. Because of these limitations the tests were

focused on the inner functionality of the converter, starting at the point where the converter receives a generated test message and ending at where the converter would send the EMS a response or trap message. With this test set up extensive functionality testing was performed, covering the whole CORBA functionality.

4.3.1 Test Scenarios

As Figure 16 illustrated, the converter creates `MessageMapping` object and creates and initializes threads for `ConverterAgent` and `ConverterORB` objects. `ConverterORB` conveys AlaramIRP and NotificationIRP object references to `MessageMapping`. As `SnmpCorbaConstans` is a static class with no functions, it does not need initialising.

The first test was to test the converter's subscribing to notifications. As Figure 18 illustrates a Set-PDU is formed in `ConverterAgent` containing OID of `attach_push` operation. Functions of the `MessageMapping` object map the variable-bindings of the PDU and invoke the `attach_push` operation on the NotificationIRP object reference. The operation returns the subscription ID of the new notification subscription and it is put into a Response-PDU. The Response-PDU is relayed to the `ConverterAgent` for sending to the managing system. Similar actions were performed also in all the other modifying operations tested by the *set* test case category: `detach`, `attach_push_ext`, `change_subscription_filter_ext`, and `set_severity` operations.

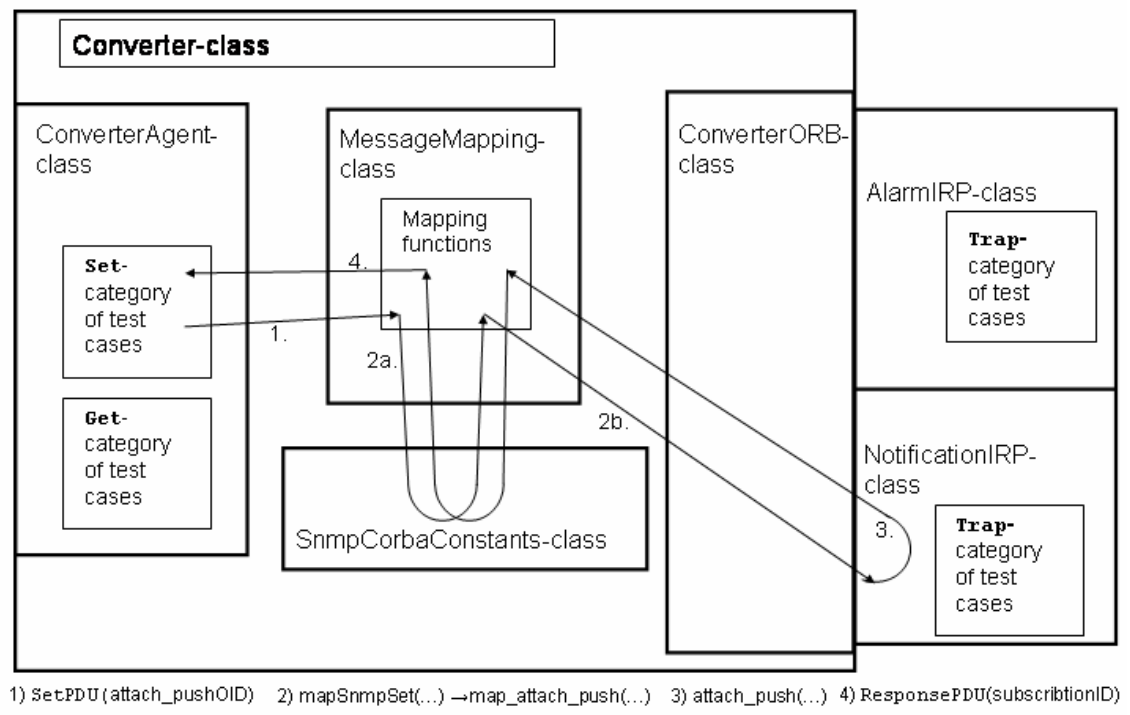

Figure 18: Using CORBA `attach_push` with an SNMP Set-PDU

The *get* category of test cases tested all CORBA operations for information fetching; they were invoked with Get-PDUs or in case of fetching an alarm list with GetBulk-PDUs. The CORBA operations controlled with Get-PDUs are `get_subscription_status`, `get_subscription_status_ext`, `get_notification_IRP_version`, `get_alarm_count`, `get_alarm_count_ext`, `get_alarm_IRP_version`, and `get_all_modified_severity`; the Get-PDU contains the operation's OID, which invokes the specified operation, and the fetched information is packed into a Response-PDU. As Figure 19 illustrates, although `get_alarm_list` and `get_alarm_list_ext` operations are invoked with GetBulk-PDUs, they function similarly as test cases invoked with Get-PDU. The difference is that the maximum size of the response (maximum number of variable-bindings) is determined in the GetBulk-PDU.

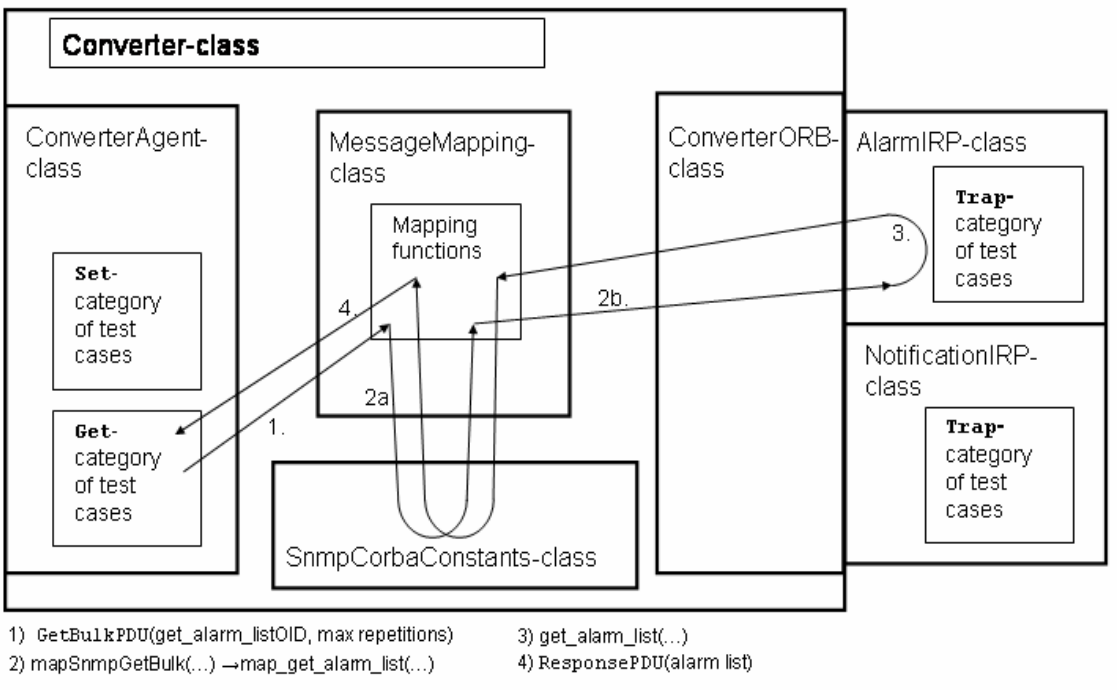

Figure 19: Fetching an alarm list with an SNMP GetBulk-PDU

The third and final test case category was cases involving CORBA alarms and events. Alarms were generated by the `AlarmIRP`-class instance and events were generated by the `NotificationIRP`-class instance, all were passed as structured event to the `ConverterORB` object. Figure 20 illustrates how a structured event is mapped to a Trap-PDU by the `MessageMapping` object and passed to the `ConverterAgent` for sending.

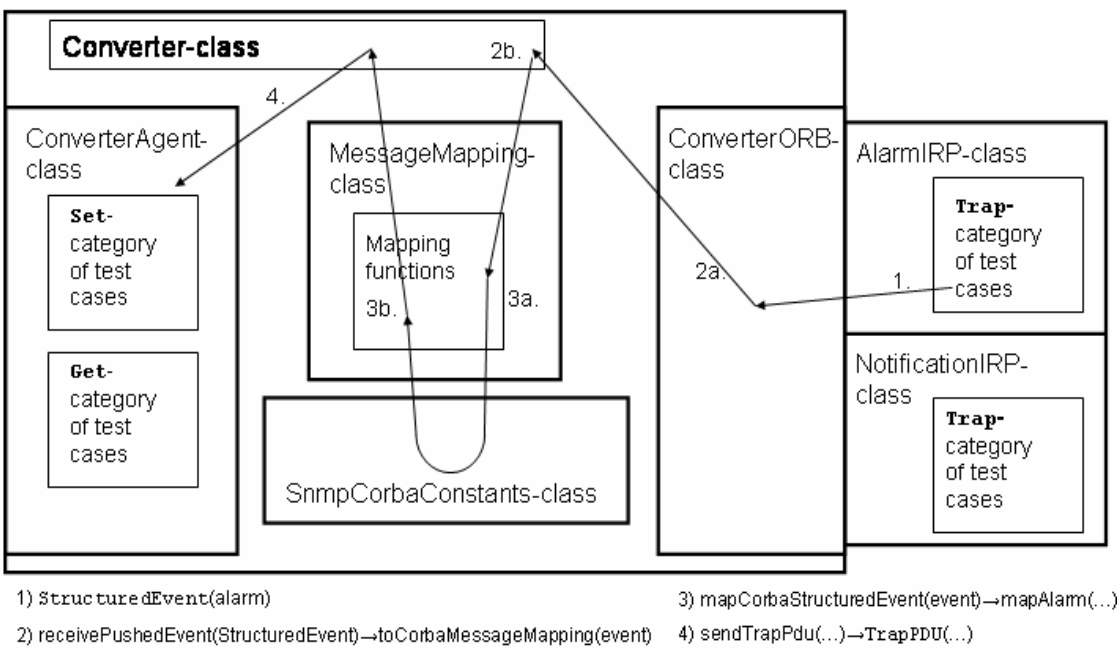

Figure 20: A structured event containing an alarm mapped to an SNMP Trap-PDU

4.3.2 Results and Analysis

The test cases showed that the converter implementation enabled invoking all the CORBA FM operations with SNMP Set- and Get-PDUs as well as monitoring CORBA notifications as SNMP Trap-PDUs. The implementation was able to map information correctly in every test case and all tested CORBA operations functioned correctly. As the different component objects were run in separate threads, the converter implementation was able to perform different kinds of test cases simultaneously; notifications were mapped and sent at the same time as invoking a get operation. If a large number of alarms was generated, they were all mapped and conveyed consecutively and instantly by the converter and none were discarded. The relevant part of fault management is to inform managing systems with correct fault information immediately, with the focus on information details and correctness; these parts the converter handles well.

As the converter is implemented between the EMS and the NE, it adds processing on

the network management. Despite of the limitations set by the test environment, some performance measurements were made. Fetching an alarm list is by far the most complex operation for the mapping functions of the converter, so it describes best how much at most the converter adds to the message delivery time.

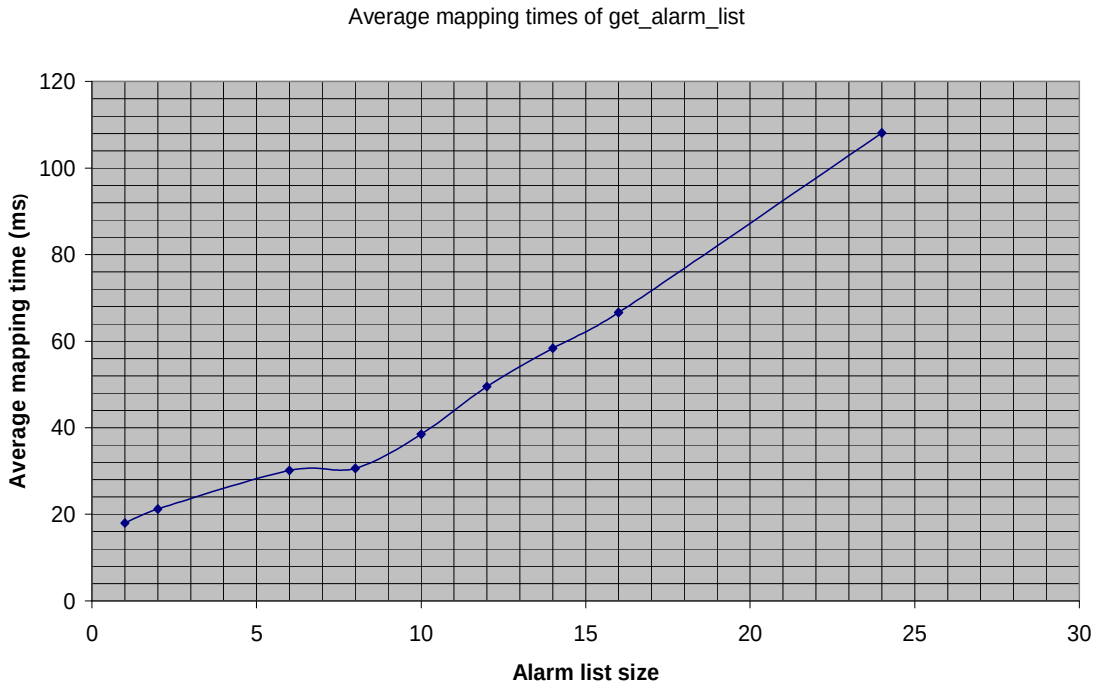

Figure 21: The average mapping times of different sized alarm lists

The time increase caused by the mapping function was tested with fetching different sized alarm lists. Because the CORBA operation returns a list of structured events, every alarm added to the Trap-PDU must be mapped separately. Figure 21 shows that increasing the size of the alarm list increases the mapping time approximately linearly. The measurements were made by adding time stamps to the start and the end of the mapping function. Measurements on each list size were run ten times. The Java implementation was run with Eclipse SDK 3.2.1 using Java 1.4.2. JRE, and the test environment was running on a laptop computer with SUSE Linux Enterprise Desktop 10 operating system, Intel Core2 T7200 2 GHz processor and 4 GB of memory. During the measurements some other processes were also running on the computer; as the scale of the measurements was in milliseconds, the resource consumption added some

fluctuation on the measurements. Still these measurements can be considered giving indication of the real mapping performance. Approximating from the measurements and the graph in Figure 21, even with an alarm list consisting of a hundred active alarms it would still take only approximately 500 ms to map the alarm list operation; this time addition in the context of transporting management data can be considered quite harmless as the original CORBA operations take much more time. To the person running the managing system, the added converter would not create any practical slowing down compared to the previous, in the common use of fault management. Measurements of the impact that the converter has on the whole round trip time should also be measured, but they are left to be made on future implementations.

Measurements were also made on all the other mapped operations: the other get-operations were mapped in approximately 1 ms, set-operations were mapped in approximately 15 ms, and alarms and events were mapped in between 7 ms and 30 ms. It can be concluded that the added delay from these mappings is of no real concern to the managing system.

There were no clear ways to test how much the converter implementation consumes memory resources, but some hints could be obtained from how it functions. The implementation runs three simultaneous threads, but most of the time they just wait for incoming messages. The converter does not maintain any statistics or lists of variables, only a static MIB file and the request IDs (four bytes each) for waiting response messages. So it can be concluded that compared to the original CORBA fault management, the converter does not add considerable load to the resources of the managed system, even if the converter were to be added into the NE.

The SNMP management messages handle all managing operations with single request-response message pair or with a single trap message. As such the converter does not greatly burden the bandwidth requirements between the EMS and the NE. The lighter UDP connection between the EMS and the converter will not increase the bandwidth usage of the network as much as adding another TCP connection would. The original

bandwidth usage between the EMS and the NE is now used between the converter and the NE, and a lighter moderately used UDP connection between the EMS and the converter is added to the bandwidth usage.

The converter's handling of errors and incorrect management commands follows SNMP practises. Unless the information in the request message is completely correct, an error response is returned, instead of the requested information or command. Error messages from the NE will be forwarded to the EMS using SNMP error responses. SNMP error code and error index message fields provide the converter a variety of possible error responses, although some CORBA error information can be forfeit in the conversion.

Even though the converter implementation was simplified, it gave indication that a converter would be useful when a CORBA-SNMP conversion is beneficial. The implementation enabled the testing of the conversion of existing CORBA functionalities and acts as a good starting point for further examinations.

5 Conclusions

At the start of this thesis none or very few papers could be found on the specific details of protocol conversion from CORBA to SNMP. Based on this, the objective was set to study the conversion of fault management between CORBA and SNMP with focus on the conversion from CORBA based FM to SNMP. This thesis described the background and details of the conversion and proved the concepts functional. With a simplified implementation, a proof of concept for conversion of a CORBA based NE to using SNMP without changing its inner fault management mechanisms was reached. At the end it was observed that the added CORBA-SNMP converter was quite simple to construct, although only the fault aspect of the whole FCAPS-model was considered in the implementation.

The converter implementation in this thesis was designed with the emphasis on the working philosophy of CORBA fault management. The managing system utilised the CORBA operations of the NE by SNMP PDUs, and the notifications from the NE were in essence in the same CORBA form, only delivered in an SNMP PDU. This is however not mandatory; the converter implementation can be designed to work in a more SNMP oriented way and use more elaborate mapping functions. This way some of the CORBA fault management functionality would have to be forfeit, but the NE could be managed more in the way of a native SNMP using NE.

It can be concluded that the benefits of the CORBA-SNMP conversion over its drawbacks are much greater to the operator that is considering the conversion. The converter does not add considerable delay, strain on bandwidth usage, or consume memory resources and the benefits of unifying the managing system of all the network elements of the operator will likely provide performance and cost savings.

5.1 Further Study

This thesis only covered the conversion of network element's CORBA based fault management, so the next logical step would be to study the conversion in further detail

and without test environment limitations. The adding of configuration management would also add to the fault management conversion, as the presently absent conversion of managed object models would be covered. Configuration and performance management would also cover some of the actions needed to be performed as a consequence of fault management actions. As this study made only a shallow proof of concept implementation and testing on the CORBA-SNMP converter, a real prototype of the converter would be needed for more detailed testing. The prototype could be made with only fault management conversion, but would be perhaps more beneficial with configuration and performance management conversion added in some form

Adding configuration and performance management conversion to the converter requires also further consideration on the role of SNMP. SNMP is at its best in monitoring network elements and as such a natural choice for fault management. But for configuration and performance management more useful and preferred protocols are in use. Protocols such as Netconf and other XML based solutions are replacing or have already replaced SNMP in configuration management [Choi-et-al] [Yoon], and adding CORBA-Netconf mapping to the converter could be a wiser choice for converting configuration management of a CORBA based NE. Considering the design of the converter implementation, adding some other type of protocol conversion to the converter would seem possible, but its impacts would have to be studied and tested. The converter could also be expanded to work in a reverse set up, between CORBA based managing system and SNMP based NE. Studies on this sort of conversion have been written and mapping functions made as presented in Section 3.2.

References

[3GPP-XML] 3GPP TR 32.809: Telecommunication management; Feasibility study of XML-based (SOAP/HTTP) IRP solution sets

[3GPP-1063] 3GPP TS 32.106-3: Telecommunication management; Configuration Management; Part 3: Notification Integration Reference Point: CORBA Solution Set Version 1:1

[3GPP-1068] 3GPP TS 32.106-8: Telecommunication management; Configuration Management; Part 8: Name convention for Managed Objects

[3GPP-1113] 3GPP TS 32.111-3: Telecommunication management; Configuration Management; Part 3: Alarm Integration Reference Point: CORBA Solution Set

[Agrawal] Agrawal, N.: On the design of Element Management System for Node Bs in a 3G Wireless Network ; Personal Wireless Communications, 2002 IEEE International Conference on, Publication Date: 15-17 Dec. 2002

[Aschemann] Aschemann G., Mohr T., Ruppert M.: Integration of SNMP into a CORBA- and Web-based Management Environment, Department of Computer Science, Darmstadt University of Technology, Germany 1999

[Choi-et-al] Choi M-J., Oh J-M., Hong J.W.: Design and Implementation of an XML-Based Management Agent, Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP, Publication Date: 23-23 April 2004

[Deri&Ban] Deri L., Ban B.:Static vs Dynamic CMIP/SNMP Network Management Using CORBA, IBM Zurich Research Laboratory, Lecture Notes in Computer Science, Intelligence in Services and Networks: Technology for Cooperative Competition Springer Berlin / Heidelberg 1997

[Hanemann] Hanemann A., Sailer M., Schmitz D.: Assured Service Quality by Improved Fault Management, Proceedings of the 2nd international conference on Service oriented computing 2004, New York, NY, USA, ACM 2004

[Henning] Henning M., The Rice and Fall of CORBA, Q focus: component technologies, York, NY, USA, ACM 2006

[IEC] IEC: Tutorial: Element Management Systems (EMSs), http://www.iec.org/online/tutorials/ems/ 2007 [visited 28.4.2009]

[IETF-N] Network Configuration (Active WG): Netconf Status Pages, http://tools.ietf.org/wg/netconf/ [visited 28.4.2009]

[ITU3000] ITU-T: Recommendation M.3000

[ITU3010] ITU-T: Recommendation M.3010

[ITU3200] ITU-T: Recommendation M.3200

[ITU3400] ITU-T: Recommendation M.3400

[Klerer] Klerer S. M.: The OSI Management Architecture: an Overview, IEEE Network March 1988

[Klie] Klie T., Strauß F.: Integrating SNMP Agents with XML-Based

Management Systems, IEEE Communications Magazine, July 2004

[Mauro] Mauro D., Schmidt K.: Essential SNMP, 2nd Edition, O'Reilly 2005

[Mazumdar] Mazumdar S.: Inter-Domain Management between CORBA and SNMP : WEB-based Management - CORBA/SNMP Gateway Approach, Bell Laboratories 1996

[Misra] Misra K.: OSS for Telecom Networks: An Introduction to Network Management, Springer-Verlag London Limited 2004

[McHale] McHale C.: CORBA Explained Simply, www.CiaranMcHale.com/download/ 2007 [visited 28.4.2009]

[OMG3.0.3] OMG: Common Object Request Broker Architecture (CORBA) Specification, Version 3.0.3, 2004

[OMG3.1] OMG: Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, 2008

[Pope] Pope A.: The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture, Addison-Wesley 1997

[Raman-1] Raman L.: Fundamentals of telecommunications network management, IEEE Press 1998

[Raman-2] Raman L.: OSI Systems and Network Management, IEEE Communications Magazine March 1998

[SNMP4J] SNMP4J - The Object Oriented SNMP API for Java Managers and Agents, Apache 2.0 license, http://www.snmp4j.org/ [visited 28.4.2009]

[Stallings-1] Stallings W.: SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Third Edition, Addison-Wesley 1999

[Stallings-2] Stallings W.: SNMPv3: A Security Enhancement for SNMP, IEEE Communications Surveys, Fourth Quarter 1998 Vol. 1 No. 1

[Yoon] Yoon J-H., Ju H-T., Hong J.W.: Development of SNMP-XML translator and gateway for XML-based integrated network management, International Journal of Network Management Volume 13 Issue 4, John Wiley & Sons, Ltd. 2003