MANAGING NEXT GENERATION NETWORKS (NGNS) BASED ON THE SERVICE-ORIENTED ARCHITECTURE (SOA)

Design, Development and testing of a message-based Network Management platform for the integration of heterogeneous management systems

Konstantinos KOTSOPOULOS

BSc, MSc

Submitted for the degree

of Doctor of Philosophy

School of Engineering Design and Technology

University of Bradford

2010

ABSTRACT

Next Generation Networks (NGNs) aim to provide a unified network infrastructure to offer multimedia data and telecommunication services through IP convergence. NGNs utilize multiple broadband, QoS-enabled transport technologies, creating a converged packet-switched network infrastructure, where service-related functions are separated from the transport functions. This requires significant changes in the way how networks are managed to handle the complexity and heterogeneity of NGNs.

This thesis proposes a Service Oriented Architecture (SOA) based management framework that integrates heterogeneous management systems in a loose coupling manner. The key benefit of the proposed management architecture is the reduction of the complexity through service and data integration. A network management middleware layer that merges low level management functionality with higher level management operations to resolve the problem of heterogeneity was proposed.

A prototype was implemented using Web Services and a testbed was developed using trouble ticket systems as the management application to demonstrate the functionality of the proposed framework. Test results show the correcting functioning of the system. It also concludes that the proposed framework fulfils the principles behind the SOA philosophy.

Keywords: Next Generation Networks, Management Framework, Service Oriented Architecture, Middleware, Web Services.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance, help and encouragement of several individuals, who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study. First and foremost I would like to thank my supervisors, Professor Yim Fun Hu and Dr Pouwan Lei, for all their support and advice throughout this research.

I would also like to thank my colleagues and friends at the University of Bradford (both past and present) for making the group such a stimulating and enjoyable place in which to research.

I would like to especially thank my partner Ioanna, who has been a great source of motivation and inspiration and whose constant support and generous encouragement helped ensure the success of this thesis.

Finally, I would like to express my deep gratitude to my parents for instilling in me an interest in learning and an appetite for knowledge. Thanks also to my brother for his encouragement in completing this research.

DEDICATION

This thesis is dedicated to my parents, Stavro and Aleka, the two most special persons in my life. They, not only gave me life, but also fill it with all the love and affection one can wish for. Thank you.

Στους γονείς μου!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

3GPP	3rd Generation Partnership Project
ACSE	Association Control Service Element
API	Application Programming Interface
AR	Accuracy Rate
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
B2B	Business to Business
BLA	Business Level Agreements
BOA	Basic Object Adaptor
BPM	Business Process Model
BSS	Business Support Systems
C2B	Consumer to Business
CAMEL	Customized Applications for Mobile Network Enhanced Logic
CLT	Command Line Tool
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
CMS	Core Messaging Services
CN	Core Networks
CORBA	Common Object Request Broker Architecture
COM	Component Object Model
COTS	Commercial, off-the-shelf
CPU	Central Processing Unit
CRM	Relationship Management
CRUD	Create Read, Update, Delete
CSLN	Client Service Layer Network
DCOM	Distributed Component Object Model
DCN	Donor Conception Network
DII	Dynamic Invocation Interface
DOM	Document Object Model
DOT	Distributed Object Technology
DPE	Distributed Processing Environment
DR	Discard Rate

DSI	Dynamic Skeleton Interface
DTD	Document Type Definition
EGP	Exterior Gateway Protocol
EIS	Enterprise Information System-tier
EJB	Enterprise Java Beans
EM	Event Message
EML	Element Management Layer
ER	Error Rate
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standards Institute
eTOM	enhanced Telecom Operations Map
FCAPS	Fault, Configuration, Accounting, Performance and Security
FDU	Full-Duplex Utilization
GDMO	Guideline for Definition of Managed Objects
GNMA	Global Network Management Application
GNMS	Global Network Management System
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GTTS	Global Trouble Ticketing System
GUI	Graphical User Interface
HDU	Half-Duplex Utilization
HMI	Human-Machine Interaction
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ICT	Information and Communications Technology
IDE	Integrated Development Environment
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IMAP	Internet Message Access Protocol
IMS	IP Multimedia System
IN	Intelligent Networks
IP	Internet Protocol
IP-CAN	IP-Connectivity Access Networks

IPLR	Internet Protocol packet Loss Ratio
ISDN	Integrated Service Digital Network
ISO	International Standards Organization
ISV	Independent Software Vendor
IT	Information Technology
ITU-T	International Telecommunication Union – Telecommunication sector
J2EE	Java 2 Platform, Enterprise Edition
JAIN	Java APIs for Integrated Networks
JAXP	Java API for XML Processing
JDK	Java Development Kit
JMS	Java Messaging Service
JRMP	Java Remote Method Protocol
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JVM	Java Virtual Machine
LAN	Local Area Network
LLA	Logical Layer Architecture
LNFed	Layered Network Federation
LNMA	Local Network Management Application
LNMS	Local Network Management System
MAC	Medium Access Control
MAS	Management Association Services
MEP	Message Exchange Pattern
MIB	Management Information Base
MIDL	Microsoft Interface Definition Language
MIME	Multipurpose Internet Mail Extensions
MM	Management Message
MO	Managed Object
MOF	Model Object Format
MOM	Message Oriented Middleware
MOS	Management Operation Services
MPLS	Multiprotocol Label Switching
MS	Management Service
NE	Network Element

NEF	Network Element Function
NEL	Network Element Layer
NGN	Next Generation Network
NGOSS	New Generation Operations Systems and Software
NML	Network Management Layer
NMP	Network Management Platform
NMS	Network Management System
NOC	Network Operation Centre
NRA	Network Resource Architecture
OAM&P	Operations, Administration, Maintenance and Provisioning
ODP	Open Distributed Processing
OID	Object Identifier
OMA	Open Mobile Alliance
OMG	Object Management Group
ORB	Object Request Broker
ORPC	Object-oriented Remote Procedure Call
OS	Operating System
OSA	Open Services Architecture
OSI	Open System Interconnection
OSS	Operational Support Systems
PC	Personal Computer
PDU	Protocol Data Unit
POP	Post Office Protocol
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RPC	Remote Procedure Calls
REST	Representational State Transfer
RMI	Remote Method Invocation
ROI	Return of Investment
RO	Reference Point
ROSE	Remote Operations Service Elements
RSS	Really Simple Syndication
SCM	Service Control Manager
SDH	Synchronous Digital Hierarchy
SDP	Service Delivery Platform

SID	Shared Information Data/Model
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMI	Structure of Management Information
SML	Service Management Layer
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Service Oriented Architecture Protocol
SS7	Signaling System #7
SQL	Structured Query Language
TCon	Terminal Connection
TCP	Transmission Control Protocol
TDM	Time-Division Multiplexing
TINA	Telecommunication information Networking Architecture
TMF	Telemanagement Forum
TMN	Telecommunications Management Network
TR_IP	Total Receive Internet Protocol
TT_IP	Total Transmit Internet Protocol
TT_OK	Total Successful Transmit Packets
TTS	Trouble Ticketing System
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol
WAN	Wide Area Network
WBEM	Web Based Enterprise Management
WIMA	Web-based Integrated Network Management Architecture
WiMAX	Worldwide Interoperabilty for Microwave Access
WLAN	Wireless Local Area Network
WS	Web Service
WSDL	Web Service Definition Language
XML	eXtensible Markup Language
XSD	eXtensible Schema Definition

XSLT	eXtensible Stylesheet Language Transformation

Chapter 1 : INTRODUCTION

1.1 Statement of the problem

Corporations nowadays are increasingly dependent on computers and networking services to run their business. Keeping these network services operational is synonymous with keeping the business operational. Network management is the key to successful network operations and thus has a direct impact on the day-to-day business operations.

Traditionally, management systems were developed and customized ‘in-house’ by network and service providers. However, these attempts tended to result in very complex and high costs management infrastructures. In an effort to reduce cost and to manage the complexity, networks and service providers have adopted the strategy to purchase individual management systems (hardware and software) from different vendors. However, such individual systems incur significant difficulties in interoperability and functional reuse.

With the deregulation of the telecommunication industry, cooperation between service and network providers increases. Such cooperation is generally aimed at gaining access to global markets. The increased competition but also cooperation between network providers greatly affect the way networks and services are managed today and will be managed in the future. The need for greater interoperability across organizational boundaries, can also been seen as a consequence of the globalization of services, where global service delivery usually requires significant

management system interactions across different providers. Due to the increasing business-to-business computing integration on the Internet and the growth of global markets, the need for inter-communication cooperation is ever increasing.

Standardization has been a key influence for the design and integration of management systems. The architectural landscape of management systems includes the standardization effort of several telecommunication groups such as Telecommunication Network Management (TMN) [ITU-T] and the Internet Engineering Technology Taskforce [IETF].

However as telecommunication is increasingly more embedded in modern organizations, mainstream computer software industries are becoming more influential in the telecommunication management domain. One problem with the development of management systems is that they frequently need to adopt several standards, rather than one single standard. This multi-standard approach is illustrated in the TeleManagement Forum's management process areas (e.g. Fulfilment, Assurance and Billing) [NGOSS04]. Frequently, several standards with their associated information models and protocols would be relevant to a management area. For example, performance management applications may use the eTOM fulfilment information model [TMF] and its specification for the representation of the performance information, but may also need to be consistent with IEFT or TMN standards for network and element management modelling.

The technological advancements in telecommunication is forcing a trend towards unification of network and services, setting up a stage for the emergence of Next Generation Network (NGN). NGN is essentially an IP based network that enables customers to receive a wide range of services such as voice, data and video over the same network. The services provided by the NGN are independent of underlying network and access is enabled across a wide range of broadband technologies, both wireless such as 3G, Wi-Fi, WiMax and wire line.

NGN operates in a very dynamic environment. Services provided by the NGN infrastructure need to be updated and improved continuously. Devices are added, removed and configured/re-configured in the transport network, making the management of NGN a challenging task. NGN might be considered as one network, but it is by far the most complex of all. Its management has to deal with multiple vendors, multiple applications, multiple physical devices from data and voice networks, multiple databases, and multiple service layers. Any management solution for NGN must be architected in a way that it can scale to manage, adapt to and support current, emerging and future services and technologies without the need for long term and complex upgrades. In the past, many different approaches have been proposed in order to solve the problem of integrating management systems but these approaches did not scale up and the business model that was used by each connected system was hard to organize with too much dependencies and centralization [ADAM98], [VINO97], [TRIM01], [BOHO02] [REDL98].

The NGN management plane should be flexible and scalable enough in order to accommodate heterogeneous legacy management systems as well as new generation management systems that span across different layers of the NGN infrastructure and need to be operate as one agile entity. Moreover, the NGN management architecture should be able to reduce the complexity of the involved management systems, increase the potential for reuse of management functionality and increase the speed of development and deployment of these systems. In addition, the level of automation of management system needs to be high in order to provide greater capability and to manage higher levels of complexity in networks and systems. The management architecture needs to adopt mainstream information technologies and development techniques rather than maintaining a reliance on telecommunication specific technologies [KOTS08].

1.2 Aims and objectives

The aim of this thesis is to design and specify a network management architecture that focuses on managing large scale heterogeneous telecommunications environments, such as NGNs. More specifically, the thesis proposes a management framework that integrates heterogeneous management systems in a loose coupling way. The key benefit of the proposed management architecture is the reduction of the complexity that derives from integrating heterogeneous management systems.

In order to achieve this aim, the following objectives are pursued:

- To examine the standardization frameworks related to the management of telecommunication networks.
- To investigate the technologies used for integrating traditional management networks as well as NGNs.
- To propose a management framework based on the Service Oriented Architecture (SOA) philosophy for the integration of heterogeneous management systems in a loose coupling manner.
- To develop a Network Management System that is based on the web service technology.
- To design a Network Management Middleware Layer that can simplify the task of bridging distributed management systems.
- To develop a Network Management Middleware platform that is based on the Enterprise Service Bus.
- To develop a testbed in order to test the performance of the Network Management Middleware platform.

1.3 Research contributions

This thesis contributes to defining a management framework, based on the SOA principle that can be used as the foundation management infrastructure for NGNs. The following summarizes the original contributions of this thesis in the design and development of the NGN management infrastructure:

- The design and development of a Network Management System (NMS) that follows the principles of SOA and exposes network management functionalities as Web Services. Moreover, an agent-

based model has been developed based on the SNMP framework. The agents reside in Network Elements to collect performance, faults, and configuration management information from them.

- The design and development of an XML-based gateway that exposes the management information in a common XML-based message format, paving the way for interoperability. This gateway converts management information into XML-based messages to enable management information retrieval from any NMSs.
- The design and development of a Network Management Middleware Layer based on messaging and asynchronous communication that removes the integration complexity from the management systems. Moreover, it handles the heterogeneity on the information expressed by legacy management systems that do not conform to web service standards. Original contributions include the design and development of the following service components:

    a. Transformation Service that transforms management information into a common information model. This transformation contains message decomposition with needed information (i.e. metadata).
    b. Validation Service that validates management information.
    c. Content-based Routing Service that determines the destination of each management message based on the content of the message to categorise messages into management topic queues.

d. Finally, a Persistent Store and Message Archive Service that keeps the record of every message sent by management systems to increase reliability.

- A trouble ticketing system has been developed as a part of the overall proposed architecture. It has been used as a Management Service in order to consume management information provided by the Network Management Middleware Layer.
- A testbed has been developed in order to test the performance and behavior of the Network Management Middleware Layer. Several experiments have been conducted in order to evaluate the behavior of the proposed SOA-based management platform.

1.4 Structure of the thesis

The thesis consists of 7 chapters. The description of each chapter is as follows:

Chapter 2 identifies the business drivers for the telecommunication management community. It analyzes the standardization bodies which define key management functionalities and architectures that have influenced the design of the telecommunication management systems. In addition, this chapter pinpoints the architectures' contributions and influences in the design of management systems.

Chapter 3 investigates the technologies that have been used by the telecom operators for integrating their networks. This chapter concluded that these approaches are not capable of supporting the NGN's management plane. It further illustrates that the focal point of the

telecommunications networks is now shifting from traditional architectures to SOA-based architectures. Moreover, this chapter introduces the SOA concept as well as the technologies that enable it. Finally, the proposed Network Management Platform that has been designed based on the architectural principles is presented in this chapter.

Chapter 4 presents the management communication (Low Level Management Communication) between network devices. In more detail, the design and the development of an NMS that is based on web service technology and performs fault, performance and configuration management functions is presented. Moreover, the design of an XML Gateway that converts management information into XML and sends the information to other applications is presented.

Chapter 5 describes the design and develop a Network Management Middleware Layer that is based on messaging and asynchronous communication. Several service components have been created in order to enable the communication and transfer of management information of heterogeneous NMS systems.

Chapter 6 presents a trouble ticketing system that has been developed as a Management Service of overall proposed architecture with the aim to demonstrate how the Network Management Middleware Layer can expose heterogeneous management information for consumption by a Management Service. This chapter also includes tests that have been performed in order to evaluate the performance of the proposed Middleware Layer. Several test scenarios have been derived and experiments are conducted to examine the behavior of the proposed

Network Management Platform. Furthermore, a theoretical analysis is presented in this chapter to illustrate the architectural design considerations that have been used in order to meet the SOA principles.

Chapter 7 draws the overall conclusions and lists a set of possible future activities from various research directions.

1.5 Publications from Thesis

During the development of the thesis, the research was peer reviewed and published in international research conferences and journals. The publications which are based on the research in this thesis are:

- K. Kotsopoulos, P. Lei, Y.F. Hu, “Managing NGNs using the SOA Philosophy”, IEEE Int. Conf. Innovations in NGN: Future Network and Services, First ITU-T Kaleidoscope Academic Conference. ISBN: 978 92-61-12441-0, pg. 47-54, Geneva, 12-13 May 2008.
- K. Kotsopoulos, P. Lei, Y.F. Hu, “SOA-based Information Management Model for Next-Generation Network”, IEEE International Conference on Computer and Communication Engineering ICCCE08. ISBN: 978-1-4244-1691-2, pg.1057-1062, Kuala Lumpur, 13-15 May 2008.
- K. Kotsopoulos, P. Lei, Y.F. Hu, Book Chapter: “The adoption of Service-Oriented Architecture (SOA) in managing Next Generation Networks (NGNs)”, Handbook of Research on Heterogeneous Next Generation Networking: Innovations and Platforms, IGI Global, 2008.
- Y.F. Hu, M. Berioli, P. Pillai, H. Cruickshank, G. Giambene, K. Kotsopoulos, W. Guo, P.M.L. Chan, “Broadband Satellite Multimedia”,

IET Communications. ISSN: 1751-8628, Volume 4, Issue 13, pg.1519-1531, Sep. 2010.

Chapter 2 : THE EVOLUTION OF TELECOMMUNICATION MANAGEMENT FRAMEWORK

2.1 Introduction

The growing demand for customer-controlled management of services combined with the growing convergence of telecommunications, computing and entertainment brings new challenges to telecommunication network operators and service providers. To manage the converged communication networks, an effective and efficient telecommunication management infrastructure is fundamental. When developing a telecommunication management framework, architectures derived from standardization bodies need to be considered and examined. Telecommunication management architectures specified by some of the de-facto standardization bodies such as the International Telecommunication Union-Telecommunication standardization sector (ITU-T), Internet Engineering Task Force (IETF), TeleManagement Forum (TMF), Telecommunications Information Networking Architecture (TINA) and Distributed Management Task Force (DMTF) are widely accepted by the telecom industry. This chapter identifies the key actors and their roles involved in the telecommunication management field. The evolution of the network and service management as well as the future directions of the telecommunication management development can be defined by understanding their architectural fundamentals and their limitations.

2.2 Drivers for the Telecommunication Management Community

The telecommunication industry has been greatly influenced by standardization bodies. Standardization bodies set out policies and practices for fair competition to reduce the cost and also avoid the 'network vendor lock-in' problem, where telecommunication providers are forced to use proprietary management systems [TARK09]. Moreover, the telecommunication marketplace is continuously changing due to the technological innovations, increasing competition and deregulation [EURO04]. Deregulation forces traditional telecommunications services (fixed telephony) to provide less expensive products and services [EURO04]. Thus, there is a greater need for interoperability between telecommunication network operators and network management frameworks in order to overcome these challenging factors [DAVI99].

The vision of NGNs has become a realisation and is expected to run for the next decades [M.3060]. A challenge that is crucial for the establishment of NGNs is to build an appropriate architecture for operation, administration and maintenance across of network elements of a diverse range of telecommunication networks. Currently, several initiatives and projects have set up to investigate the management issues of the NGN infrastructure [EURO06], [M.3060]. Furthermore, the massive increase in Internet usage result in rapid growth and demand for Internet protocol (IP)-based services to be supported by the telecommunications industry. This leads to a more complex and diverse ICT (Information and

Communications Technology) marketplace. The convergence of telecommunication networks and the ICT networks based on IP brings new challenges to the standardization bodies involved in the network management activities. Figure 2.1 illustrates the separation of the telecommunication networks and the Data networks. Each network provides its own technology and management frameworks. Figure 2.1 shows the management of telecommunication networks based on frameworks such as the Telecommunication Management Network (TMN), Common Management Information Protocol (CMIP) and TINA. Data networks similarly provide their own set of protocols and management specifications for managing their own networks such as the Web Based Enterprise Management (WBEM) framework and Simple Network Management Protocol (SNMP).

Furthermore traditional telecommunication networks operate on self-contained and highly managed, real time networks in order to provide a deterministic quality of services to their users. If no enough resources are available, the service request by users will be rejected. The telecommunication providers not only deliver sophisticated quality of services based on the Internet protocols but also support real time functions such as authentication, location determination, user registration, real time pricing, bandwidth management etc. that distributed over a number of servers. On the other hand, the Internet consists of a '*loose*' federation of network operators that provide 'best effort' service over shared data network infrastructure [JENK06].

The TMF's New Generation Operations Systems and Software (NGOSS) and TMN's NGN specifications attempt to bridge the gap of the Telco and ICT marketplaces [EURO06], [M.3060]. Convergence technologies such as Distributed Object Technologies (DOT) [HENN06] and Service Oriented Architecture (SOA)-based frameworks [ERL09] [ERL10] are the intermediary technologies that can integrate the management frameworks and remove the boundaries of the two marketplaces. These convergence technologies will be discussed in the next chapter.

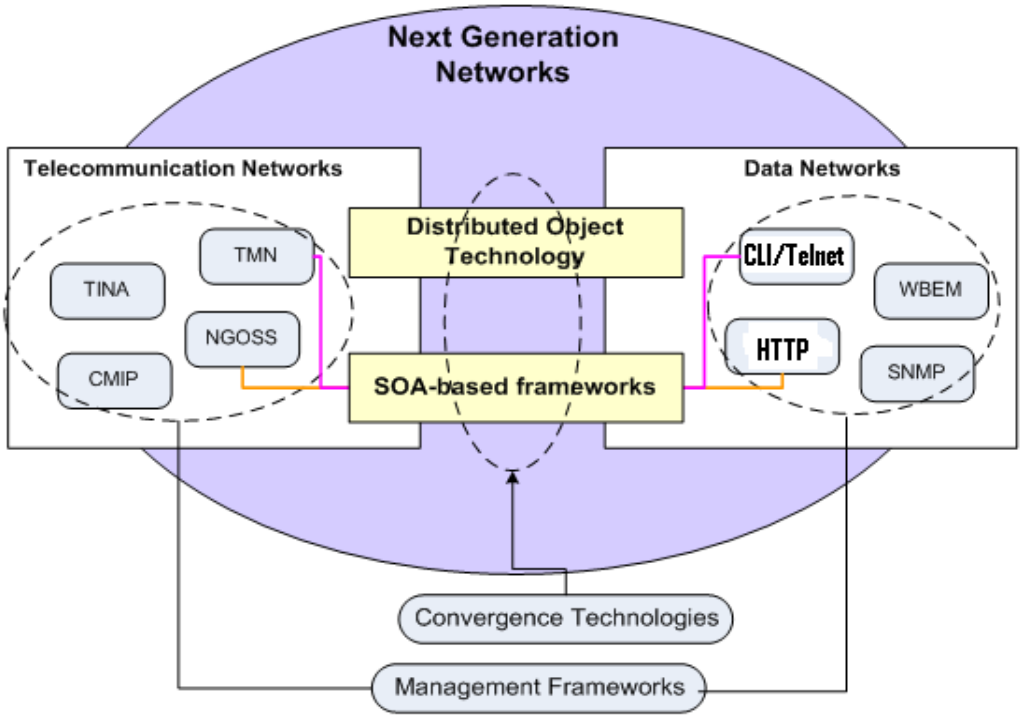


Figure 2.1: Management frameworks in Telco and ICT marketplace

The late 1980s and early 1990s are dominated by two key network management standards: the Open System Interconnection (OSI) with the CMIP framework adopted by telecommunication industry, and the IETF

with the SNMP framework that is widely used in data network. In the early 2000, ICT enterprise management and telecommunication management started to converge [MORA02]. This happened due to the globalization of markets, the deregulation of the telecommunication markets, the increasing Business-to-Business (B2B) transactions and the increasing demand to lower operational costs and increase the software reuse. Moreover, the domination of the IP, as a common communication protocol for local and wide area networks meant that telecommunication management standard bodies had to adopt specifications and implementations derived from computer industries rather than those specified by telecommunication sectors. Figure 2.2 illustrates an example of the convergence of telecommunication network and data network. In this example, a data service and a voice service simultaneously traverse a data network and a telecommunication network. The data network is managed by a service provider A and the telecommunications network is managed by a service provider B.

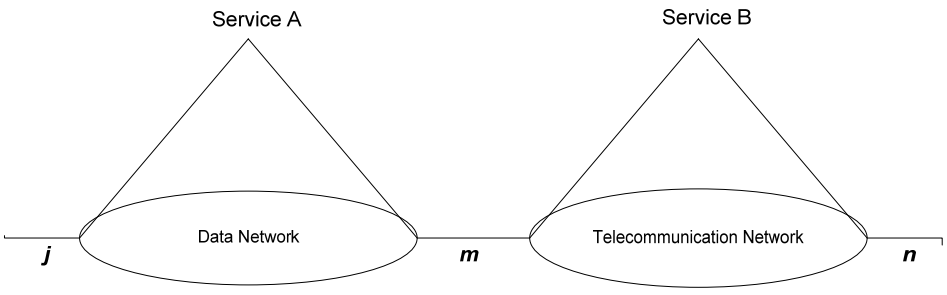


Figure 2.2: Convergence of telecommunication network and data network

The overall Quality of Service (QoS) from one end to the other (*j* to *n*) is difficult to calculate because the QoS of the data network (from *j* to *m*) has a different QoS definition from that defined by the telecommunications network (from *m* to *n*). Moreover, a customer that uses both services does not have any knowledge about the networks that he uses. The customer is not aware of the differences in the QoS provided between the two networks. The cause of poor QoS might lie in one service provider's networks but who is responsible for his QoS? As a result, management information needs to be able to transverse from one service to another. Management systems need to be integrated for this reason.

With the advent of NGN and the convergence of different transport technologies, the management plane needs to be able to provide management functions across heterogeneous and geographical distributed systems. From the telecommunication provider's point of view, the main business drivers for the management functionality are

- to improve the process flow across the organization,
- to increase the management process automation,
- to improve the Quality of Service (QoS) of the service management,
- to reduce the cost of service provisioning
- to have tighter customer management control.

With these business drivers in mind, the telecommunication providers need to be able to create Operational Support System (OSS) solutions from reusable management components that are available from different vendors. These management components (standardized or proprietary) need to ensure the integrity of the information flows and to satisfy the end-

to-end processes in order to meet the business and operational requirements of the telecommunication provider. Standard bodies such as the DMTF are now promoting the use of open source APIs when developing a standard [WEST00]. By adopting open source frameworks vendors can speed up the development process.

2.3 An Overview of Telecommunication and Network Management Architectures

This section presents standardization frameworks involved in the design of telecommunication management systems. Furthermore, this section identifies the contributions and influences of these architectures. More specifically, TMN, SNMP, CMIP, NGOSS, TINA and WBEM architectures that play important role throughout the spectrum of the management plane starting from the low layer network management to the high layer service/business management, are examined. Figure 2.3 gives an overview of the evolution of the management frameworks.

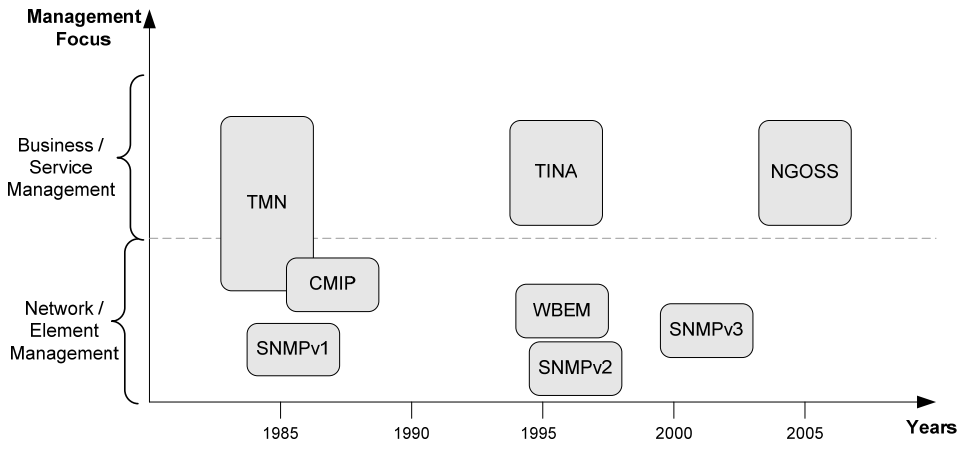


Figure 2.3: Evolution of the management frameworks

In figure 2.3, the *y* axis represents the management focus of the management framework and the *x* axis represents the timeline of the framework's establishment. There can be seen that some management frameworks are focused on the Network and Element management such as SNMP and CMIP, and other management frameworks are more focused on the Service and Business management.

2.3.1 Telecommunication Management Network (TMN)

2.3.1.1 The TMN Reference Architecture

In 1986, the ITU-T proposed the concept of a TMN model in order to address the interoperability of multi-vendor equipment used by service providers and to define standard interfaces between service provider operations [M.3010]. It defines a generic, management-oriented architecture that can be applied to all kinds of management services. Furthermore, the organization extended the concept of management to include not only networks and network elements, but also service functions of the service providers [PAV97]. The architecture uses concepts from the OSI Systems Management architecture and applies them in the context of telecommunication management [M.3010]. Figure 2.4 depicts the TMN architecture.

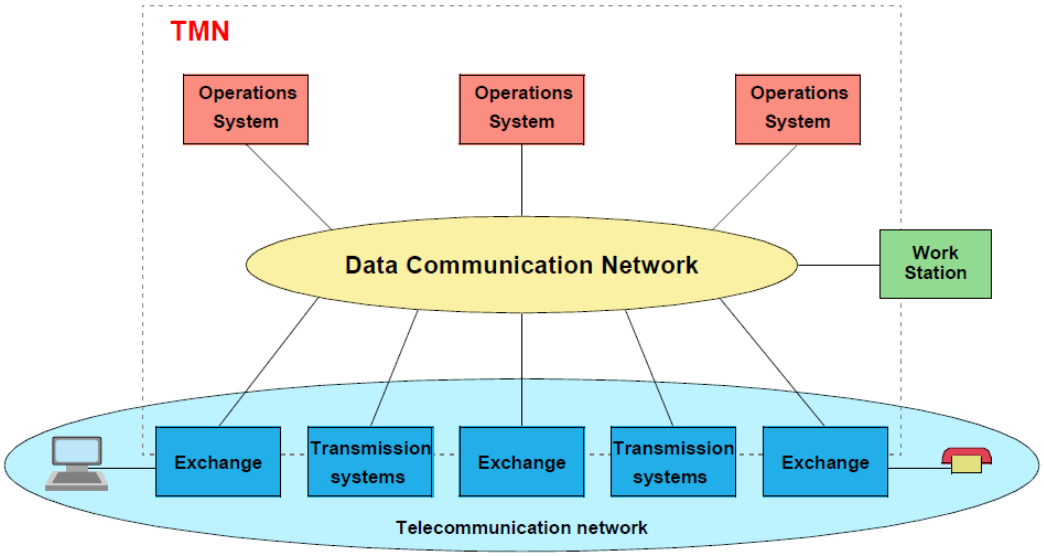


Figure 2.4: The architecture Telecommunications Management Network

The TMN provides an organized architecture to achieve the integration between various types of Operating Systems (OS's) and/or telecommunications equipment for the exchange of management information using an agreed architecture with standardized interfaces including protocols and message [M.3010] as shown in figure 2.5.

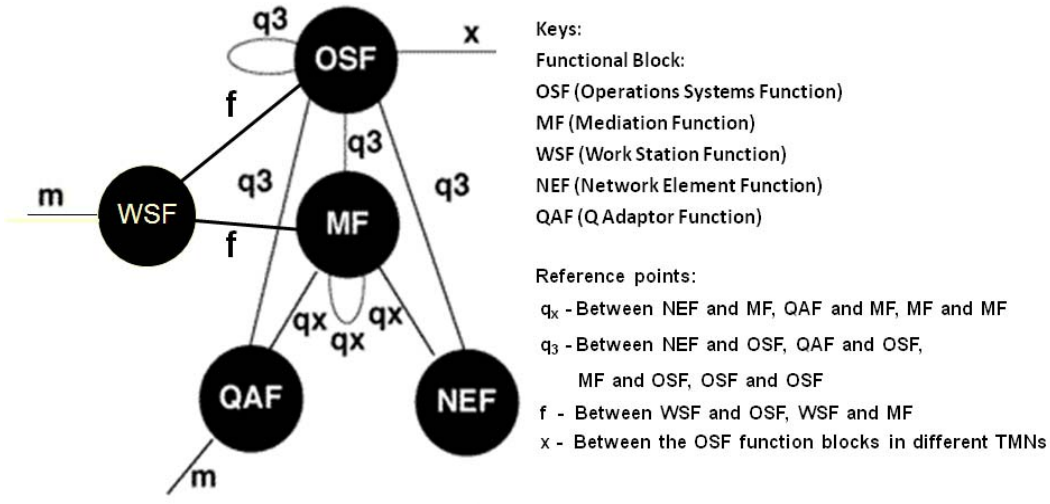


Figure 2.5: TMN function Blocks and Reference Points [M.3010]

TMN describes telecom network management from several viewpoints: a logical or business model, a functional model, and a set of standard interfaces. The functional architecture breaks down the management functions into function blocks. The TMN functional architecture describes the realization of a TMN in terms of different categories of function blocks and reference points among these blocks [RAMA97]. The TMN physical architecture corresponds to the physical realisation of the functional architecture. Each function block becomes a physical block, and reference points are transformed into interfaces. The Operation system (OS) is an important physical block for managing the telecommunication activities. The most important interfaces are: Q3 to link up OS with the managed resource and X interface to integrate two TMNs of different OSs.

TMN makes use of OSI Systems Management principles that is based on an object-oriented paradigm [M.3060]. In the TMN information architecture, resources are modelled using object-oriented concepts at different levels of abstraction and follow the GDMO (Guidelines for Definition of Managed Objects) and ASN.1 (Abstract Syntax Notation One) specifications. The Managed Object (MO) operations are based on the manager-agent model in which manager issues operation directives and receives notifications, the agent responds to directives and emits notifications related to MO's. The details of the manager-agent model will be discussed in section 2.3.3. The information model is fully Object-Oriented framework that can be mapped to Object-Oriented Programming Languages such as C++ [M.3020].

2.3.1.2 TMN Layer Separation

The TMN management architecture proposes the separation of the management functionality into five hierarchical layers. The ITU-T M.3010 gives a well established categorization of management layers. These layers range from lower layers that involve managing details of individual pieces of network equipment, to higher layers that are closer to the running of the business that the network supports.

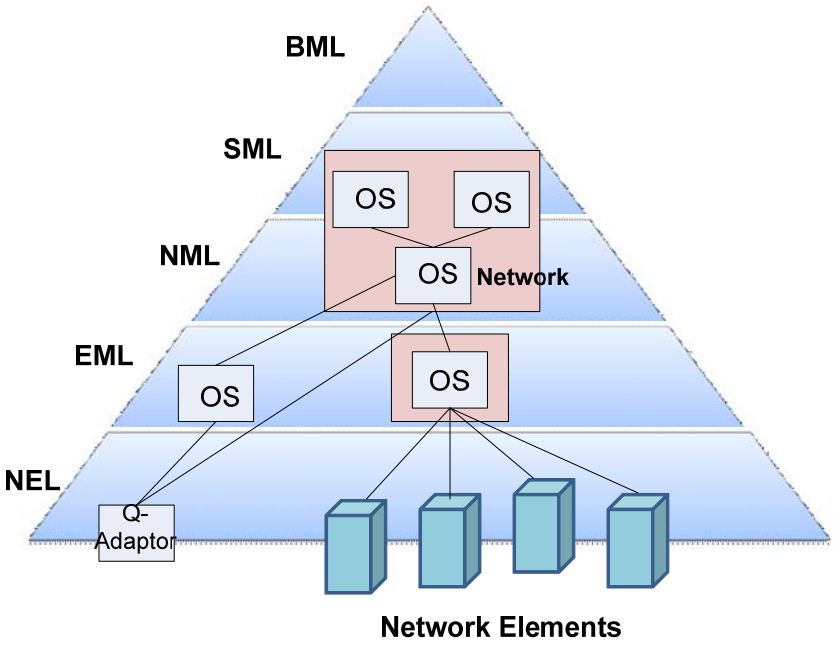


Figure 2.6: TMN logical Layer [M.3010]

The main contribution of TMN framework is the definition of a logical model that specifies the functionality of each management level:

- Business Management Layer (BML) concerns with High-level planning, budgeting, goal setting, executive decisions, business level agreements (BLAs), etc.

- Service Management Layer (SML) uses information presented by NML to manage contracted service to existing and potential customers for service provisioning, accounts, quality of service, and fault management.
- Network Management Layer (NML) has visibility of the entire network, based on the Network Elements (NE) information presented by the EML OSs. The NML manages individual NEs and all NEs as a group.
- Element Management Layer (EML) manages each network element and is responsible for the TMN-manageable information in certain network elements.
- Network Element Layer NEL presents the TMN-manageable information in an individual NE. Both the Q-Adapter, which adapts between TMN and non-TMN information, and the NE are located in the NEL.

This logical layer categorization has influenced other management standardization organizations such as Tele-Management Forum and has been adopted by many network management vendors such as Hp Openview and IBM [OPENVIEW], [IBM].

2.3.1.3 The FCAPS Model

In addition to the layering structure, the general management functionality in TMN is classified into five functional areas: Fault, Configuration, Accounting, Performance, and Security (FCAPS) as follows [M.3060]:

Fault management monitors any failure events and request tests to be performed in order to isolate these faults.

Configuration management provides functions to exercise control over, identify, collect data from and provide data to network elements.

Accounting management enables the measurement of the use of network services and the determination of costs to the service provider and charges to the customer for such use.

Performance management monitors the performance of the entire network.

Security management should minimize unauthorized or accidental access to network control functions. Security management functions deals with ensuring legitimate use, maintaining confidentialities, and data integrity.

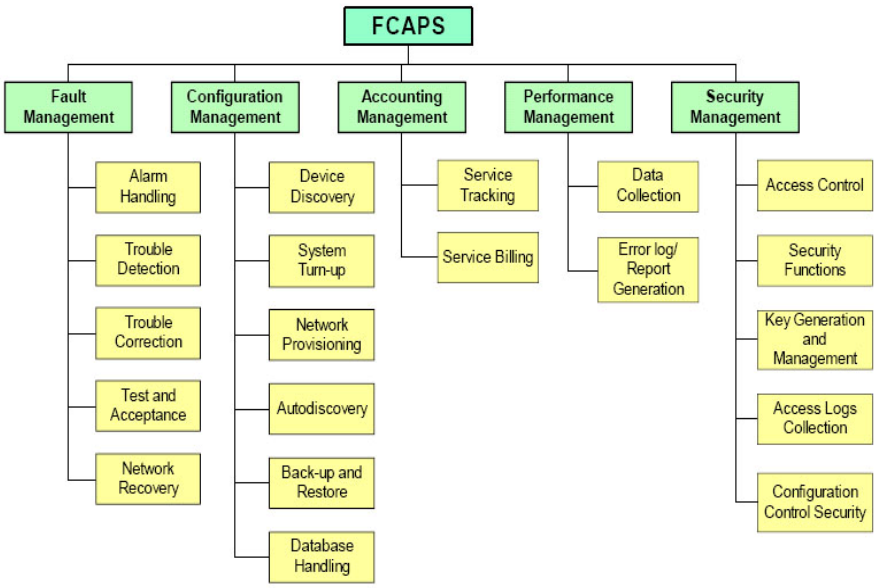


Figure 2.7: TMN FCAPS Model

Figure 2.7 illustrates the detailed view of the functions performed by the FCAPS model. These functions need to be performed at all the TMN's logical layers since telecommunication network management is evolved towards on meeting Service Level Agreements (SLA), which demands 99.99 percent availability on the network [SATM09]. As a result, service and network functions performed at different layers need to be managed according to the FCAPS model in order to provide an overall picture of the network's health and minimize the risks of failing to meet the SLAs.

2.3.1.4 TMN Contributions and Influence

TMN was envisioned as a solution to the complex problems of operations, administration, maintenance and provisioning (OAM&P), [FOWL95] and provided a generic network management framework. EURESCOM in the Next Millennium report [EURE99] concludes that the TMN concept has not been widely used by the Telecom industry due to the high resource requirements, technical complexity and the popularity and simplicity of other management standards such as SNMP. Moreover, TMN management framework produces expensive management applications with complex APIs (Application Program Interface). Its protocol stack is considered comprehensive but it brings more complexity and is considered as a heavy weight protocol stack [EURE99]. Furthermore, legacy equipments have to convert their legacy interfaces to TMN-based interfaces. This is an expensive process to do because each TMN interface is related to a specific protocol layer in the OSI reference model,

as a result each legacy interface has to make interface conversion to all TMN-based OSI layers.

2.3.2 The Telecommunication Information Network Architecture (TINA)

2.3.2.1 The TINA Development

TINA was developed by a consortium made up of over 40 companies, such as telecom vendors, telecom operators and service provides [TINA]. The aim of the consortium was to define an open architecture for telecommunication systems for broadband and multimedia communication. TINA focuses on building a distributed processing environment, especially for provisioning and deploying global services in near real time to meet the market demands. The first phase of its development (1993-1997), was aimed at defining a global architecture for telecommunication systems with advanced software technology. The second phase (1998-2000) defined specifications and initiated activities to coordinate the activities involved in the business model and the service architecture. A major design principle for TINA was the use of distributed computing (e.g. Open Distributed Processing (ODP) that adopts the Common Object Request Broker Architecture, CORBA to avoid the scalability problem faced by centralised computing [HUBA98]

2.3.2.2 The TINA Business Model

The TINA business model identifies various business stakeholders and their roles involved when considering a virtual marketplace for services

and networks. TINA attempted to standardise the relationships and interfaces between the different business roles, referred to as TINA Reference Points (RP). The business model provides mechanisms to specify, add and modify RPs and roles in the TINA system. These mechanisms provide one framework of common business that defines a set of conditions on which the creation of new business roles and RPs can be made. In addition, this framework provides an initial set of business roles and relationships to apply the TINA methodology, and requirements imposed by TINA system to cover a particular set of services.

TINA identifies the following five stakeholders in the business model:

- Consumer
- Broker
- Retailer
- 3rd Party Service Provider
- Connectivity Provider

The consumer establishes contractual relationships with the Retailer stakeholder, which represents a “one-stop shop” for TINA services for Consumers.

The Broker’s role is to provide stakeholders with the information that they need to find other stakeholders and services in a TINA system. It is considered as a directory service provider that can be accessed globally by any stakeholder. In addition, the Broker handles subscription, accounting and security and keeps track of object interfaces.

The Retailer can either provide the TINA service autonomously or can make use of 3rd Party Service Provider to offer a service. For example, a mobile phone service can provide the information about weather report, but the content is actually dynamically sourced from a 3rd party content provider and not from the Telecommunication provider [YATE97].

The Connectivity Provider is responsible for managing the transport network and for offering a technology independent connectivity service to the other business roles.

Figure 2.8 shows the relationships between the different stakeholders.

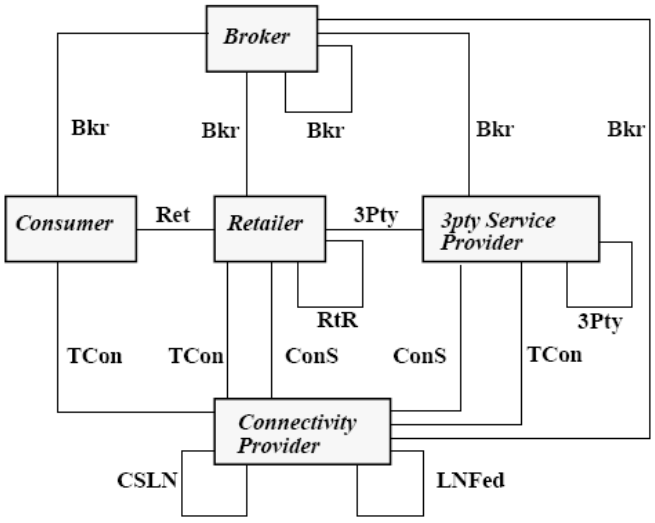


Figure 2.8: TINA Business Model

The Retailer RP (Ret) identifies the relationship between Consumer and Retailer; the Broker RP (Bkr) identifies the relationship between the Broker and other stakeholders such as Retailer, Consumer, 3rd Party Service Provider and Connectivity Provider. 3rd Party Service Provider (3Pty) RP defines the relationship between Retailer and 3rd Party Service Provider as

well as relationships among other 3rd Party Service Providers. The Connectivity Service (ConS) RP and Terminal Connection (TCon) RP identify the relationships between Connectivity Providers and Retailer, Consumer and 3rd Party Service Providers. The Client Service Layer Network (CSLN) RP and the Layered Network Federation (LNFed) RP define the relationships among Connectivity Providers in supporting cooperative connectivity across providers.

2.3.2.3 TINA Service Architecture

TINA's service architecture defines a platform for developing a wide range of services in a multi-supplier environment [PAVL98]. This platform consists of application software components, which are deployed on a Distributed Processing Environment (DPE) [PAV97]. The DPE provides a technology independent view of computing resources, allowing technology dependent aspects in applications software to be minimized. In this way, it supports the construction of portable, interoperable code and promotes easier software design and reuse.

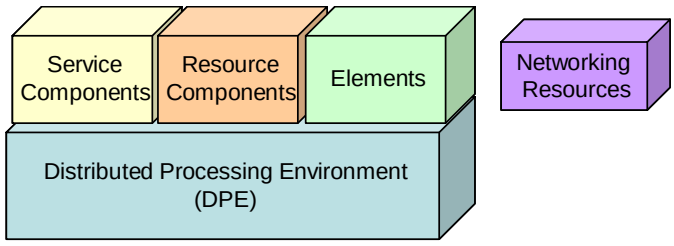


Figure 2.9: TINA Components

The application components in the TINA architecture are divided into three categories in order to achieve good structure, modularity and software reusability. These three categories are as shown in figure 2.9,

- Service components
- Resource components
- Element components.

Service components address the core functionality of TINA services, including access and management capabilities. These components are deployed in the domains related to Consumer, Retailer, Broker and 3rd Party Service Provider. Service components that require a connectivity service can use facilities provided by Resource components. Resource components are deployed within the Connectivity Provider stakeholders' administrative domains and offer high-level technology-independent abstractions of the underlying transport network in order to utilise and manage the network's resources. Element components are software representations of physical or logical resources such as switching fabrics and transmission equipment. The identification and definition of individual element components is outside of the scope of TINA framework.

TINA defined a Network Resource Architecture (NRA) to provide a set of generic concepts in order to describe transport networks in a technology-independent manner. It is concerned with how individual elements are related, topologically interconnected, and configured in order to provide and maintain end-to-end connectivity. The NRA is heavily influenced by the TMN standards [PAVL98].

The major differences between TMN and TINA

- TMN aims at integration, TINA assumes DPE.
- TMN focuses on process interactions and interface agreements, TINA is more architecture and component driven.

2.3.2.4 TINA Architecture's Contribution and Influences

The TINA framework stopped on 2000 but has subsequently influenced other standardization organizations as well as the Telecom industry. More specifically, TINA promoted a number of issues related to telecommunication management that are progressed by other organizations. These issues are: a critical analysis of the business stakeholders and their relationships, its expressed objective towards component-based architectures; and the use of mainstream distributed middleware services to support management systems and component communication. TINA's adoption of mainstream distributed object technology has been implemented by many management system developers and vendors [VALL99]. TINA represents a revolutionary departure for the telecommunications industry that is characterized by a shift from protocol-based telecommunication engineering principles to software engineering techniques such as APIs, and component interface specifications which are more closely related to the programming languages used to implement the service logic [PAV97]. Finally, the TINA business model has influenced the stakeholder representation in the eTOM specification from the TeleManagement Forum [LEW99].

2.3.3 The Manager and Agent Model

OSI management has introduced the manager-agent model [SART95]. This model is the most common model that is being used for management purposes [SART95]. SNMP and CMIP are two network management protocols for managing devices that based on the Manager-Agent model.

According to the model, manageable resources are modelled by managed objects that encapsulate the underlying resource and offer an abstract access interface. The management aspects of entities such as Network Elements are modelled through “clusters” of managed objects. A management interface is defined through a formal specification of relevant managed object types and the associated access mechanism. Management interfaces can be thought as “export” by the agents and “import” by the manager. Manager access managed objects across interfaces in order to implement management policies. Figure 2.10 illustrates the Manager-Agent model.

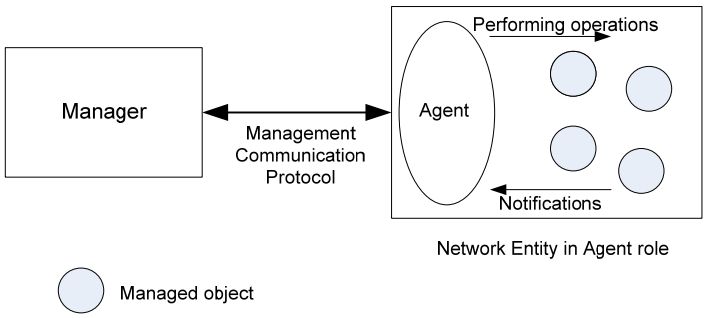


Figure 2.10: Manager-Agent Model

The management access service and protocol carries the parameters of operations to managed objects and returns management results. The management parameters and management results are a subset of other available objects residing on the agent’s boundary. The agent offers a database-like facility (MIB Data Store) which has the effect that one operation may result in many operations to managed objects inside the agent, with a combined result passed back to the manager. The managed objects can use the agent’s notification mechanism in order to send

notifications (called traps) to the manager according to criteria that the manager has preset.

2.3.3.1 Network Management Agent

A network element must have a management interface in order that an NMS can communicate with it for management purposes. For instance, the management interface allows the NMS to send requests to the network element. A request could be a configuration of a sub-interface, to retrieve statistical data about the utilization of a port, or to obtain information about the status of a connection. In addition, the network element can send information to NMS, such as a response to a request, but also to send a response when an unexpected event (for example, the failure of a fan or a buffer overflow) has occurred. Management communication is asymmetrical. This means that a managing application is the “manager” which is in charge of the management, and the network element is the “agent” that supports the manager by responding to its requests and notifying it proactively of unexpected events. Figure 2.11 illustrates the interaction between NMS and network entity node.

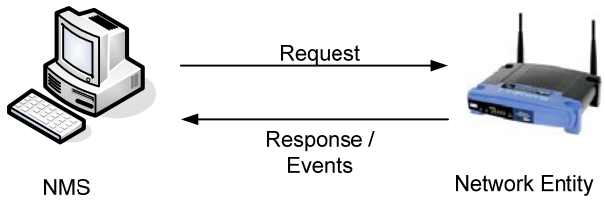


Figure 2.11: Interaction between NMS and Network Entity

The management agent consists of three main parts: a management interface, a Management Information Base, and the core agent logic.

- The management interface handles management communication. It supports a management protocol that defines the “rules of conversation” for communication between the managed network element and the NMS. It allows the NMS to open a management session with the network element. In addition, the management interface allows the NMS to make requests to the managed device and receive responses. Through the management interface, the management agent can send event messages that the NMS can receive. The Event message enables the manager to be alerted of certain faults at the network element, such as unexpected communication loss with another network element.
- The MIB is a data store that contains a management view of the device that is being managed. The data contained in this data store form the management information. The MIB is not a database in which information about the device is stored but is a way to view the device itself. For instance, when a managing application would like to modify an entry in the conceptual table, in reality, the actual configuration of the network element is changed and the communication behaviour of the network element is changed.
- The core agent logic is the function that translates between the operation of the management interface, the MIB, and the actual device. For instance, it translates the request to “retrieve a counter”

into an internal operation that reads out a device hardware register that contains the desired information. Many counters of the same type could exist inside the network element, for example, one counter per interface. Therefore, the agent logic must be able to map the name by which the counter is referred to in the MIB to the actual register whose contents are being requested. Agent logic can include added management functions that offload the processing required by the NMS.

2.3.3.2 Structure of Management Information (SMI)

The formats of the information exchanged between a manager and an agent needs to be the same for any implementation. In order to achieve a uniform representation of the information delivered over the network, management protocols such as SNMP use a subset of the Abstract Syntax Notation One (ASN.1) for the data presentations and this subset is known as Structure of Management Information (SMI) [McCL99]. ASN.1 [X.690] provides a standard way of representing data travelling across the internet. This standardization is necessary because the data can be represented in incompatible ways within different network computing devices. ASN.1 is used in order to describe the format of how messages can be sent between agents and NMSs. The SMI is not only used to define the formats of the messages exchanged by the management protocol but also used to define the managed objects. SMI provides a way to define managed objects and their behavior. The agent has in its possession a list of the objects that it tracks. One object for example is the

operational status of a router interface (for example *up*, *down* or *testing*). This list defines the information that NMS can use to determine the overall health of the device on which the agent is located in. Figure 2.12 shows the OBJECT-TYPE macro that is used to define the elements in the MIB.

```
OBJECT – TYPE MACRO :: =
BEGIN
      TYPE NOTATION ::= “SYNTAX” type (type ObjectSyntax)
                              “ACCESS” Access
                              “STATUS” Status
VALUE NOTATION ::= value (VALUE ObjectName)

Access ::= “read-only”
                  | “read-write”
                  | “write-only”
                  | “not-accessible”
Status ::= “mandatory”
                  | “optional”
                  | “osolete”
```

Figure 2.12: SMI OBJECT-TYPE macro

The SMI specification in RFC 1155 [ROSE90], defines the general framework which a MIB can be defined and constructed. It identifies the data types that can be used in the MIB and specifies how resources within the MIB are represented and named. The philosophy of SMI is to provide simplicity and extensibility within the MIB. As a result the MIB can store only simple data types: scalars and two-dimensional arrays of scalars.

2.3.3.3 Management Information Base (MIB)

Management protocols provide the ability to query devices on the network. The communication with the device can be done by retrieving information from the MIB which is contained in the device. The SMI provides the way

to define the managed objects, while the MIB is the definition (using the SMI syntax) of the objects. The leaf objects of the tree are the actual managed objects, each of which represents some resources, activities or relevant information that can be managed. The tree of structure defines a grouping of objects into logically related sets. MIB is best thought of as a conceptual data store. The MIB is not the same as a database. The MIB does not store information about the real world (the actual managed device) in a file system; instead, it offers an abstraction of the managed device that is used for management purposes. When the manager retrieves some information from the MIB, it represents an aspect of the device. For example, an internal register that keeps track the number of packets that has been received over a port. When the manager manipulates the information in the MIB, the actual settings of the device are modified, as a result, affecting the behavior of the device. Management information provides the capability that network managers need to control and manage the device. MIBs are one of the central concepts in network management [STAL98] [KAVA00].

The following figure (figure 2.13) illustrates the structure of the MIB.

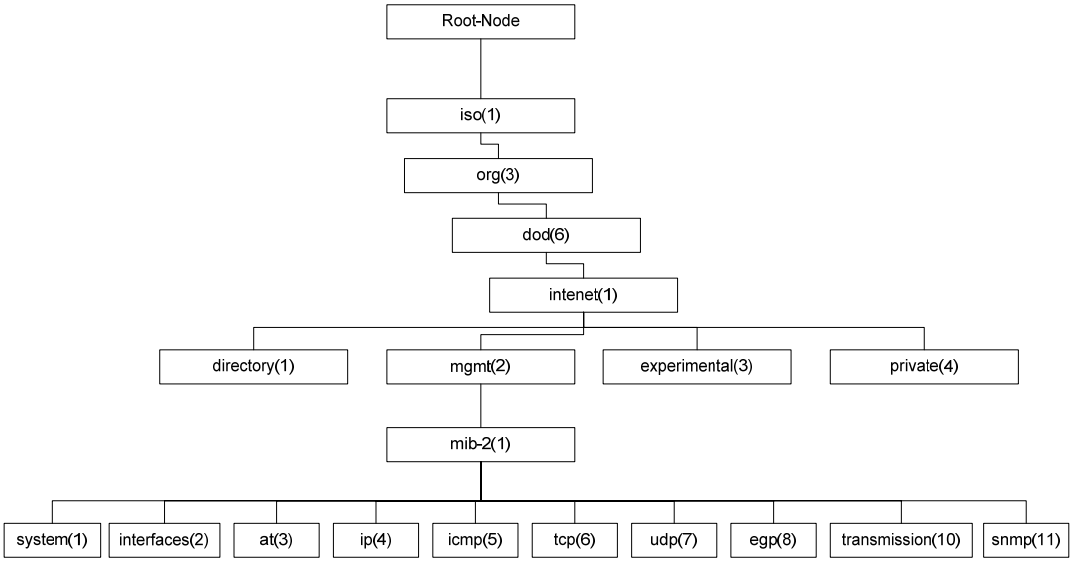


Figure 2.13: Structure of an MIB

The current full standard for the MIB is defined in the RFC 1213 [McCL91]. This version is called MIB-II and has been evolved from the previous specification MIB-I. The MIB-II structure is divided into groups that reside in four layer OSI protocol suite model. There are ten groups in the MIB-II definition. Those groups are listed below:

- System group
- Interfaces group
- Address Translation group
- Internet Protocol group
- Internet Control Message Protocol group
- Transmission Control Protocol group
- User Datagram Protocol group
- Exterior Gateway Protocol group
- Transmission group

- The Simple Network Management Protocol group

The System group has three objects. These objects contain descriptive information about the managed Network Elements. It describes the top level characteristics and general configuration information about the managed Network Elements. Every object of this group is mandatory. If an agent is not configured for a value for any of these objects, the objects must have a default initialization value of 0.

The Interfaces group objects deal with the Network Element's lowest level of connection to the network. This group allows the management control of the lowest layer of the TCP/IP protocol suite. Since NEs could have more than one network interface, Interfaces group provides a count of the number of interfaces present in the Network Element and related information about each interface.

The Address Translation group provides a mapping of a Network Element's internetwork layer address e.g. IP address. The Internet Protocol group contains the managed objects for providing information on IP operations, such as IP routing tables and address conversion tables.

The Internet Control Message Protocol group contains input and output statistics. This group has read-only counter objects for maintaining various statistics and error counts for the ICMP protocol. It provides ICMP messages such as destination unreachable, time exceeded, parameter problem, echo request and echo reply.

The Transmission Control Protocol group gathers statistics about the Network Element's TCP connection. The User Datagram Protocol group

contains statistics and information about the Network Element's UDP connection.

The Exterior Gateway Protocol group contains managed objects needed for the EGP protocol. It collects statistical information about the EGP protocol.

The Transmission group contains the network access layer interface types. For instance, it defines the Network Element's transmission over Ethernet, Token bus (IEEE 802.4), Token ring (IEEE 802.5), Serial port (RS-232) connections.

Finally, the Simple Network Management Protocol (SNMP) group contains the objects that are related to the SNMP protocol. It represents a collection of meaningful counters, status conditions, and errors detected.

Figure 2.14 illustrates the ten groups in the MIB II definition in relation to the OSI protocol stack.

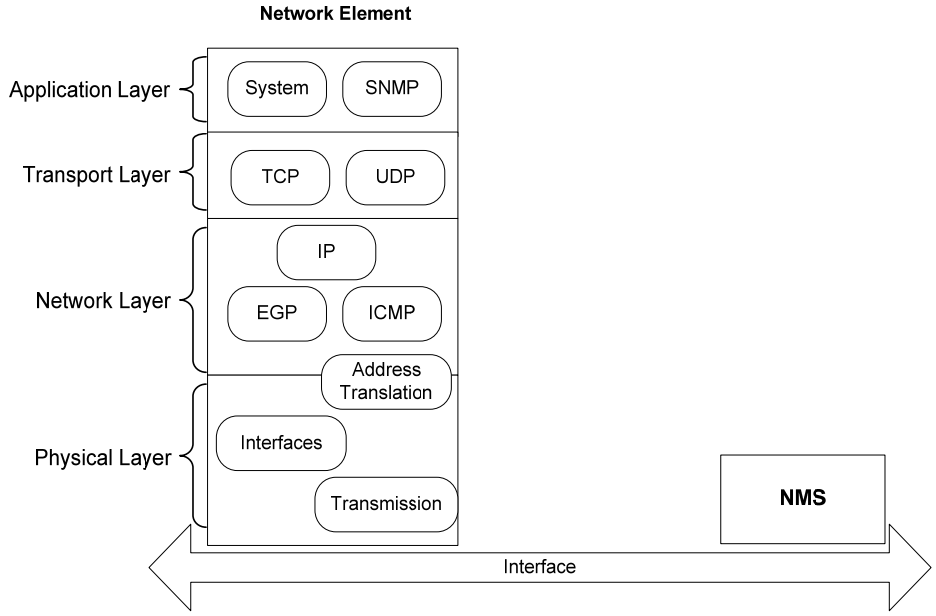


Figure 2.14: MIB groups

2.3.4 IP-Based Network Management: SNMP

The SNMP is the most popular and dominant application-layer protocol used for monitoring and managing network devices in IP-based data communication network [MAUR01]. SNMP was designed in the late 80's to facilitate the exchange of management information between networked devices. The SNMP protocol enables network and system administrators to remotely monitor and configure devices on the network (devices such as switches and routers) and uses the UDP as the transport protocol for passing data between managers and agents. The SNMP specification is contained in RFC 1157, dated May 1990. UDP, defined in RFC 768, was chosen over the Transmission Control Protocol (TCP) because it is a connectionless protocol i.e. no end-to-end connection is made between the agent and the Network Management System (NMS). This aspect of UDP makes it unreliable, since there is no acknowledgment of lost packets at the protocol level. It is up to the SNMP application to determine if packets are lost and need to be retransmitted. However the benefit of using the UDP protocol is that it requires low overhead i.e. less or no impact on the network performance [KAST91].

Due to UDP’s reliability issues, further research has been made in order to overcome UDP’s limitations. RFC 3430 presents an experimental approach for implementing SNMP over TCP protocol [SCHO02]. It proposes that the SNMP provides both TCP and UDP connections at the same time. The selection of transportation protocol can be made according to the size of the SNMP message through default policies. When an SNMP message is larger than a predefined size, the SNMP

manager selects TCP for transporting the message, otherwise it selects UDP. SNMP over TCP offers flow control and efficient segmentation, consequently, management messages over TCP results in a reliable exchange between managers and agents. SNMP over TCP did not receive wide support [MAUR01] due to the extra signalling load and delay incurred in the handshake procedure.

2.3.4.1 SNMP Protocol Structure and Operations

There are three versions of SNMP. These versions are the SNMPv1, SNMPv2 and SNMPv3 (refer to Appendix A for more details and comparisons). SNMPv1 is the standard version of SNMP, the SNMPv2 was created as an update of SNMPv1 and SNMPv3 updated the security issues arise by the previous versions. Figure 2.15 represents the structure of an SNMP message being sent using TCP/IP. The SNMP message is encapsulated, first by the UDP header and then by the IP header.

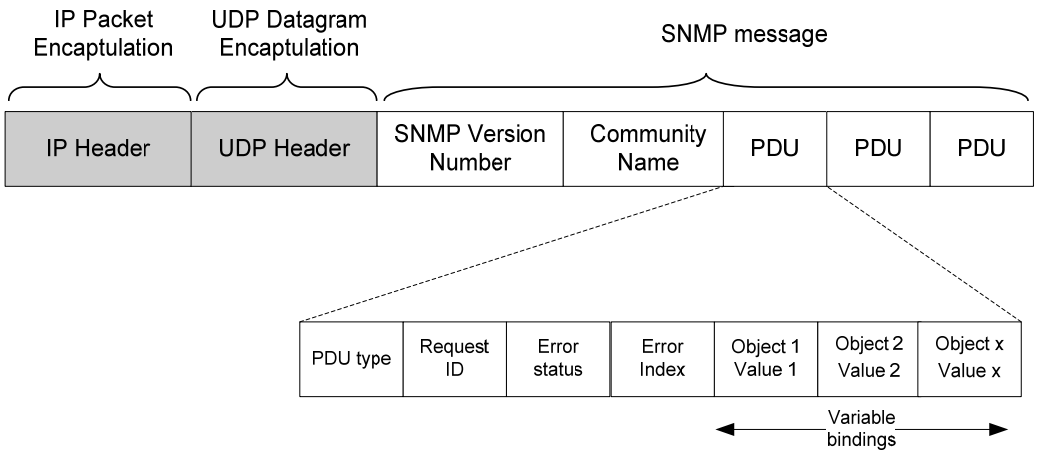


Figure 2.15: SNMP message

The SNMP message consists of three components:

- SNMP Version Number: Indicating the version of the SNMP protocol that is being used (SNMP 1, SNMP 2C, SNMP 3).
- The Community name: If the SNMP version is 1 or 2c then the community name is a simple string value up to 255 bytes. For the SNMP version 3 as presented in the RFC 2572, the community name consists of a number of authorization and authentication fields.
- Data: A sequence of Protocol Data Units (PDUs) associated with the request. PDUs define the type of operations performed by the SNMP manager. For example, GetRequest, SetRequest etc. There can be multiple PDUs in a single message.

Each PDU defines the following fields (figure 2.15):

- PDU type: Identifies the type of the PDU (Get, GetNext, Trap, Inform etc).
- Request ID: Associates SNMP requests with responses.
- Error status: Only the SNMP response message sets this field. The SNMP message request sets this field to zero. This field indicates the number of errors and the type of the error.
- Error index: Only the SNMP response message sets this field. This field associates an error with a particular object instance.
- Variable bindings: This field is served as the data field. It associates a particular object instance with its current value.

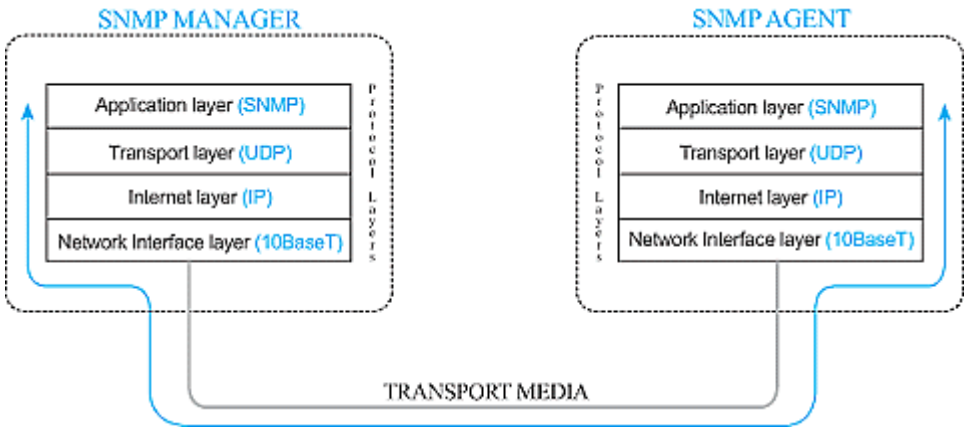


Figure 2.16: TCP/IP communication model and SNMP

Figure 2.16 depicts the SNMP architecture. The data path between the manager application process and the agent application process passes through four layers: UDP, IP, Data Link, and Physical Link on the manager side, and passes through the same layers in reverse on the agent side. When either a manager or an agent needs to perform an SNMP function (e.g. a request or notification), the following events take place in the protocol stack:

Application: First, the actual SNMP application (manager or agent) decides what it is going to do. For instance, it can send an SNMP request to an agent, send a response to an SNMP request (this would be sent from the agent), or send a notification to the manager. The application layer provides services to an end user, such as an operator requesting status information for a port on an Ethernet switch.

UDP: The next layer, UDP, allows two hosts to communicate with one another. The UDP header contains the destination port of the device to

which it is sending the request or a notification. The destination port will either be 161 (query) or 162 (notification).

IP: The IP layer attempts to deliver the SNMP packet to the intended destination, as specified by the IP address.

Medium Access Control (MAC): The final event that takes place for an SNMP packet to reach its destination is for it to be handed off to the physical network, where it can be routed to its final destination. The MAC layer is comprised of the actual hardware and device drivers that put the data onto a physical piece of wire, such as an Ethernet card. In addition, the MAC layer is responsible for receiving packets from the physical network and sending them back to the protocol stack so they can be processed by the application layer.

2.3.4.2 SNMP contribution and influence

The SNMP model and protocol were developed with the design philosophy that the agents are simple and the cost to support network management must be low [AMIR95], [LOPE00] Due to that philosophy, SNMP gained a wide acceptance and is the most widely implemented management framework in the Telecom industry today [LOPE00], [JING09]. With the emergence of NGNs, the networks are expanding fast and the amount of data is increased, resulting in complex heterogeneous networks. In such scenarios, SNMP protocol stack that is simple and has few operational commands, is insufficient and could not provide scalability and efficiency. Scalability refers to the number of agents that can be managed by a single manager system and the efficiency is how quickly and effectively the

network management system will be able to cope when the management data increases. The simplicity of SNMP is not able to cope with the large amount of management information. Management data increases due to the fact that the entire managed system increases with the growth of the network and the quantity of the management data in each agent [KOTS08].

2.3.5 CMISE/CMIP

CMIP is an OSI protocol used with the Common Management Information Services (CMIS) to support information exchange between network management applications and management agents. CMIS defines a system of network management information services. CMIP supplies an interface that provides functions, which can be used to support both ISO (International Standards Organization) and user-defined management protocols.

CMIP used in the in the TMN framework and is mostly implemented for the telecommunication sector by companies such as Ericsson, Nortel and Motorola. It follows the Manager/agent model similar to that of SNMP [WARR89].

2.3.5.1 The CMISE

The Common Management Information Service Element (CMISE) [WARR90]. defines services for accessing management information concerning the network, controlling the network and receiving status reports from the network. Furthermore, it provides commands for

accessing the agent in the network device. These commands have three categories:

- Management Association Services provided by Association Control Service Element (ACSE)
- Management Notification Services
- Management Operation Services.

The Management Association Services (MAS) provides primitives that control the connection establishment with other CMISE managers. These primitives are involved with manager to manager communication. The following table (table 2-1) contains these primitives.

Table 2-1: MAS primitives

MAS Primitives	Description
M-INITIALIZE	Generates connection establishment to peer CMISE users for transferring management information.
M-TERMINATE	Terminates an established connection between peer CMISE service users.
M-ABORT	Terminates the connection between CMISE peers in the case of an abnormal connection termination.

The MAS commands as can been seen from the table above, provides manager to manager communication only to other CMISE enabled managers. CMISE has not been designed to operate with managers that use different communication protocols such as SNMP managers. This limitation can be solved with translation techniques proposed by [NAKA95] where SNMP can be translated into CMIP and vice versa by using a rule description. This technique proposes a protocol conversion and management information translation by using a description rule for

translating the management information and storing the content of the management information into a virtual MIB. The proposed technique could be applied in a network that is managed by only these two management protocols (SNMP and CMIP), but in an environment such as NGN, where the transport stratum is not homogeneous and different management protocols are required to manage the network, this technique is not sufficient.

The Management Notification Services (MNS) are used by the CMIP management agents to inform the managers that some event has occurred or to set events. The M-EVENT-REPORT primitive is used in order to report an event about a managed object to a CMISE manager.

The Management Operation Services (MOS) are operations performed by the CMIP. These operations are listed in the following table (table 2-2).

Table 2-2: MOS primitives

MOS Primitives	Description
M-CREATE	Creates an instance of a managed object in the agent's MIB.
M-DELETE	Deletes an instance of a managed object in the agent's MIB.
M-GET	Requests managed object attributes. The request can handle one object or a set of objects.
M-CANCEL-GET	Cancel previously requested and currently outstanding invocations.
M-SET	Set managed object attributes.
M-ACTION	Request an action to be performed on a managed object

2.3.5.2 CMIP-based Communication

The CMIP is based on Remote Operations Service Elements (ROSE) and Association Control Service Element (ACSE) services. ROSE provides remote interaction using request/response primitives [WARR90]. There are

four ROSE primitives: RO-INVOKE, RO-RESULT, RO-ERROR and RO-REJECT [WARR90]. ROSE enables initiation or execution of operations on remote systems.

ACSE is a sub-layer of the application layer which allows CMISE to set up and terminate connections. In other words, is responsible for establishing and releasing application associations [WARR90]. Figure 2.17 illustrates the Manager/Agent CMIP-based communication.

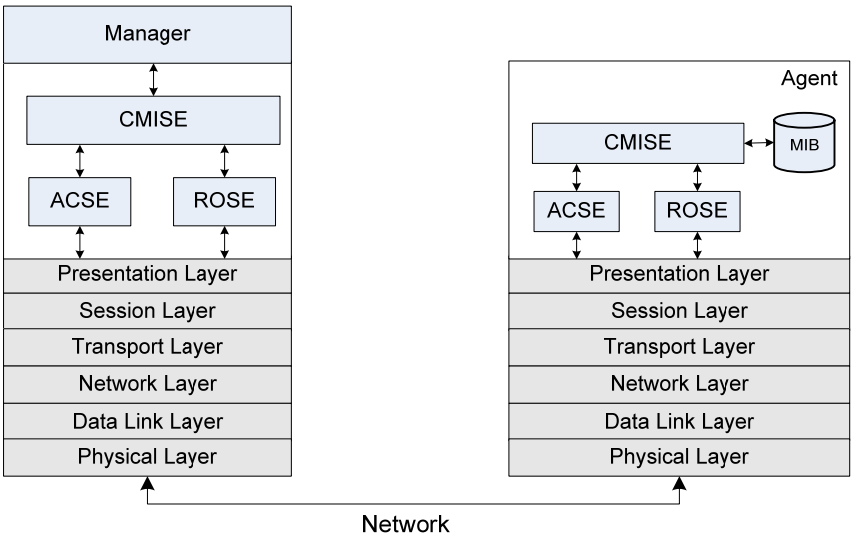


Figure 2.17: Manager/Agent CMIP-based communication

According to figure 2.17, the management function interacts with the CMISE. The management information is exchanged through protocol stacks supported by both manager and agent. The management protocol that provides communication between manager and agent is the CMIP. CMISE uses the ACSE primitives to initiate and establish a connection with the remote agent. When the connection is established, CMISE initializes a MOS primitive to perform a specific function. This primitive is not transmitted directly over the CMIP protocol but it is sent to ROSE

element. ROSE encapsulates the CMISE primitive into a request and sends the message to the ROSE element on the agent. The ROSE element on the agent has to perform the opposite procedure in order to pass the message to the CMISE. Next, the agent's CMISE process the manager's request by accessing the MIB and acquiring the value of the requested management object. The same process is performed by the agent's CMISE in order to send the management information to the manager.

2.3.5.3 Comparing SNMP and CMIP

Table 2-3 illustrates the differences between SNMP and CMIP.

Table 2-3: Comparison of SNMP and CMIP

Feature	SNMP	CMIP
PDU length limitations	484 octets	Unlimited
Interconnection model	Connectionless	Connection-oriented
Interaction method	Polling based	Event based
Information Model	Variable-oriented (attribute)	Object-oriented
MIB language	SNMP SMI	GDMO
ASN.1	Full support	subset
Events/traps	unconfirmed	Confirmed & unconfirmed
Complexity	Agent is simple	Agent is complex
Implementation and maintenance	Simple	Complex and difficult
Management Entity Interactions	Manager/Agent, Manager to Manager	Manager/Agent, Manager to Manager
Robustness	Low due to UDP	High due to TCP
Performance	High for LAN and MAN. Acceptable for networks with limited bandwidth	Low for LAN and MAN. High for networks with limited bandwidth
Scalability	High	Low
Industry acceptance	High	Very Low

SNMP and CMIP are both based on the manager-agent model and both provide manager to manager communication. SNMP has a message length up to 484 octets because it is limited by the connectionless UDP

transportation protocol. The CMIP does not have any message limit due to the use of TCP protocol. This means that CMIP can request more management information per message whereas SNMP has a predefined limit. Therefore, more functions can be accomplished with a single request. On the other hand SNMP does not have to send acknowledgements for every message exchange due to the use of UDP protocol compared to the CMIP. As a result, the SNMP with smaller messages and with a connectionless communication pattern can produce less network overhead compare to CMIP. With SNMP's datagram transmission method, messages can be lost without the SNMP manager receiving notification. CMIP agents are more sophisticated than SNMP in that CMIP provides more powerful primitives that allow management applications to accomplish relatively sophisticated management tasks with a single request. As a result, the CMIP has more comprehensive automatic event notification functions compared to SNMP that uses mostly the polling method due to the simplicity of the agent. The CMIP information model is object-oriented compared to the variable-oriented model that SNMP uses. CMIP uses the Guideline for Definition of Managed Objects (GDMO) for defining managed objects within TMN-based systems. GDMO is a specification that defines a structure description language for specifying objects classes and object behaviours. SNMP uses the SMI for defining the MIB structures of the managed objects. Both GDMO and SMI are based on ASN.1. CMISE/CMIP was designed to be much more powerful and therefore is more complex and resource intensive to implement. Only large systems would be able to

handle a full implementation of CMIP [NAKA95]. SNMP was designed to be more simple and lightweight. CMIP agent increases the overhead on the network elements compare to SNMP agent. Network elements with low memory cannot cope with the resources that CMIP occupies. Therefore, CMIP is a protocol that has not been widely adopted and few vendors support it [INTGR], [OPENVIEW], [SUN96].

2.3.6 Web-Based Enterprise Management (WBEM)

Web-Based Enterprise Management is a set of management and internet standard technologies developed to unify the management of distributed computing environments [CARE02b]. DMTF is the industry organization that leads the development of the WBEM standard. WBEM consists of three standards: the Common Information Model (CIM), Web Based Enterprise Management and an XML binding for the CIM. The relationship between the three standards is illustrated in figure 2.18b.

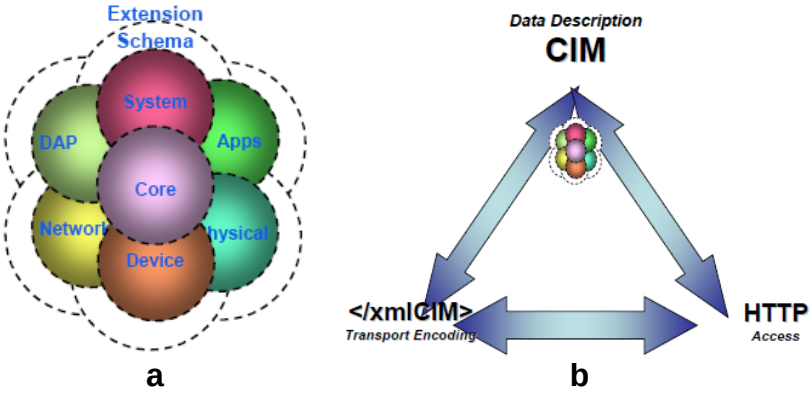


Figure 2.18: (a) Common Information Model, (b) Key DMTF specification

CIM is an information model, a conceptual view of the managed environment. It is specified on Model Object Format (MOF), but is increasingly being represented in UML. CIM attempts to unify and extend the existing instrumentation and management standards such as SNMP and CMIP by using object-oriented constructs and design [WEST00]. The CIM model consists of a specification and a schema. The specification defines the details for integration with other management models. The CIM schema provides the actual model descriptions and captures notions that are applicable to all common areas of management, independent of implementations. The CIM schema is a combination of the core and common models as illustrated in figure 2.18a.

The CIM Core schema is a set of classes, associations and properties that provide a vocabulary in order to describe managed systems. The core schema is a starting point for determining how to extend the common schema. The latter represents information models for particular management areas, but it is independent from any particular technology or implementation i.e. it can be represented in JAVA or C++ or in any other object oriented programming language. Examples of common models include system, networks, applications and devices.

In order to manipulate the management information, WBEM needs an access protocol. Thus, WBEM defines the XML for CIM (xmlCIM) specification, so that messages with content based on the CIM model can be passed in XML documents. The xmlCIM Encoding specification defines XML elements, written in Document Type Definition (DTD), which represent CIM classes and instances. The transport of the management

information is passed over the HTTP protocol. HTTP provides a highly flexible management protocol for exchanging CIM based, XML encoded management information.

DMTF has been accepted by many key industry actors such as Cisco and Microsoft especially due to the CIM model. XML over HTTP, which is offered by WBEM for transportation of management information, has been the key factor for the development of inexpensive management infrastructure. On the other hand, WBEM provides rich but complex set of information models, which lack of explanation of how they can be used, in what types of application and in what way. As a result, management system developers struggle to identify the appropriate information objects for their applications. WBEM does not provide methodological guidance for designing management applications by using CIM as it is considered out of the scope of the standard.

2.4 ITU Next Generation Network Management Framework

2.4.1 The NGN Architecture: Service and Transport Strata

These days, modern telecommunication involving satellites, mobile phone networks such as GSM/GPRS, wireless LAN, WiMax and Bluetooth provide new services like Video on Demand, telephony Voice over IP, Games on Demand and Content Cashing or Video on Demand (VoD) casting etc. [EURO04]. Next Generation Networks will accommodate heterogeneous networks with high level of distribution and complexity.

Figure 2.19 illustrates the NGN environment consisting of multiple technologies.

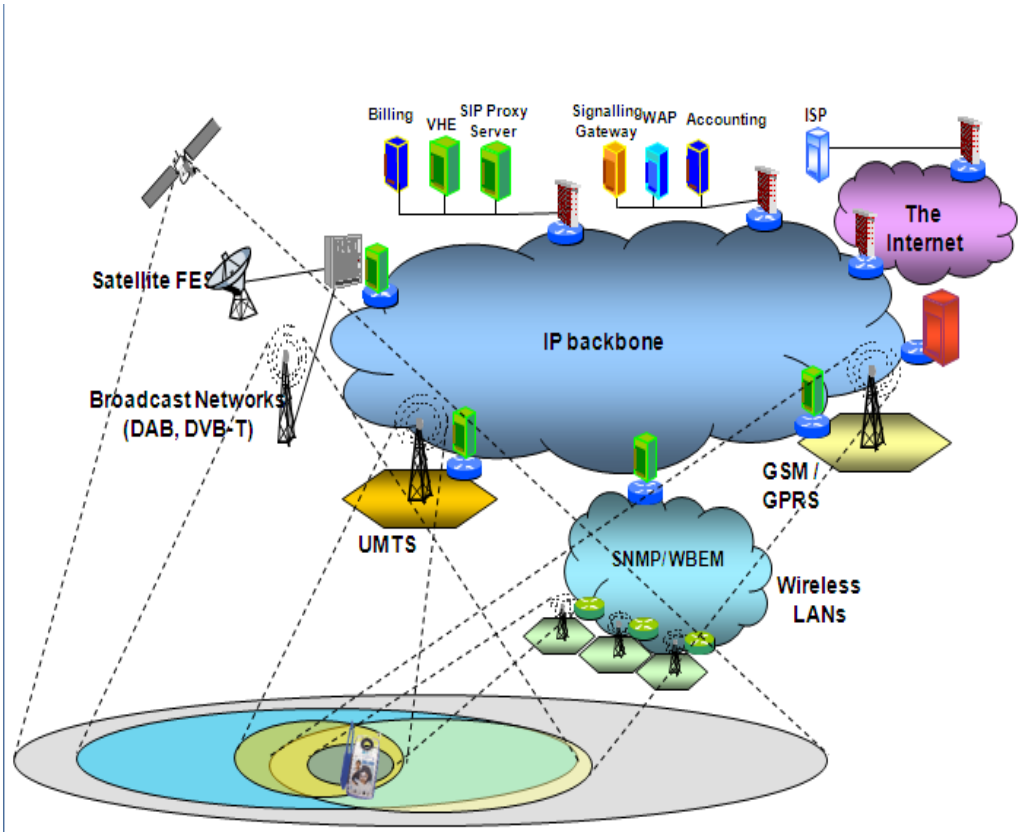


Figure 2.19: Heterogeneous environment of NGN and relation with legacy network

ITU defines the term Next-Generation Network (NGN) in Recommendation Y.2011 [Y.2011] as a packet-based network able to provide telecommunication services and able to make use of multiple broadband, QoS-enabled transport technologies and in which service-related functions are independent from underlying transport-related technologies. It offers unrestricted access for users to different service providers. It supports generalized mobility, which will allow consistent and ubiquitous provision of services to users.

The NGN architecture, as it is recommended by the ITU, is divided into two independent functional stratums, the Service stratum and the Transport stratum (figure 2.20) [M.3060].

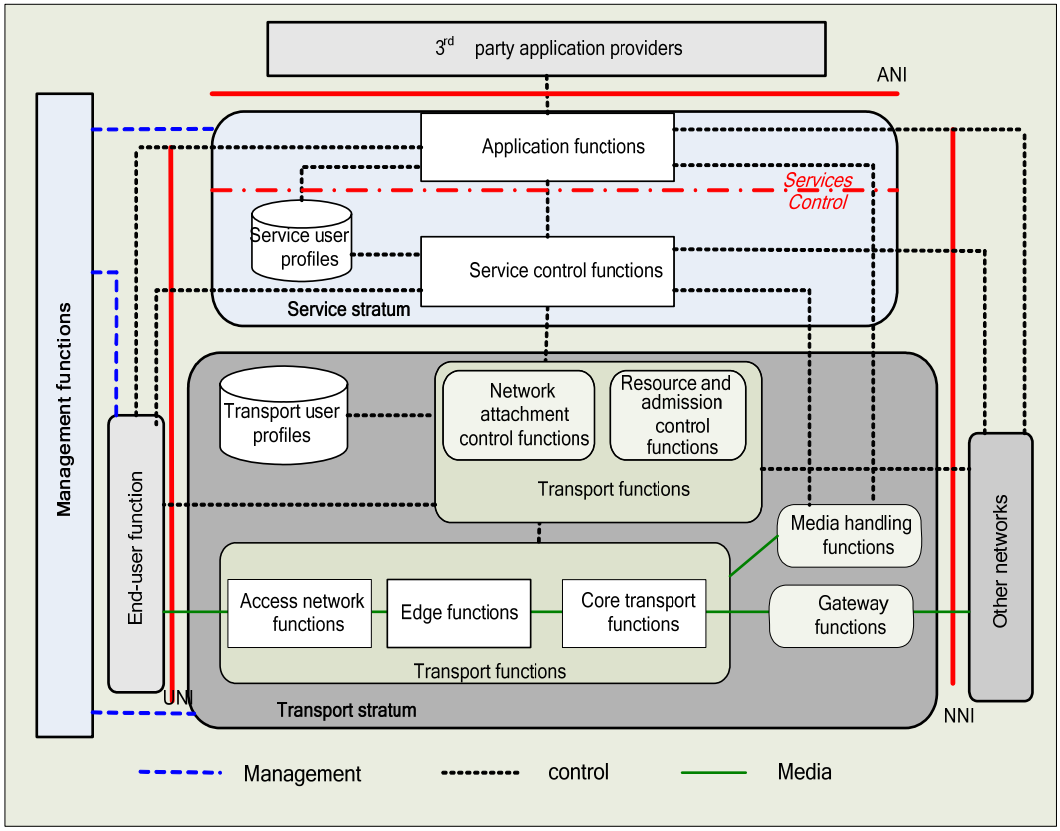


Figure 2.20: NGN architecture [M.3060]

By separating the transport from the service stratum the system provides flexibility in several aspects. One of the benefits is the installation independency. This means that the equipment used on stratum is independent of the equipment that is used on other stratum allowing flexible deployment scenarios to meet the capacity requirements of each component. New services can be deployed to the service stratum (i.e. session-based services and non-session services) while the transport equipment remains unchanged. Another benefit of that separation is the

migration independency. The transport elements can be upgraded or replaced with new technologies without changing service provisioning facilities. A common Transport stratum could be used by different retail sections of the same provider group. This modularity is a unique feature of the NGN architecture [M.3050.1].

The NGN Service stratum provides the functions that control and manage the network services in order to enable end-users services and applications. The services can be voice, data or video applications. In more detail, these functions provide session-based services such as IP telephony, video chatting and videoconferencing and non session-based services such as video streaming and broadcasting. In addition, the service stratum functions provide all of the network functionality associated with existing Public Switched Telephone Network/Integrated Services Digital Network (PSTN/ISDN) services. The Transport stratum provides functions that transfer data between peer entities and functions that control and manage transport resources in order to carry these data among terminating entities. The data could be user, control and/or management information data. In addition, the Transport stratum is responsible to provide end-to-end QoS, which is a desirable feature of the NGN. IP is recognized as the most promising transport technology for NGN. Thus, the IP provides IP connectivity for end-user equipment outside NGN, as well as controllers and enablers that reside on servers inside NGN.

2.4.2 The TMN NGN Management Framework

TMN has been extended to include the management of the architectural evolution of Next Generation Networks. The ITU-T M.3060 recommendation [M.3060] defines the framework for NGN management in terms of four basic architectural views: Business process view, Management functional view, Management Informational view and Management physical view. Each of these views gives a different perspective into the management plane. This management framework consists of functions that give the ability to manage the NGN in order to provide services with expected quality, security and reliability. Figure 2.21 illustrates the four architectural views of the NGN management architecture as well as the security considerations.

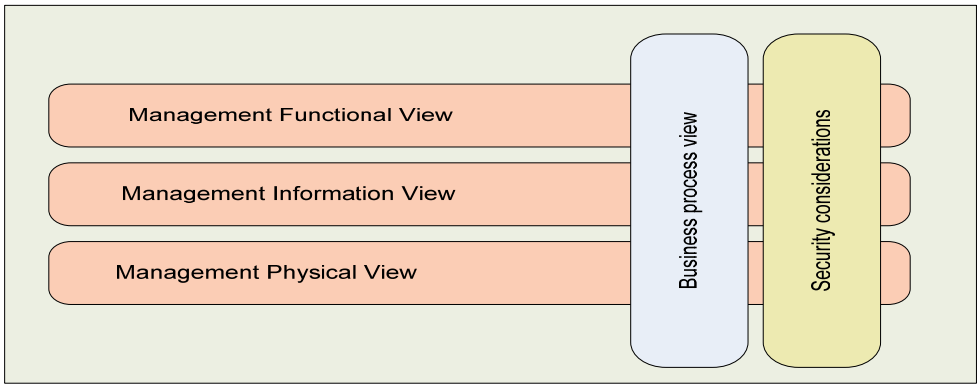


Figure 2.21: NGN management architecture

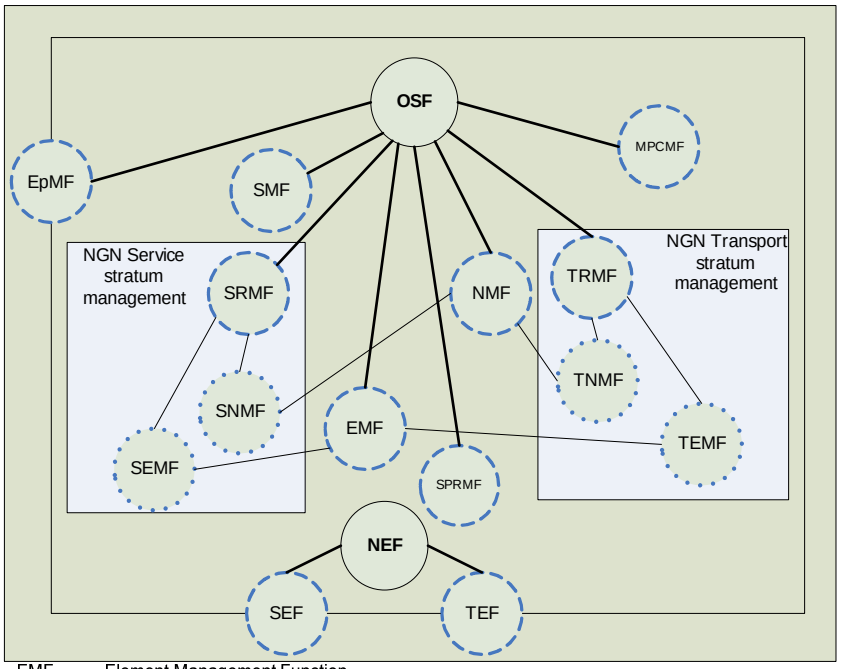
2.4.2.1 Business Process View

The business process view of the NGN is based on the enhanced Telecom Operations Map (eTOM) model which is specified in the ITU-T

recommendation M.3050 series [M.3050.1]. eTOM is examined in detail later in this thesis in section 2.4.3.2.

2.4.2.2 Management Functional View

The functional view of the NGN management is a structural and generic framework of the management functionality. A management function is the smallest part of a business process or management service [M.3060]. Figure 2.22 illustrates the different types of management function blocks. Refer to ITU-T Rec M3060 for a complete description of NGN Management function blocks.



EMF Element Management Function
EpMF Enterprise Management Function
MPCMF Market, Product & Customer Management Function
NEF Network Element Function
NGN Next Generation Network
NMF Network Management Function
OSF Operations Systems Function
SEF Service Element Function
SEMF Service Element Management Function
SMF Service Management Function
SNMF Service Network Management Function
SPRMF Supplier/Partner Relationship Management Function
SRMF Service Resource Management Function
TEF Transport Element Function
TEMF Transport Element Management Function
TNMF Transport Network Management Function
TRMF Transport Resource Management Function

Figure 2.22: NGN Management Block Functions (ITU-T Rec M.3060)

The management of the NGNs is very complex [NARA00] [PANT08]. It is easier to deal with this complexity by dividing the management functionality into layers. The Logical Layer Architecture (LLA) organizes the functions into groups or logical layers. Each logical layer deals with particular aspects of management functions. Figure 2.23 illustrates the Logical Layer Architecture.

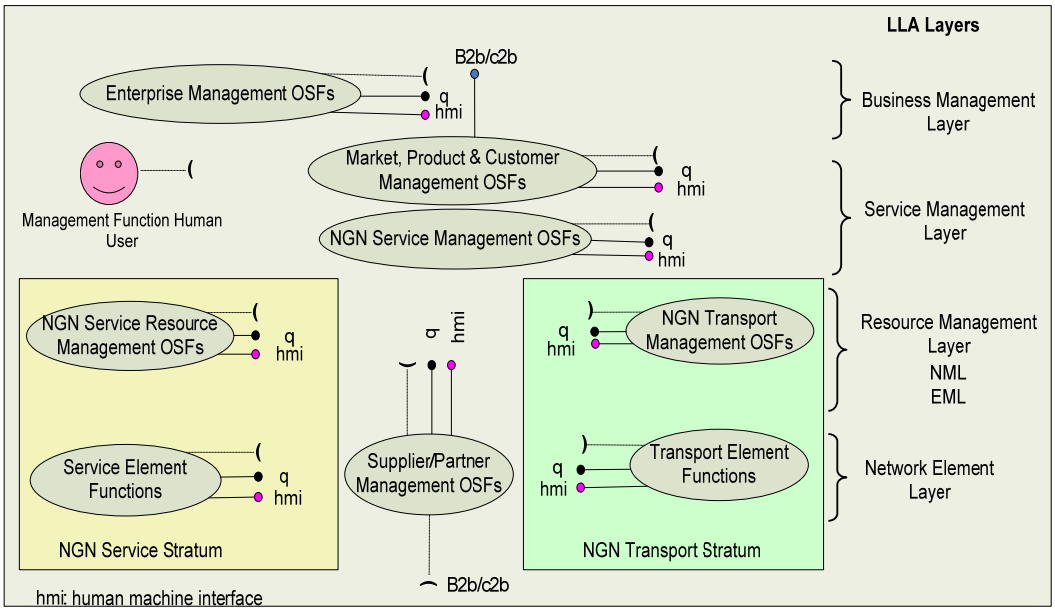


Figure 2.23: NGN management logical layer architecture

The management functionality is categorised into the following groups:

- Enterprise Management: Enterprise Management group is responsible for the basic processes and functions that are required for managing any large business.
- Market, Product and Customer Management: The main purpose of this group is to provide a common functionality for order management of Service Provider's products and to administer and

manage functionality that uses information from the Service Management Layer. In addition, it manages the instances of Product Objects during their whole lifecycle and handles the dialog with customers through a well-defined business interface.

- NGN Service Management: This group is responsible for managing the delivery and assurance of services to end-users according to the customers' expectation.
- Resource Management: This functional group deals with the management of the logical service and transport infrastructures. The Resource Management is divided into Service Resource Management: and Transport Resource Management.
- Service and Transport Element Management: A specialization of Network Element Function representing the telecommunication service and transport functions.
- Supplier and Partner Relationship Management: Deals with the supplier's and partner's communication for importing external transport or service resources that the enterprise will use.

2.4.2.3 Management Informational View

The management of a telecommunications environment is an information processing application. In order to effectively manage complex networks and to support network operator/service provider business processes, it is necessary to exchange management information between management applications which are implemented in multiple managing, and managed systems. Thus, telecommunication management is a distributed

application. The Management Informational view is an object-oriented or service-oriented approach which allows the Open Systems Interconnection management principles to be applied in the NGN context. A network information model is a uniform, consistent and rigorous method for describing the resources in a network, including their attribute types, events, actions and behaviors. The network information model is generic to ensure that a wide range of network resources can be modeled. ITU-T Recommendation M.3100 [M3100] defines a generic network information model for TMN that is based on the OSI management information model [ISO93]. In the OSI information model, the management view of a managed object is described in terms of attributes, operations, behavior and notifications. Attributes are the properties or characteristics of an object, operations are performed upon the object, behavior is exhibited in response to operations and finally, notifications are emitted by the object [ISO93]. The TMN uses the same concepts in describing its information model. The physical resources in the TMN are represented by managed objects, registered on appropriate branches of the object identifier tree. Definitions are inherited from the OSI management information definitions [ISO93].

2.4.2.4 Management Physical View

The management physical view, as defined by the ITU-T M.3060 [M3060], consists of physical blocks and communication interfaces. A physical block is an architectural concept representing a realization of one or more function blocks. Actually, a physical block can be a hardware system, a

software application, or a combination of the two. A communication interface is an architectural concept enabling interoperable interconnection at reference points between physical blocks by realizing the reference points.

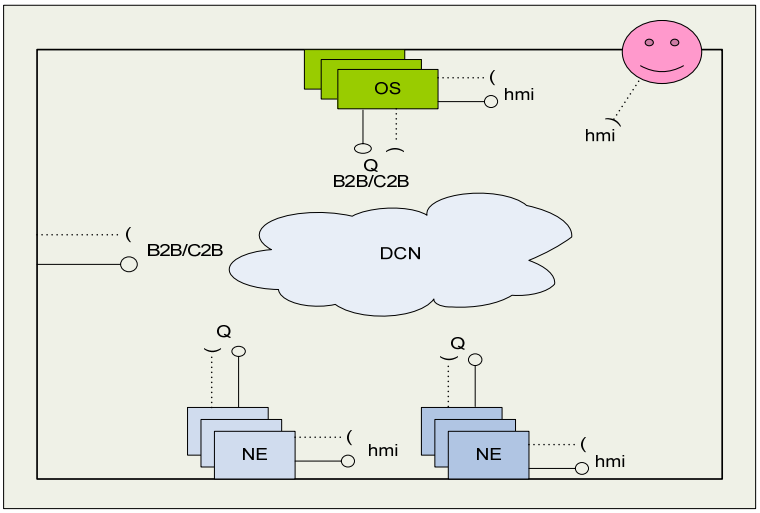


Figure 2.24: NGN management physical view

Figure 2.24 above illustrates a simplified management physical view proposed by the TNM specification [M.3060]. The physical blocks in the management physical view contain the Operations Systems (OS), the Network Elements (NE) and the Data Communication Network (DCN). The OS is a system that performs OSFs. The NE consists of telecommunication equipment and support equipment or any item or groups of items considered to belong to the telecommunications environment that performs NEFs. The DCN is a support service that provides the capability to establish paths for information flow among physical blocks in a management environment.

The DCN may consist of a number of individual sub-networks of different types, connected together. The communication interfaces are: Q interfaces, B2B/C2B interfaces and the Human Machine Interface (HMI) interfaces. The Q interface is characterized by that portion of the information model shared between the OS and those management elements to which it directly interfaces. The B2B/C2B interface is used to interconnect two administrative domains or to interconnect a compliant environment with other networks or systems. Finally, the HMI is an interface applied at HMI reference point, which is exposed for consumption by the users [M3060]. TMN proposes the use of an adaptor to act as a gateway among legacy network equipment and TMN-based Operations Systems. Most of legacy network equipments understand ASCII-based information that is not TMN-conformant operation. The Q adaptor proposed by TMN provides programmatic interface to a legacy equipment to adapt the legacy information to TMN compatible information.

2.4.2.5 Security Consideration

Security has the mission to protect important business assets against different types of threats. Assets can be of different types such as buildings, employees, machines, information, etc. NGN Management is specifically concerned with the management of security aspects of the NGN and with the security of the NGN Management infrastructure. ITU-T Recommendations X.805 and M.3016.x series are considered for securing the NGN management infrastructure. ITU-T X.805 recommendation [X.805] defines concepts and components intended to provide reusable

countermeasures across multiple layers of the infrastructure, including transport and service stratum. The M.3016.x series [M.3016] focuses on end-to-end security, both when management traffic is separate from user traffic and when they are mixed together. To overcome the complexity of securing the NGN infrastructure, including its management plane, there is a need to automate the application of various security services, mechanisms, and tools by using operation systems to automate the process.

2.4.3 The TMF NGN Management Framework

2.4.3.1 The Next Generation Operations Systems and Software (NGOSS)

The TeleManagement Forum (TMF) is a non-profit global consortium that provides strategic guidance and practical solutions for the telecommunication management and the development of management systems and standards. It was established in 1988 as the OSI/Network Management Forum under the sponsorship of the ITU [TMF]. Later the name was changed to TeleManagement Forum. The strategic goal of the TMF is to identify and create standard interfaces that allow a network to be managed consistently across various network element suppliers. In the TMF, a next-generation solutions framework, the NGOSS has been developed to enable general use reuse of carrier and vendor expertise on processing, information models, systems integration methods, application components in constructing operations and business support systems (OSS/BSS) [SASA09].

NGOSS aims to deliver a framework that will help produce New Generation OSS/BSS solutions. The goal of NGOSS is to provide a rapid development of flexible, low cost ownership of OSS/BSS solutions in order to meet the business needs of the Telecom industry [EURO06]. NGOSS promotes the use of open-standard commercial off-the-shelf information technologies, instead of proprietary telecommunication technologies. The use of this approach reduces significantly the cost and improves software reuse and operational flexibility. More specifically, NGOSS provides specifications that expose the functionality contained in a NGOSS component. A component is a software entity that is independently deployable and uses contracts in order to expose its functionality. The contracts are structured into four parts. The first part is the functional part that describes the capabilities provided by the component. The second part is the management part, which describes the management requirements needed to operate the functional capabilities. Next, is the non-functional part that defines aspects needed to provide proper operation of the capabilities (e.g. costs, security etc). The last part of the contract structure is the model part that contains various types of Unified Modelling Language (UML) models which describe the functional and non-functional aspects of the contract. NGOSS comprises a number of technological elements as shown in Figure 2.25. Among them, SID and eTOM are the most influential one.

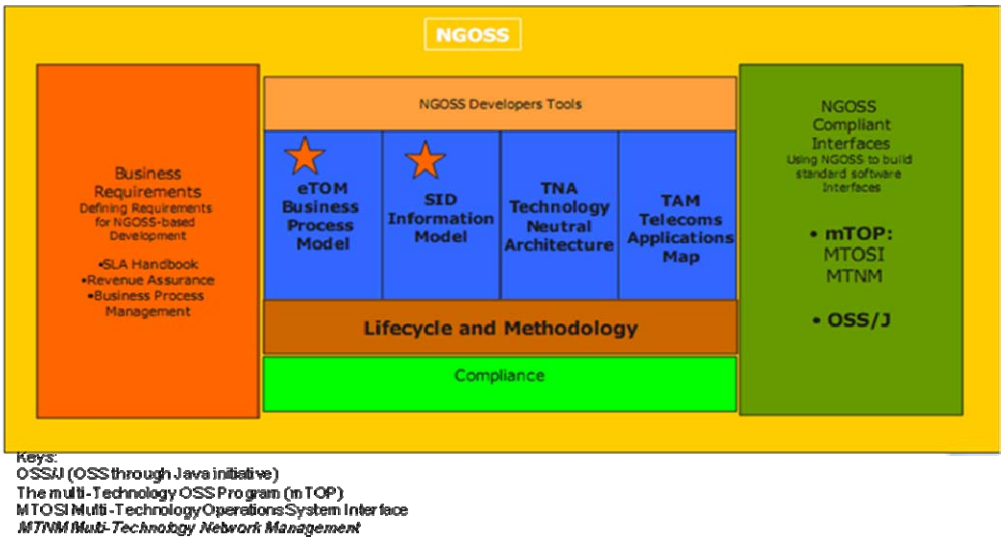


Figure 2.25: overview of an NGOSS Framework

2.4.3.2 The Enhanced Telecommunication Operation Map (eTOM)

eTOM is a reference framework that categorizes the business processes that a service provider will use. More specifically, it is a Business Process Model (BPM) that attempts to map out the high-level telecom business processes. This framework is presented as a hierarchical (top-down) approach to modelling business processes. The business processes are organized as multi-level matrix with horizontal (functional) and vertical (flat-through) process groupings. Figure 2.26 illustrates this matrix with the horizontal and vertical process areas. At the top level (Level 0 Processes), eTOM identifies three vertical processes: (i) Strategy, Infrastructure and Product, (ii) Operations, and (iii) Enterprise Management. Furthermore, in this framework four horizontal process areas are identified: (i) Marketing, Product and Customer Processes; (ii) Service Processes involved in developing and managing services; (iii) Resource Processes for managing

network and IT resources; and (iv) Supplier/Partner Processes for managing the interaction with the suppliers and partners.

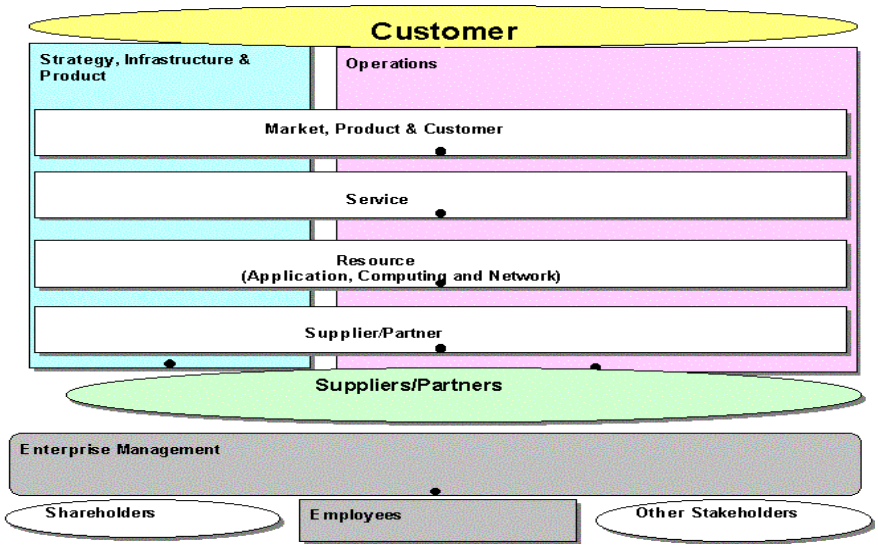


Figure 2.26: eTOM business process (level 0)

eTOM further divides the processes in each of these areas as shown in figure 2.27. The Strategy, Infrastructure and Product Process is separated into vertical processes such as Strategy and Commit, Infrastructure Lifecycle Management and Product Lifecycle Management. These vertical processes are further divided into horizontal processes that are related to Marketing and offer Management, Service Development and Management, Resource Development and Management, and Supply Chain Development and Management.

Most of the TMF work is focused on the Operations Processes. Operations include processes that support customers, network operations, and management. The Operations Processes are divided into three vertical processes: Fulfilment, Assurance and Billing. Fulfilment is responsible for

delivering products and services to the customer. This includes service configuration and activation, order handling and resource provisioning. Assurance consists of proactive and reactive maintenance activities, service monitoring, resource status, performance monitoring and troubleshooting. It includes activities in order to proactively detect possible failures, and to collect performance data in order to identify and resolve potential problems. Billing processes are responsible for collecting usage data records (accounting), various rating functions and billing operations. This includes production of timely and accurate bills, providing pre-bill use information and billing to customers, processing their payments and performing payment collections.

The Enterprise Management level is composed by processes related to Strategic and Enterprise Planning, Enterprise Risk Management, Enterprise Effectiveness Management, Knowledge and Research Management, Financial and Asset Management, Stakeholder and External Relations Management and Human Resource Management.

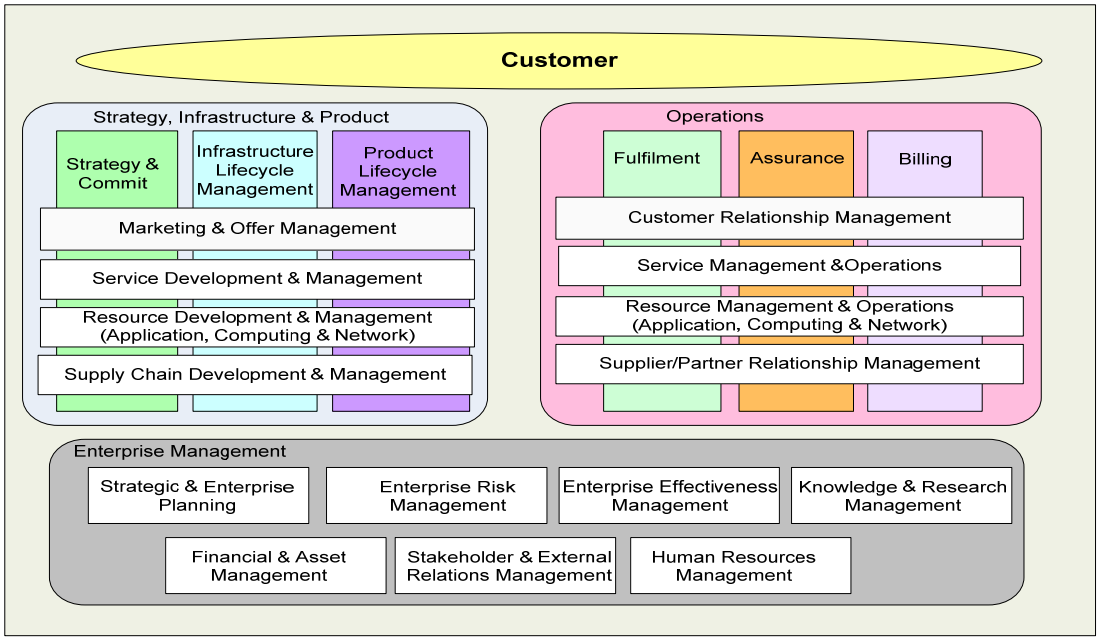


Figure 2.27: eTOM business process framework

The eTOM framework provides many benefits to the service providers. One major advantage is that it can provide better integrated business interactions between service provider and their customers, as well as other service providers and network operators. eTOM is used as a guiding reference for the service providers in designing and dividing business processes and does not intend to be prescriptive as how a service provider is organized or how tasks are carried out.

2.4.3.3 Shared Information Data (SID) Model

The SID model is the NGOSS information model that represents business concepts as well as their characteristics and relationships. Figure 2.28 illustrates these relationships. The description of this concept is described in an implementation independent manner. SID defines semantics and behaviour of the managed entities as well as the interactions among them.

Furthermore, it provides a standard representation by using standard types that describe domain information (e.g. customers, network service, orders and configuration definitions) in an NGOSS system. SID and eTOM collaborate to illustrate how the business process works to contribute to the enterprises as a commonly accepted standard.

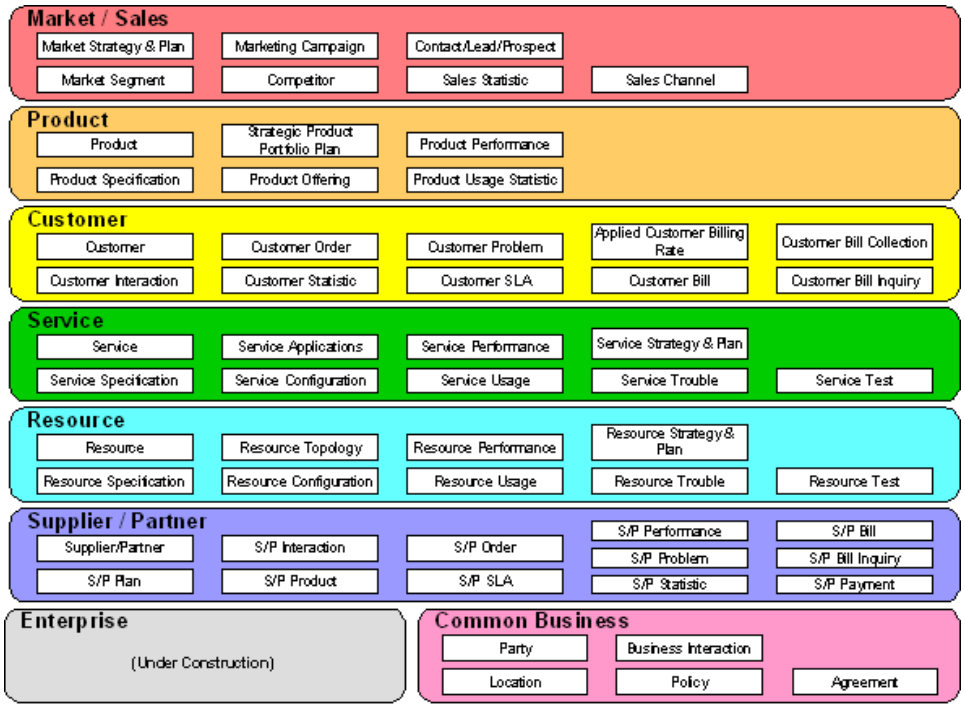


Figure 2.28: SID business entity framework [M.3190]

The SID acts as a repository for all the business and technical information used by a Telecom service provider. Examples might include sales and marketing information, contractual information involving SLAs and their performance histories, customer contract details, customer billing data and payments details, and network and computer equipment inventories [TMF]. The SID business model is not intended to act as a centralized repository. Instead, it is a distributed entity, with component portions

residing in a wide range of repositories, which could be spread all over a wide geographic area. Many of these repositories could include industry standardized databases and legacy applications. These repositories can be sourced from different suppliers and use different data access methods. In order to deal with the compatibility issue, the SID model ensures that all of this information is made available to other applications in a consistent manner, irrespective where or how the original data is stored [EURO06].

2.4.3.4 TMF's Architecture Contribution and Influence

TMF is well accepted by the Telecom industry as a starting point for the development of management systems. eTOM reflects the importance of Internet-style service delivery and Business-to-Business co-operation. The NGOSS provides designs and models that are implementation technology neutral. These models use UML approach for the implementation. SID offers the information language which can be used within the system level views of the NGOSS as well as within the eTOM processes. eTOM especially has influenced the development of telecommunication applications in the area of Fulfilment, Assurance and Billing. However, due to the level of abstraction of the eTOM processes is very high, further decomposition of those processes create very complex process descriptions. As a result, commercial design and implementation restrictions could be applied. eTOM provides a starting point for management development and does not provide a set of processes ready for implementation. Another limitation of the eTOM framework is that it

does not provide a methodology for how to develop and further refine the process models. Finally, the TMF models are difficult to use within heterogeneous environments; for example, environments where other standards are not conformed to TMF standards.

2.5 Conclusion

The management plane is one of the most vital parts of the telecommunication infrastructure as it provides the necessary functions to monitor control and configure different services. This chapter identified the business drivers for the telecommunication management community. Furthermore, this chapter presented an analysis of the standardization bodies which define key management functionalities and architectures that have influenced the design of the telecommunication management systems. More specifically, it has examined the TMN, TINA, CMIP, TMN's NGN Management, TM Forum's NGOSS, and IP-based network management protocols: SNMP and WBEM. All of them play important role throughout the spectrum of the management plane. In addition, this chapter pinpointed those architecture's contributions and influences in the design of management systems. As telecommunication management is being shifted towards software engineering to integrate distributed real-time functions such as authentication, bandwidth management, the software architecture and technologies in the management plane have great influence in defining the management architecture. These will be studied in the next chapter.

Chapter 3 : NGN MANAGEMENT PLANE TECHNOLOGY ANALYSIS

3.1 Introduction

The emerging need for converged services and the rapid expansion of the multimedia and digital traffic are driving the need for networks that are packet-based and able to provide all kind of services in any place, at any time, and on any device. NGNs will consist of heterogeneous networks having high level distribution and complexity. As a result, the management plane needs to be able to deal with this complexity and to successfully manage the network operation as well as the digital data services. The management plane is involved not only with the operations of facilities and services, and business relationships with customers, partners and suppliers but also captures the behind-the-scenes operations that are required to enable services to be delivered.

The NGN management plane handles both OSS and BSS functions. The OSS provides a set of processes that a network operator requires for monitoring, controlling and analyzing the network. Moreover, the OSS includes processes that are required to manage and control faults, and perform functions that enable interactions with customers. Operations Support includes the term network management which means to control and manage the network elements.

BSS provides processes that a service provider requires to conduct relationships with external stakeholders including customers, partners and

suppliers with SLA that is a part of a service contract where the level of service is formally defined. The boundary between Operations Support and Business Support is indistinct as shown in figure 3.1. Business Support functions are the customer-oriented subset of Operations Support. Business Support processes takes an order from a customer for a new service and then this order must flow into the Operations Support processes in order to configure the resources necessary to deliver the service [EURO04]. In other words, the management plane of NGN has to manage all network equipments as well as customer services.

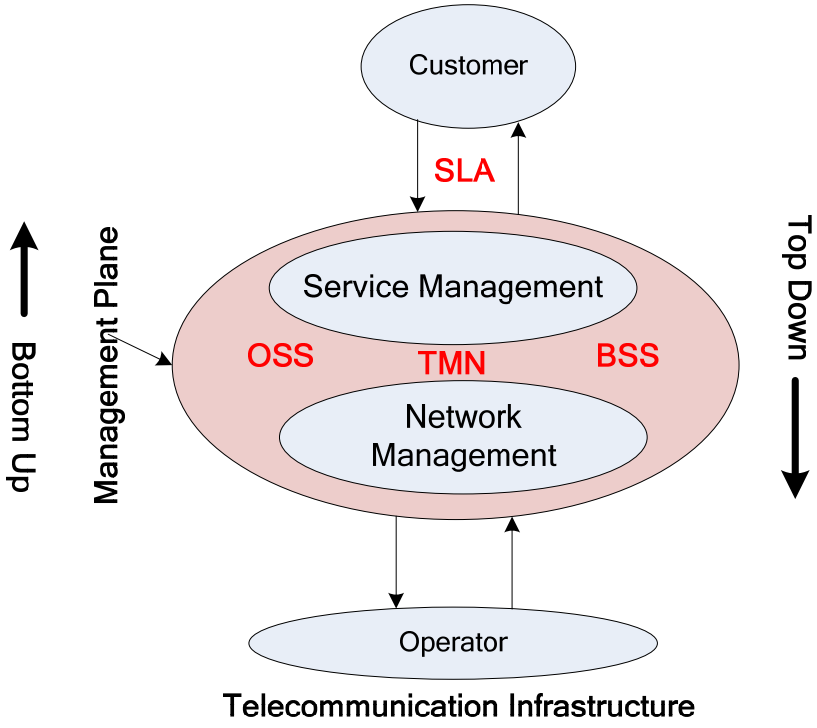


Figure 3.1: The Management plane: Operations Support and Business Support

While Chapter 2 examined the evolution of telecommunication management frameworks, the current chapter is focused on distributed technologies that have been adopted by the telecom operators and

network/service providers in order to provide integration solutions for the telecommunication management architecture. More specifically, this chapter examines the current technologies that are deployed in the transformation of traditional management networks to all IP-based NGNs. First it discusses the limitations of these distributed technologies that will not fully meet NGN's requirements. Second, it introduces the concept of SOA with the main focus on the features that allow telecommunication networks to 'open up' for collaboration. Third, it illustrates the change of architectural styles of telecom industries towards the adoption of SOA that supports business agility and adaptability. Then, this chapter examines the technologies that will enable the SOA design and development. The chapter concludes by proposing a design of SOA-based Network Management architecture for managing NGN.

3.2 The NGN Management Architecture

3.2.1 The Evolving Management Architectures

Network management has evolved from a simple manager-agent model to complex OSS and BSS systems. The objectives and nature of management systems have changed during this evolution. Figure 3.2 illustrates the four stages in the evolution of the management plane, OSS/BSS [IEC02].

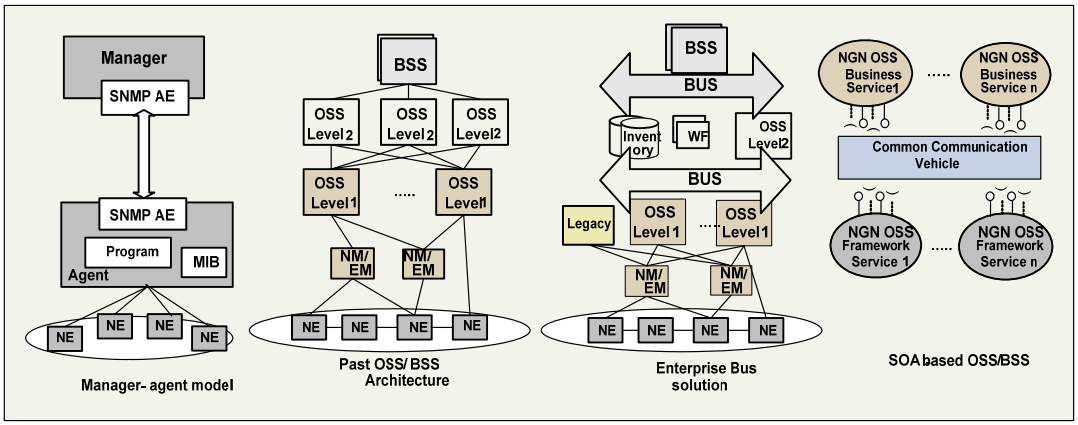


Figure 3.2: Stages of OSS/BSS evolution

3.2.1.1 First Stage: The Manager-Agent Approach

At the first stage of the OSS/BSS evolution, the OSI and IETF network management models utilize a simple manager-agent model, together with protocol-based communication between manager and managing entity (agent). The SNMP [STAL98] is certainly the most widespread use of network management solution that is used by the most of the industry since 1990 and is the first stage of the OSS/BSS evolution. The manager manipulates the management information through the MIB which exists in the network entity. The manager-agent model has a tightly coupled architecture where the manager is dependent on the agents as explained in Chapter 2. The disadvantages are as follows [ZHAN06], [KREG05]:

- Due to the lack of cooperation between NMSs it is hard to implement advanced management functions.
- Multiple management interfaces bring heavy burden and apply more complexity to different network management systems.
- The integration of the NMSs that fulfil the constantly evolving business requirements is difficult to implement.

3.2.1.2 Second Stage: The OSS/BSS Point-to-Point Architecture

At the second stage, operators are required to manage sub-systems within their networks such as the SDH transmission systems, a set of TDM switches or a SS7 network. The management systems are focused on elements and how they function as a system. For example, the SDH standards developed an architecture and information models that can represent end-to-end connections and their components [G.774.05]. With a network-wide management view, the services offered on the network require management. Hence, managing systems require extra layers.

Different levels of OSS are introduced in order to provide extra functionalities into the management domain. This implementation added more complexity into the management architecture. Due to the fact that OSSs were tight-coupled with each other any change to the network architecture could result in configuration problems. This architecture follows a point-to-point integration model. Point-to-point integration uses proprietary messages and custom APIs in order to connect software components and the operational policies are embedded in the application. This architectural style does not provide flexibility and scalability that is required in large scale distributed environments such as telecommunications and often lacks of agility which results in an expensive implementation [BEHA10].

3.2.1.3 Third Stage: A Distributed Approach with The Enterprise Bus Solution

The OSS/BSS architecture adopted today by most operators is the data bus OSS/BSS, which is the third stage of the management evolution. This architecture differs from the previous in the sense that it includes a middleware between the layers of the OSS levels and between the OSS and BSS. By adopting middleware technology such as the Distributed Object Technology (DOT) including CORBA, DCOM, RMI, the communication between OSS layer and BSS as well as among OSS is provided through messages. Appendix B provides a comprehensive description and comparisons on these technologies.

The middleware concept involves the passing of data between applications using a communication channel that carries self-contained units of information. Thus, the architecture becomes more loosely-coupled and support integrated management capabilities [EMME00]. The functionality is modular and higher level processes orchestrate its use. The system has become large and inherently distributed and proper distribution exists. This implementation usually uses workflow engines in order to orchestrate the different components and make them work as one large scale application. The limitation of this approach is that this architecture does not provide interoperability between heterogeneous platforms. Large scale architectures that accommodate different systems usually require different platforms. In order to integrate different platforms adaptation is required, which makes the architecture more complex and

less loosely-coupled. The functionality of this architecture cannot be reused to a high degree due to the adaptations.

3.2.1.4 Fourth Stage: A Distributed Approach with SOA and ESB

In the previous architectures that are based on DOT, every OSS component follows a different design pattern that requires integration and mapping of functionality between components. The time to integrate any solution increases exponentially with the number of systems because each component's interfaces have to be considered separately and not as a part of a pre-integrated framework.

This thesis proposes the adoption of a service-oriented approach based on the SOA philosophy, where services are independent resources and their implementation details are hidden behind the service interface, as the fourth evolution stage and using the Enterprise Service Bus (ESB) as the enabling middleware technology for implementing SOA.

Table 3-1 presents the differences between the DOT approach and the service oriented approach.

Table 3-1: Differences between Distributed Architectures and Service oriented Architectures

DOT-based Approach	SOA-based Approach
Function Oriented	Business Process Oriented
Designed to Last	Designed to Change
Cost Centered	Business Centered
Application Block	Service Orientations
Tight Coupling	Loose Coupling
Homogeneous Technology	Heterogeneous Technology
Object Oriented	Message Oriented

The architecture in this stage of the management evolution consists of well defined loosely-coupled services that use standardized interfaces in order

to provide flexibility and scalability. Loosely-coupled services allow the architecture to achieve faster integration cycle and by making use of standardized interfaces, the management architecture can be more scalable due to the “plug-in” connection approach [ERL05]. Services are connected to the Enterprise Service Bus in a loosely coupled fashion [CHAP04]. This architecture is more agile and can provide more automation functions.

In Table 3-2 the comparison of the tightly coupled systems with loosely coupled systems is shown.

Table 3-2: Tight coupling versus Loose coupling

	Tight coupling	**Loose coupling**
Physical connections	Point-to-point	Via a mediator
Communication style	Synchronous	Asynchronous
Data model	Common complex types	Simple common types only
Interaction pattern	Navigate through complex object trees	Data-centric, self contained messages
Control of process logic	Central control	Distributed control
Binding	Statically	Dynamically
Platform	Strong platform dependencies	Platform independent

3.3 SOA in Telecommunications Network Management

3.3.1 An Overview of Telecommunication Network

The efforts to realize the idea of a service-based Telecommunication network can be dated back as early as 1980’s with the standardization of Intelligent Networks (IN) [SIDH00]. IN is an architectural approach that

custom service logic can be created by service provider for enhanced features on calls in the PSTN networks. IN development had made the telecom network a programmable environment to deliver new value-added services to generate revenue. IN introduced a set of functional entities consisting of distributed functions that are required to interact during call originations and call terminations in the provision of IN call related services. These functions decouple the service development from the network infrastructure. From that onwards, the IN infrastructure has evolved to support some new features and requirements of the evolving networks. One example of this new feature introduced in the Telecommunication networks is the use of CAMEL technology that has been used in the GSM networks to enable services such as roaming and international pre-paid calls. The CAMEL technology is based on the IN [ETSI]. The IN has defined an overlay service architecture on top of a physical network and extract the service intelligence from the legacy network switches into dedicated central control points.

However, IN is not able to fulfil some of the requirements that new converged networks impose such as shorter time to market new services and network independent services. IN and CAMEL services have become popular over the last two decades, but they did not develop into the open market services originally envisioned. The IN program-base was too limited, since it was too program-specific and did not follow mainstream programming paradigms (such as C++ and Java). Furthermore, the telecom world was still a closed and monopolistic environment with no major competition. Nevertheless, global deregulation and the acceptance

of mobile communication and internet have forced the market to become a competitive environment, leading to the need for more innovative architectural paradigms and frameworks for service platforms. RPC and functional programming enabled the IN vision to move towards to object orientation. Programming languages such as C++ and Java enabled the creation of middleware concepts that allowed the implementation of distributed and scalable service delivery platforms and provided abstraction from the details of the underlying network signalling and transport protocols [VENI00].

Initiatives such as the Open Services Architecture (OSA)/Parlay, Open Mobile Alliance (OMA) or Java APIs for Integrated Networks (JAIN) aimed to make telecom service implementations easier than with traditional IN. These architectures were based on object-oriented and distributed middleware technologies such as CORBA and RMI which in combination with C++ and Java provided the basis for flexible service implementations. They achieved flexibility by abstracting from the signaling protocol details of the underlying telecom networks, such as ISDN User Protocol or Session Initiation Protocol (SIP). These protocols with specific APIs featured telecom-related capabilities such as call control, messaging, conferencing, location, and charging [MAGE03].

JAIN [ORACLE] defines a component model for structuring application logic of communications applications as a collection of reusable object-oriented components. These components form a ‘pool’ of reusable functions that is composed of other higher-level components. The higher-level components are able to create new services that are richer in

functional capabilities and need shorter time to market. The JAIN specification also defines the contract between these components and the container that will host these components at runtime. Furthermore, JAIN execution environment provides support for asynchronous applications supported by event models. Application components receive events from event channels that established at runtime. Network resource adapters create representations of calls and pass events generated by the calls to the JAIN execution environment. Application components are in turn invoked by the JAIN execution environment to process these events in a transactional context [JCP].

OSA/Parlay is a joint effort between 3GPP, ETSI and Parlay Group [OSA/PARLAY]. Parlay is an open API for application access to telecom network resources. This technology integrates telecom network capabilities with IT applications via secure, measured and billable interfaces. The Parlay APIs are network independent, and applications can be hosted within the telecom network operator's environment.

Although Parlay and JAIN were promising frameworks, market acceptance was slow, since most network operators did not 'open up' their networks to third parties. Moreover, these frameworks produced complex APIs for non telecom experts, and object-orientation was not fully accepted by the telecom engineers [MAGE07], [KNUT05].

The Parlay Group in 2000 developed a simplified version of the OSA/Parlay APIs called Parlay X [PARLAY4]. Parlay X is based on the emergence of Web Services technology centred on XML. The Parlay X APIs can be used in conjunction with the OSA/Parlay APIs via gateways

or can be used as an independent API. Parlay recognized that IT was creating its own open services market, resulting development of many innovative services due to the use of mainstream internet programming technologies such as Web Services that form the basis of Service-orientation. Thus, the concept behind Parlay X emerged from the use of the internet programming paradigms that they were successfully creating new market shares. Network operators are providing Web Services to let customers make and receive telephone calls, send and receive instant messages, multimedia, charge specific transactions onto telecom bill, etc.

Within the mobile domain, OMA is a standardization body that develops open standards for the mobile phone industry [OMA]. OMA is not focusing on delivering platform implementations, but it provides specifications of service enabling functionalities such as instant messaging, location, presence information, transactions etc. It is left to vendors to implement the platforms and functionalities that OMA describes. OMA provides specifications that are also based on the Web Service technologies [OMA]. Recently OMA focused on initiating actions for standardizing IP Multimedia Subsystems applications. Inspired by the Parlay Group, OMA developed the OMA Service Environment, which allows the creation of applications that are aligned to the SOA principles.

Recently, OASIS Telecom [OASIS08b] was created in order to bring the full advantages of SOA to Telecommunication industry. The OASIS consortium drives the development and adoption of e-business and Web Service standards boosting the convergence of Telecommunications networks and SOA.

Today, with the Internet's success at providing multimedia communication services such as e-mail, VoIP, instant messaging and videoconferencing, the telecom industry is pressurised to implement an open service market based on an open set of enabling services and service components. Web 2.0 and mashup applications are the latest success of internet's service platforms. Web 2.0 and mashup concepts are based on the user-centric platforms [CAET07]. User-centricity refers to the approach that is built around the needs and requirements of the end-user. User-centric platforms extend this approach to allow end users to create their own User-Generated Contents [OECD07]. In other words, Web 2.0 and mashup innovative idea is built around the concept that a client can become a service provider.

Figure 3.3 illustrates the approaches of SOA within the telecommunication industry. This figure shows how the telecom industry evolved from Intelligent Networks to Service-based frameworks

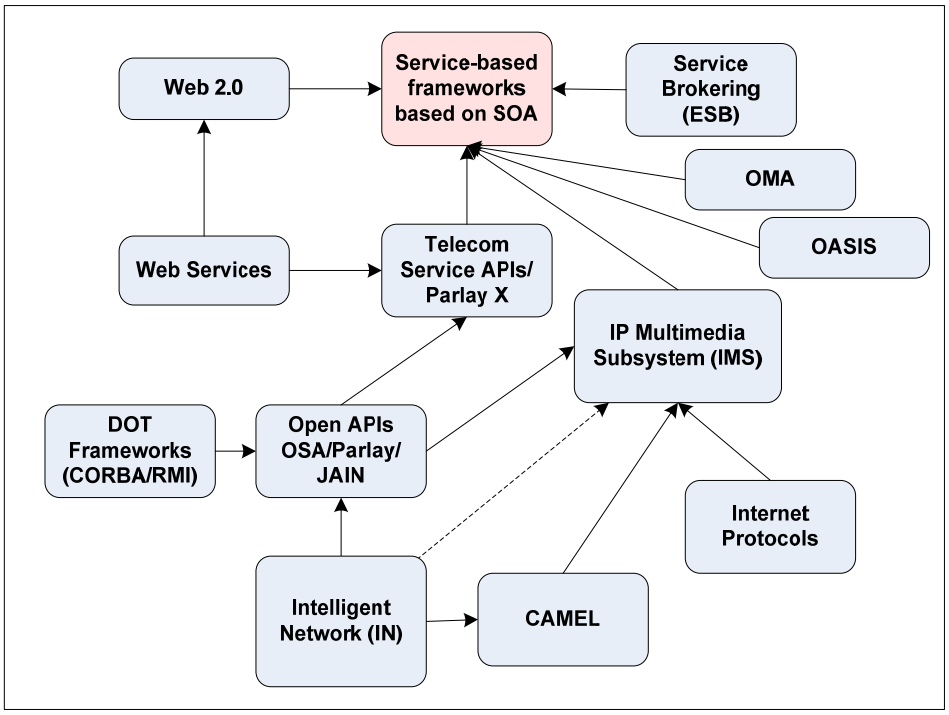


Figure 3.3: Intelligent Networks towards SOA

3.3.2 IP Multimedia Subsystem (IMS) and the Service Delivery Platform (SDP)

Combining Intelligent Network (IN) concepts and exploiting Internet Protocols for session control has led to definition of the IMS architecture. IMS architecture introduced by 3GPP, defines a service provision architecture that can be seen as the Service Delivery Platform for NGN [CHAE05]. IMS is now considered as the global standard for a unified service control platform for converging fixed, mobile and cable IP networks. IMS provides access to IP-based services independent of the underlying connectivity networks. Thus, it has been incorporated by ITU-T into its NGN architecture. IMS is a collection of functions linked by standardized interfaces that provides an abstraction layer above the underlying transport network technologies. It specifies that user equipment

could access IMS if the network access is provided via suitable IP-based network.

The IMS does not focus on standardizing implementation of services. It acts as a platform for converging application servers as long as they provide standardized SIP control interface. This means that existing telecom service platforms based on IN architecture such as CAMEL platform, OSA/Parlay gateways, and SIP servers can be reused and potentially combined as long as they provide an IMS/SIP adapter interface. IMS differs from other standard VoIP architectures in the context that it can provide secure combinational services [MAGE06]. Service operators acquiring functionality from IMS such as presence information, group management, session control and messaging can develop other services such as chat rooms, videoconferencing and other services.

Service broker has emerged in the IMS as a component that links together different service components from different server types in a flexible manner at service creation and execution time. The service broker mechanisms have not been standardized by the IMS and today are considered to be a part of a Service Delivery Platform infrastructure on top of an IMS. IMS is just a specific network abstraction for IP-based networks below the Service Delivery Platform. Figure 3.4 illustrates the platform of convergence within the NGN environment.

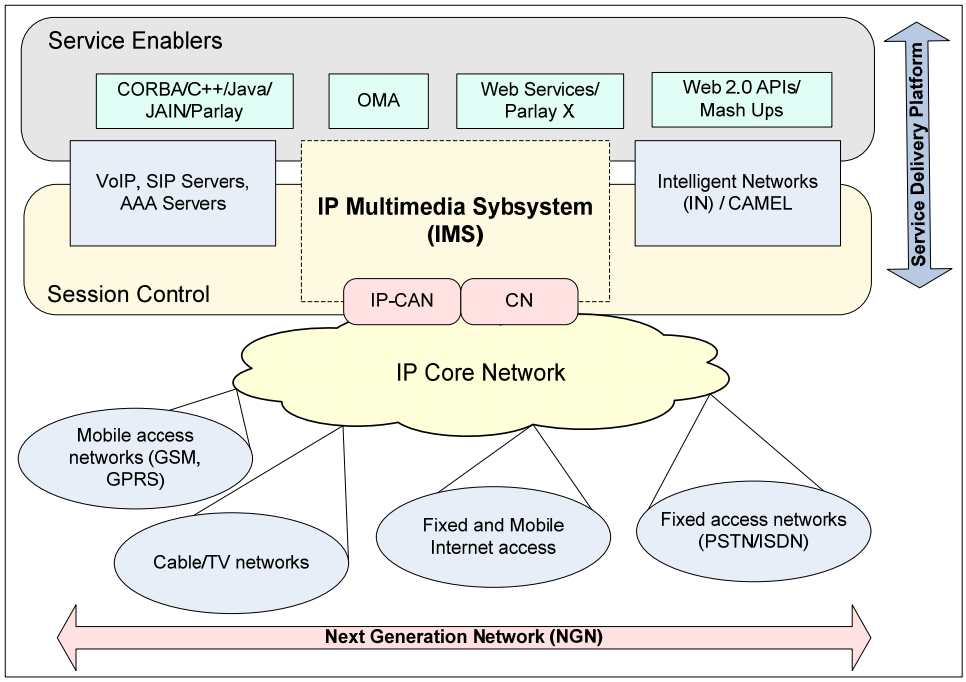


Figure 3.4: IP Multimedia Subsystem in NGN infrastructure

As seen in figure 3.4, different networks form the basis of the NGN infrastructure. NGN converges and shares the network resources of the transport networks facilitating interoperability between networks through the IMS. Within IMS, the transport layer could split into IP-Connectivity Access Networks (IP-CAN) and Core Networks (CN). An IP-CAN is a collection of network entities and interfaces that provides the underlying IP transport connectivity between user equipments and IMS entities, e.g. GPRS. A CN is a collection of entities providing IP transport connectivity between an IP-CAN and another CN, between two IP-CANs, or between two other CNs. In addition, CN provides connectivity to service layer entities, such as IMS. Over the NGN, a new application-enabling layer exists (service enablers), which is supported by the IMS, SOA and standardized service enabling frameworks. This layer is responsible for

abstracting the different access networks and decoupling the business and service logic from the underlying network implementation. In this context, several tools and development environments have been created to allow fast and cost effective service creation and delivery. These development environments form the Service Delivery Platform (SDP).

There is no a single agreed definition of the term SDP, it usually refers to a system architecture that enables the efficient creation, management, execution and operation of one or more classes of services [HP07]. SDP has emerged as a consequence of telecom network evolution towards to IP-based solution, aiming at substituting network specific ‘stove-pipes’ with common and horizontal service architecture. The benefit of having horizontal (layered) service architecture instead of vertical (stove-pipe) architecture is that the services need less time to market, are less expensive and services are independent of the transport technologies. As networks evolved from circuit-centric to packet-based networks, SDP functionality has been extended beyond communication services, to include content services, streaming services, and broadcasting services.

The Moriana Group [MORI08] describes the features of an SDP as a complete ecosystem for the rapid deployment, provisioning, execution, management and billing of value added services. SDP supports the delivery of voice and data services and delivers the content in a way that is both network and device independent. Moreover, SDP aggregates different network capabilities and services as well as different sources of content and allow application developers to access them in a uniform and standardized way.

There are a number of standardization consortia working on the SDP framework including Parlay, OMA, and TMF. SOA is a fundamental concept in the design and development of the SDP in terms of building products and delivering complex customized SDP solutions. SOA guarantees flexibility for making SDP subsystems to interwork. Moreover, SDP uses SOA to create a common set of services and a common conception of business process and business object life cycles, for example, customers, service, products, and resources. SOA principles are also used for integrating external systems with the SDP. Consequently, Web Services, orchestration and service bus concepts have become technical ingredients of complex SDP solutions. SDPs that use SOA compliant subsystems can evolve gradually, therefore maximizing the possibility of Return of Investment (ROI) [CARL08].

There are several initiatives that have tried to provide the SOA into the telecom industry as seen in this section. The use of a set of ubiquitous and open standard technologies gives SOA the capabilities to be able to function over heterogeneous networks, hardware and software technologies. SOA enables faster and cheaper service creation to the telecommunication domain. Web Service technology allows network resources to be exposed as independent building blocks that can be combined with external resources provided by third parties. Such that Telco can provide all IP-based services based on SDP framework in which service can be activated and deactivated dynamically in the service stratum of the Next Generation Network. On the other hand, devices will be added, removed and change configuration in the Transport stratum at

the same time. Therefore we do need a SOA framework in next generation network management too.

3.3.3 Managing NGN with SOA

3.3.3.1 SOA Principles

SOA has gained popularity due to the wide use of Web Services [ERL05]. Web Service technology enables service-orientation that makes use of autonomous, self-described services which are loosely-coupled by using technologies such as Simple Object Access Protocol (SOAP) [W3C07a] used as a communication protocol, Web Service Description Language (WSDL) [W3C01] used for service description and Universal Description, Discovery and Integration (UDDI) [OASIS08] used as a service registry. Beyond the basic framework of Web Services, SOA defines the service composition which is the next step in developing and extending Web Services. Through service composition, it is possible to build new services composed by other simple services. Two main models are performing the Web Service composition. The first model is the orchestration model and the second is the choreography model, [ERL10], [PEL03]. For more details about Web Services, readers can refer to Appendix C.

A middleware called Enterprise Service Bus provides technological solutions to intercept messages between services. ESB incorporates the concept of mediation and allows the interoperability between clients and data sources in Information Systems. An ESB is actually a middleware that provides integration and service composition by building services upon industrial standards such as XML, SOAP, WSDL, WS-Addressing, and

WS-Security [W3C06b], [W3C07a], [W3C01]. [W3C06a], [OASIS07a]. Moreover, ESB provides a communication channel that is mostly asynchronous by applying Message-Oriented Middleware and Publish/Subscribe methods.

The technology that enables service-oriented implementations is the Web Services technology. Web Services are interfaces describing a collection of operations that can access the network through standardized XML messages. Web Services use a standard, formal XML notion (its service description) which covers all the details needed to interact with the service, including transport protocols, message formats and location. Services can be independent from the software or hardware platform on which they are implemented and they are independent from the programming language in which they are written. This happens due to the fact that the interface hides the implementation details of the service. Hiding the implementation details allow Web Services to be loosely coupled, with cross-technology implementations. Web Services perform a specific task or a set of tasks/operations. They can be used independently or with other Web Services to complete a business transaction or a complex aggregation [KREG01]. Web Services provide a way of communication among applications running on different operating systems, written in different programming languages and using different technologies whilst using the internet as their transport. Appendix D provides a detailed examination of the ESB as well as comparisons with DOT technologies.

There are no official sets of service-orientation principles, but there are common principles mostly related to service-orientation [ERL05]. These common principles are briefly described as follows:

- **Services are autonomous**: The logic governed by a service resides within an explicit boundary. The service has control within this boundary, and is not dependent on other services for it to execute its governance.
- **Services share a formal contract**: In order for services to interact, they need not share anything but a collection of published metadata that describes each service and defines the terms of information exchange.
- **Services are loosely coupled:** Dependencies between the underlying logic of a service and its consumers are limited to conformance of the service contract.
- **Services abstract underlying logic**: Underlying logic, beyond what is expressed in the service contract metadata, is invisible to the outside world.
- **Services are composable**: Services may compose others, allowing logic to be represented at different levels of granularity. This promotes reusability and the creation of service abstraction layers.
- **Services are reusable**: Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.

- **Services are stateless**: Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- **Services are discoverable**: Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

3.3.3.2 The SOA-based NGN Network Management Architecture

NGN management has to deal with multiple vendors, multiple applications, multiple physical devices from data and voice networks, multiple databases, and multiple service layers (infrastructure plane, control plane, service plane). Any management solution for NGN must be architected in a way that it can scale to manage the current and future NGNs. This scalability challenge is a requirement for flexibility so that the solution can be rapidly adapted to support new services and technologies in the future without the need for long term and complex upgrades. By adopting the SOA philosophy, the vital management operations can be applied as services (i.e. retrieving the status of a device, controlling it, changing its configuration settings and provisioning). Services are software components with formally defined, message-based, request-response interfaces and the logic behind those interfaces is hidden from the users. Figure 3.5 shows an example of using the SOA to converge the heterogeneity of the different entities in a management system. All the FCAPS functionalities as well as the different OSSs could be integrated and operate as one OSS providing an agile management control.

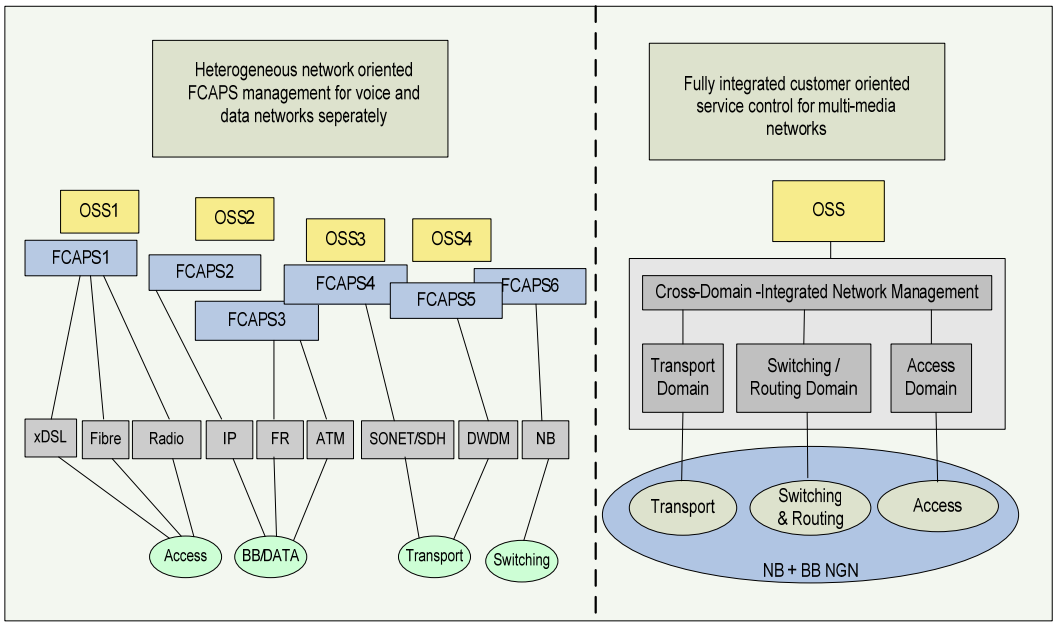


Figure 3.5: Network and Service management implementation

With reference to eTOM framework and TMN framework, a network management framework based on SOA to enable service operations and business operations is required. Enabling service operations would require the network management framework to incorporate FCAPS functions. A well-run service operations would in turn enable fulfilment, assurance and billing (FAB) functions in the business operations as defined by eTOM.

The eTOM FAB functions are summarised below:

- **Fulfilment:** operations for providing customers with their requested products and services in a timely and correct manner. It translates the customer's business or personal need into a solution, which can be delivered using the specific products in the enterprise's portfolio. This process informs the customers of the status of their purchase

order, ensures completion on time, as well as ensuring a delighted customer.

- **Assurance:** includes all activities for the execution of proactive and reactive maintenance activities to ensure that services provided to customers are continuously available and performing to SLA or QoS performance levels. It performs continuous resource status and performance monitoring to proactively detect possible failures. It collects performance data and analyses them to identify potential problems and resolve them without impact to the customer. This process manages the SLAs and reports service performance to the customer. It receives trouble reports from the customer, informs the customer of the trouble status, and ensures restoration and repair, as well as ensuring a delighted customer.
- **Billing:** involves everything necessary for the collection of appropriate usage records, production of timely and accurate bills, for providing prebill use information and billing to customers, for processing their payments, and performing payment collections. In addition, it handles customer inquiries about bills, provides billing inquiry status and is responsible for resolving billing problems to the customer's satisfaction in a timely manner. This process grouping also supports prepayment for services.

The proposed framework consists of two levels of management operations have been identified in the network management framework: the local management level and the global management level. Figure 3.6 illustrates the functions identified in the proposed model.

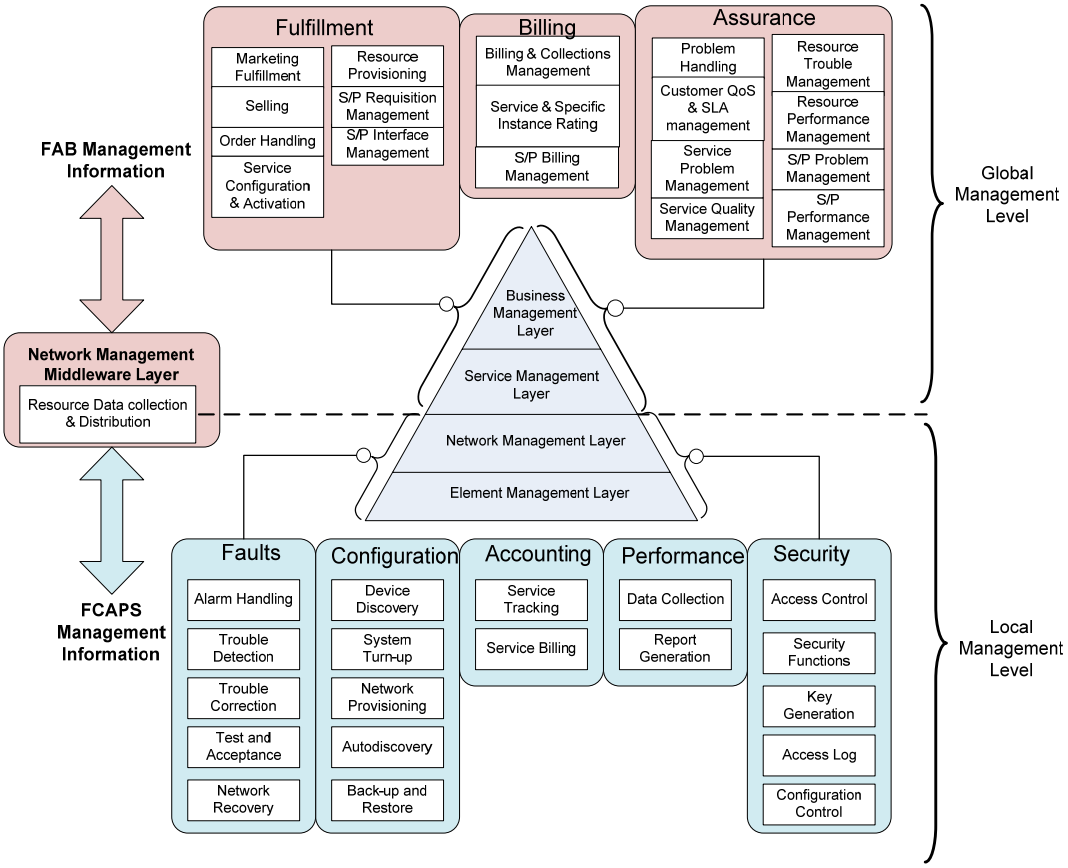


Figure 3.6: Proposed management model's functional architecture

3.3.3.3 Global and Local Network Management Functions

At local management level, individual NMSs that perform management operations within the Network provider's boundaries and are referred to as Local Network Management Systems (LNMSs). Within the NGN context, network providers operate in the transport stratum. A network provider needs to ensure that its network meets the requirements in the QoS SLAs specified by the service providers. As a result, in the proposed framework, the management functions that the network provider needs to perform are confined in the Element Management Layer and Network Management Layer of the TMN model.

At the global management level, management operations that are performed by service providers at the service stratum of the NGN architecture. Service providers use the NGN network infrastructure operated by network provides in order to provide services to their users. Hence, service providers will require to have a global view (a combination of heterogeneous management information) provided by the network operators. As such, the management functions at the global management level are performed by a Global Network Management System (GNMS) located in the Service Management Layer and Business Management Layer of the TMN model.

More specifically, LNMSs perform FCAPS management functions specified by the TMN at the local management level and the GNMS performs global management level functions include FAB operational and management functions defined by eTOM.

The proposed NGN network management framework focuses on bridging the heterogeneous management information that exists between LNMSs at the local management level and GNMSs at the global management level. Thus, functions that handle the heterogeneity in the management information need to be considered. For that reason, the framework introduces a middleware layer referred to as Network Management Middleware Layer that bridges the two management levels.

The middleware layer will primarily perform a dedicated function defined by eTOM resource data collection and distribution [TMF]. This function is responsible for performing the following operations:

- Collect management information and data

- Process management information and data
- Distribute management information and data
- Audit data collection and distribution

Figure 3.7 shows the architecture of the proposed Network Management Platform that uses the SOA to converge the heterogeneity of the different entities in a management system.

All the LNMSs as well as the GNMS collaborate as one OSS/BSS providing a fully integrated customer-oriented service control.

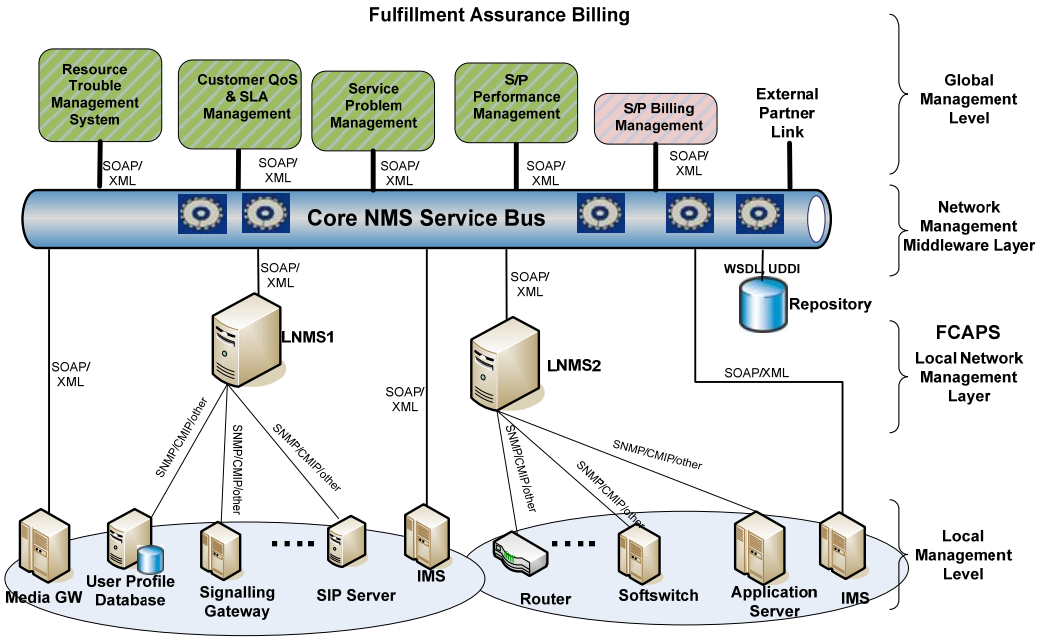


Figure 3.7: The architecture of the Proposed Network Management Platform

3.3.3.4 Network Management Architectural Layers

As illustrated in figure 3.8, the NGN Infrastructure Layer contains the managed devices or resources that form the NGN infrastructure such as Softswitch, Media Gateways, IP Multimedia Subsystems, etc. These devices use different management protocols for carrying out management

information. The resources containing agents are processing entities that receive management requests and send management responses to the Network Management Layer.

The next layer in the NGN management infrastructure is the Local Network Management Layer. This layer contains LNMSs that perform FCAPS functions. It makes use of NMSs which are controlling entities that collect management information from the agents residing in the managed resources. Figure 3.8, shows an example in which different management applications are used for managing different resources with various network management protocols as discussed in the previous Chapter 2. The network devices, routers, application systems, etc. are part of the resources to be managed.

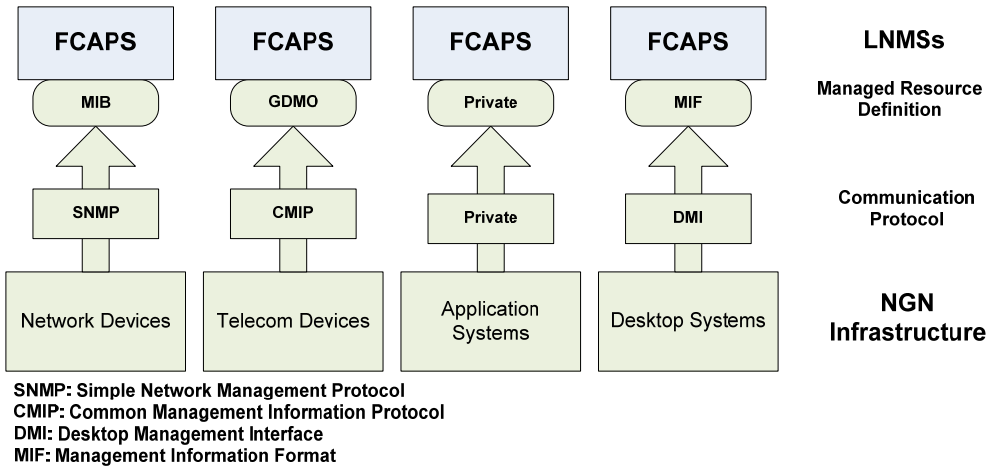


Figure 3.8: Local Management Level, network management protocols

The management information collected from different managed resources is stored in LNMSs databases. The management information needs to be distributed to other external management systems such as trouble

ticketing system, at the global network management level and other OSSs, etc. For this purpose, an XML gateway designed in order to extract management information from the database and send it to the Network Management Middleware Layer. The XML gateway is defined for the purpose of mapping the management information into XML-based messages. After mapping, the gateway transmits the XML messages via SOAP protocol to the Network Management Middleware Layer. The XML gateway is a software component that can reside in LNMS or can be allocated at the Network Management Middleware Layer to cater for legacy network management systems that do not have the XML gateway installed.

The Network Management Middleware Layer is designed with open standards and developed by using open source software. Since the commercial network management systems such as HP OpenView are proprietary, it is expensive to run and hard to maintain. The benefits of open standards were mentioned in chapter 2. In addition, open standards ensure compatibility and choice. The main advantage of open source software is that it is free but the disadvantage is to find support if the user has any problem [GALL05].

The Network Management Middleware Layer consists of the Core NMS Service Bus that utilizes the Resource data collection and distribution function standardized by the eTOM. The Core NMS Service Bus performs adaptation functions for translating different management messages into a unified format. Moreover, this layer performs dynamic routing and dispatch of management requests to multiple receivers at the global management

level such as GNMSs from different service providers. In addition, this layer is also responsible for deriving information models to map different types of data formats, using XML as the means of exchanging management data between heterogeneous management systems. This concept involves the passing of management data asynchronously among heterogeneous management systems using a communication channel that carries self-contained units of information. XML is used for this purpose as a document exchange by exchanging structured data among management systems. The management data received from the different network elements on the infrastructure layer are mapped into XML-based messages and transmitted over SOAP protocol through the XML-gateway's northbound interfaces. XML is suitable for coping with multiple information models due to the fact that the management data encoded in XML documents are self-describing. Using message-based communication, the physical resources such as signalling gateways and routers are abstractly decoupled from the higher level management applications. As a result, senders (i.e. IMS, Signalling G/W etc.) and receivers (i.e. trouble ticketing systems) are never aware of each other. The middleware layer is responsible for getting the management messages to their intended destination. The Core NMS Service Bus manages the connection points among multiple management end points, as well as the multiple channels of communication among the connection points.

On top of the Network Management Middleware Layer reside the GNMSs where high/peer managers communicate with the Network Management Middleware Layer via northbound interfaces exchanging messages based on XML format. This high level layer consists of multiple services that are responsible for performing management functions and taking decisions accordingly. For instance, trouble ticketing systems, which are responsible for notifying the service operator of faults that have occurred in the managed network. Furthermore, other management systems can be connected in this layer such as BSS that are performing business management functions such as customer care and customer billing according to the management information received by the Network Management Middleware layer.

3.4 Conclusion

Next Generation Networks will accommodate heterogeneous networks with high level of distribution and complexity. Thus, it will issue new challenges to the OSS architectures. The traditional OSS architectures will no longer be able to support the complexity of the NGNs as a result, the redesign of the management architecture is necessary.

This chapter examined the Distributed Object Technologies that have been used by the telecom operators for integrating their networks. This chapter concluded that these approaches are not capable of supporting the Next Generation Network's management plane. It further illustrated that the focal point of the telecommunications networks is now shifting from traditional architectures to SOA-based architectures. Moreover, in

this chapter, the SOA concept has been introduced as well as the Web Service paradigm, in order to illustrate the benefits of that technology, which is the enabler of the SOA philosophy. Architectures using the Service-Orientation principles could deliver agility, scalability, reusability, and flexibility in distributed heterogeneous environments such as NGN. Finally, the proposed Network Management Platform that has been designed based on the architectural principles has been presented in the chapter.

Chapter 4： Network Management Systems

4.1 Introduction

Networks and distributed processing systems are growing rapidly and have become critical in today's businesses. Network management will help to ensure high network availability, secure communication, effectively manage network devices, easy use of the network and related technologies. Many network management architectures and models have been proposed by various standard organizations and vendors [STAL99]. Some of them are widely implemented in the real world while others are concepts at the development stage.

In this chapter, the design of a Local Network Management System based on the SNMP framework with performance, fault and configuration management functions is presented. This chapter focuses on two major distinctive management components: an NMS and the agents in the managed devices.

First, the chapter presents the design and the development of an LNMS that consumes management information obtained from agents. The NMS performs performance, fault and configuration management functions and has been developed as a web service. Lastly the design of an XML-gateway that converts stored management information into XML data format in order to connect to the middleware that integrates heterogeneous network management systems together, is presented.

4.2 Levels of Management Communication

In the Network Management Platform (NMP), there are two levels of management communication: the Low Level and the High Level Management Communication. The Low Level Management Communication refers to the communication among Network Elements and their associated LNMSs and the High Level Management Communication involves the interactions between LNMSs and GNMSs at the global management level. From the TMN architecture point of view, the Low Level Management Communication involves management interactions between the managed objects at the physical devices and the FCAPS functions at the LNMSs by using management protocols such as SNMP, CMIP etc. The High Level Management Communication involves management interactions between the FCAPS functions and the FAB functions at the GNMAs by using XML. The classification into the two levels of management communication enables different protocols and architectural patterns in the management architecture to perform management functions in every layer of the NGN framework. Figure 4.1 illustrates the relationship between the levels of management communication and the management layers of the TMN model. This chapter focuses on the Low Level Management Communication between NEs and NMSs.

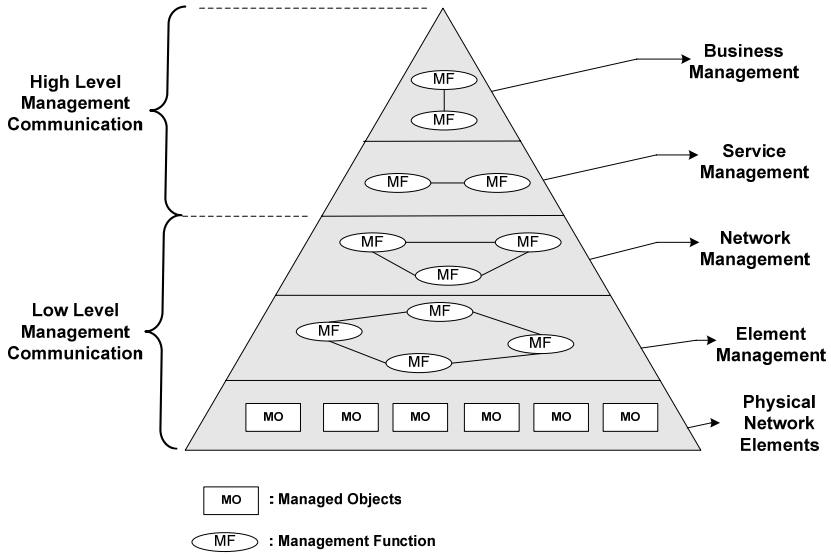


Figure 4.1: Relationship between levels of management communication and the management layers of the TMN model

4.3 Components of Network Management Systems

An NMS framework consists of the following components:

- Managed devices or NEs, each with an agent, which provides remote access to management information.
- A manager (management system) that runs management applications to monitor and control managed elements.
- A management protocol, SNMP, CMIP, etc. that is used to convey management information between the management systems and agents. Management information is a collection of managed objects in MIB format.

Figure 4.2 illustrates the interaction between NMS and NEs (NE). Network A and Network B are two different networks that are managed by different network operators. NMS is the managing system that is responsible for collecting management information from the NEs to perform the FCAPS

functions. The communication link provides the path for exchanging information between the NMS and the NEs.

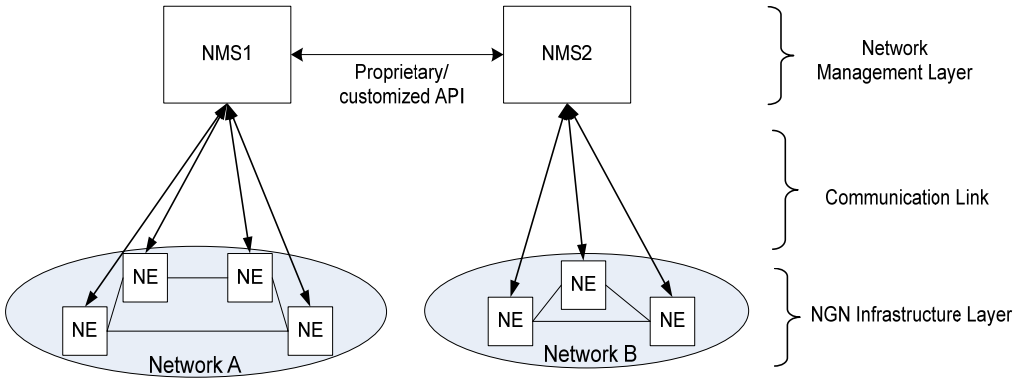


Figure 4.2: Network Management interactions

For the Telco network operator, NMS provides the appropriate tools for them to manage their networks [AMIR95]. These tools are applications to monitor the network, service provisioning systems, trouble ticketing, network planning etc. Unlike the NE, a management system exists only for network management. If a management system fails to function, the network itself should not be affected. This is a fundamental requirement for the operation of the NMS. However, without a management system, network monitoring and maintenance will become much more difficult. If an element in the network fails, the failure will go undetected and consequently, the quality of the services (QoS) provided by the network will be degraded. The communication among NMSs is performed via proprietary APIs. These APIs are implemented by the NMS software providers for external communication. Each NMS has its own API for external communication but these APIs are usually providing limited or

restricted access to other applications (i.e. other NMSs). Furthermore, it forces other applications to support the proprietary API in order to extract management information.

Figure 4.3 depicts a typical NMS architecture that consists of three layers [STAL99], [CLEM07].

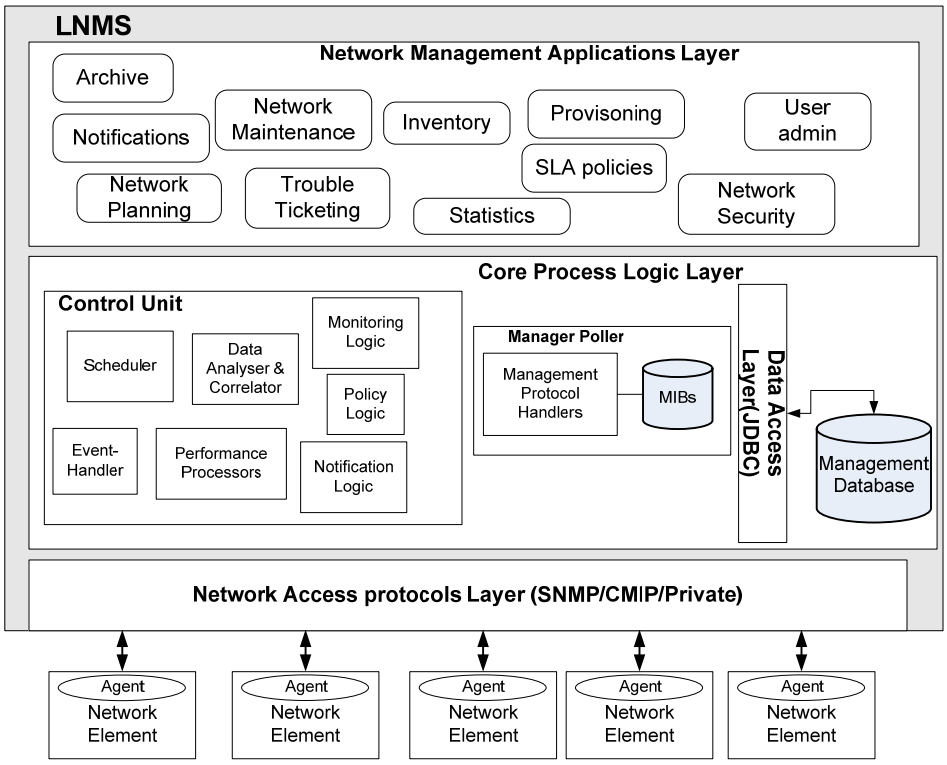


Figure 4.3: NMS functional architecture

4.3.1 Network Access Protocols Layer

The Network Access Protocols Layer is concerned with transport functions. It contains mechanisms for establishing socket connections with the underlying network. For instance, this layer establishes SNMP, CMIP, etc. connections when requested by the NMS. For example, when an SNMP request is specified by the NMS, the Network Access Protocol layer establishes a UDP connection at 161 port number.

4.3.2 Core Process Logic Layer

The Core Process Logic Layer, is responsible for performing core management functions. It contains the necessary functions in order to acquire management information from the network, process the management information and finally store it to the management database [CLEM07].

All the NMS's logic is contained in this layer including the Manager Poller, the Control Unit, and the Management Database, which are described below:

- Management Database: The management database stores all the necessary information that is concerned with the management information retrieved by the agents as well as operational information that is required by the NMS. Current NMSs use relational databases (i.e. PostgerSQL) as an NMS database. A Relational database support dynamic views; hence, changing the data in a table will alter the data depicted by the view. It also hides the complexity in the data and reduces the data storage requirements. Moreover, it contains user access credentials, authentication information, etc. to allow the database administrator to implement authentication and authorisation mechanisms in order to control access to the data in database tables. Thus, using relational database will increase the performance of the NMS compared to using standard databases with no inter-relationship between different tables and they can meet all types of data needs [DATE06]. The database access is performed via a Java Database

Connectivity (JDBC) API that provides methods for querying and updating data in the database. JDBC uses common and standard method calls to allow the NMS to use different databases for storing the management information.

- Manager Poller: This includes a Management Protocol Handler and a master MIB data store, described below:
    - Management Process Handler: This provides mechanisms to create management requests for the collection of management information from agents residing in every device of the network. It processes each management protocol (SNMP, CMIP, TL1, etc.), encodes and decodes messages received by the agents and creates requests by making use of different primitives.
    - MIB data store: contains a pool of Management Information Bases (MIBs) related to the network's information structures and data attributes that the agents in the managed network use to enable the NMS understand the information that it retrieves from the agents.
- Control Unit: This performs functions related to the processing of the management information through different functional components in order to process the management information including the following:
    - Scheduler: The scheduler is a function that instructs the Manager Poller to collect management information from agents at specific time intervals.

- Data Analyser and Correlator: This functional entity provides context to the data being collected. This function maps the agent's information to an understandable format. For instance, the agent sends the status of a router (i.e. UP(1)) and after the data analysis function the status will be stored in the database as "the router is working".
- Performance processor: Collects and processes performance statistics based on the information that is retrieved from agents.
- Monitoring Logic and Notification Logic: Apply rules for specifying parameters related to charts creation, notifications/events creation and storage.
- Event Handler: Provides the communication channel between the NMS and the agents. This function encapsulates various parameters such as community strings, protocol version, variables etc.
- Policy Logic: Provide domain specific assets of the NMS such as different access levels and user customization features. Furthermore, it provides automated management and execution of both short-duration and long-duration management tasks.

4.3.3 Network Management Applications Layer

This layer consists of front-end user applications that contain application logic as well as GUIs in order to interact with the network administrator.

Most modern applications are presented on web pages so that they can be accessed remotely [SUBR00]. These applications can vary from one NMS to another depending on how complex and how complete a software solution the network operator requires. These applications are tightly coupled with the NMS's Core Logic and cannot be modified, integrated or used on other NMS's Core Logic. The Network Management Applications do not have direct interaction with the Core Logic of the NMS. All interactions are performed through the NMS database as can be seen in the figure below (figure 4.4). The User administrator application is the only application that controls the Core Logic of the NMS. This application is responsible for configuring the NMS, such as initializing the NMS, specifying time intervals within which the agents should be invoked, indicating specific parameters that the agent requires, etc.

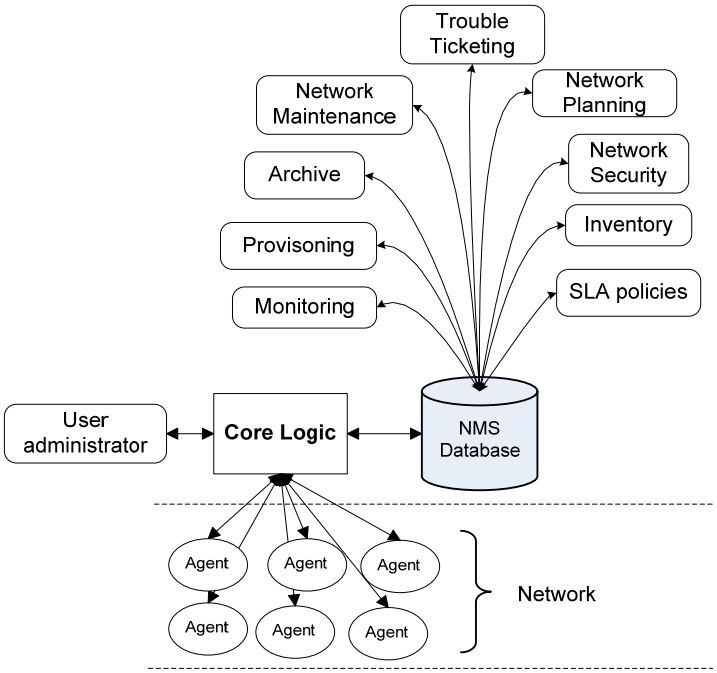


Figure 4.4: NMS relationships

Commercial NMSs cannot be easily customized in order to meet the network provider's requirements and they are difficult to integrate with other NMSs. The reason behind this problem is that NMSs are proprietary products developed by NMS software vendors, which in most cases are not tailored to meet individual network operations' needs [YOON06]. Any modifications to the network management infrastructure imply modifications to the network management software by the software vendor. This adds a substantial extra cost to the implementation and maintenance of network management infrastructure. In comparison to commercial NMSs, open-source NMS can be easily modified and customised to meet the requirements of a network operator but the NMS applications are usually not very comprehensive and complete as the commercial solutions [MAUR01].

Most existing NMSs are monolithic OSS systems with legacy management applications and are usually heterogeneous in nature operating in isolation. The management information extracted from the network infrastructure is stored in a management database and remains isolated without being used by higher layers of the management framework, for example, Service and Business Management Layers.

The emergence of the NGN will require the collaboration and the convergence of those individual heterogeneous management systems to create an agile management framework that can meet the business needs of the different service and network providers.

4.4 Local Network Management System design in an NGN Infrastructure

4.4.1 Network Management Requirements

ITU-T has specified requirements for managing the NGNs [M.3060]. This thesis categorizes those requirements into two distinctive parts: the local management requirements and the global management requirements. Local management requirements are requirements that can be applied within the boundaries of the network/service provider. Global management requirements are specific to a global management framework that requires management information exchange among different organizations.

The local requirements require that .the management infrastructure should have the ability to [M.3060]:

- proactively monitor trends;
- manage customer network;
- integrate end-to-end services provisioning;
- deliver management information to the management information user and to present it in a consistent and appropriate manner;
- automatically and dynamically allocate network resources;
- support service quality-based network operations;
- provide survivable networks in the event of impairment;
- ensure secure access to management information by authorized management information users, including customer and end-user information;

- support the availability of management services any place any time to any authorized organization or individual (e.g., access to billing records shall be available 24/7);
- support the collection of charging data for the network operator regarding the utilization of resources in the network either for later use by billing processes (offline charging) or for near-real time interactions with rating applications (online charging).

The global requirements focus on the ability of the NGN management infrastructure to operate as one integrated management framework consisting of multiple management systems. These global requirements cannot be met by using NMSs that operate in isolation. Instead management systems need to interoperate. A proposed Network Management Middleware Layer will handle the intercommunication among different management systems that will allow the NGN management architecture to meet the global requirements.

The global requirements require the NGN management system to [M.3060]:

- provide the management capabilities that will enable organizations offering NGN services to enable end-user service improvements including customer self-service (e.g., provision of service, reporting faults, online billing reports);
- provide management functionalities that are independent of company organizations, which are subject to change, while maintaining the concept of organizational boundaries;
- exchange management information across network boundaries;

- provide an abstracted view on resources (network, computing and application) that hide complexity and multiplicity of technologies and domains in the resource layer;
- provide consistent cross-technology management interfaces on NEs (service and transport elements) allowing an integrated view of resources and include available management technology implementations, as appropriate;
- set business processes and management services that will enable service providers to reduce the time-frame for the design, creation, delivery, and operation of new services;
- manipulate, analyze and react to management information in a consistent and appropriate manner;
- allow an enterprise and/or an individual to adopt multiple roles in different value networks and also multiple roles within a specific value network;
- support B2B processes between organizations providing NGN services and capabilities.

4.4.2 Local Network Management System Design

The local requirements should be fulfilled by the NMSs of individual underlying transport networks (e.g. WLAN, UMTS, WiMAX, etc.). However, satisfying the local requirements alone will not enable inter-communication among different management systems, as can be seen from the NMS architecture shown in Figure 4.3. Such NMS architecture

represents a self-contained, standalone, isolated architecture where communications with other NMS is virtually impossible.

As the ITU has emphasised the need for global collaboration among service/network providers and the need for collaboration among different management systems, the development of individual NMSs should take into account the global requirements.

For an individual NMS, hereafter referred to as the local NMS (LNMS), to become a part of a global network management architecture, its management information should be exposed to other systems via standardized interfaces that are technology neutral. Moreover, LNMSs should be loosely coupled in order to be repurposed and reused without being dependent on other management systems. To integrate different LNMSs together and to be able to fulfil the NGN global management requirements, there is a need to share and exchange data with a common message format.

To realise this vision, an LNMS architecture for individual underlying transport networks of the NGN based on the Web Service concept is proposed. Web Service technology enables SOA, which can be applied in order to solve the integration aspects of the management architecture.

Figure 4.5 illustrates a LNMS architecture that extends the architecture of Figure 4.3 by building on top of the architecture a Web Service Layer that provides XML-gateway functions in order to expose the management information in a common format.

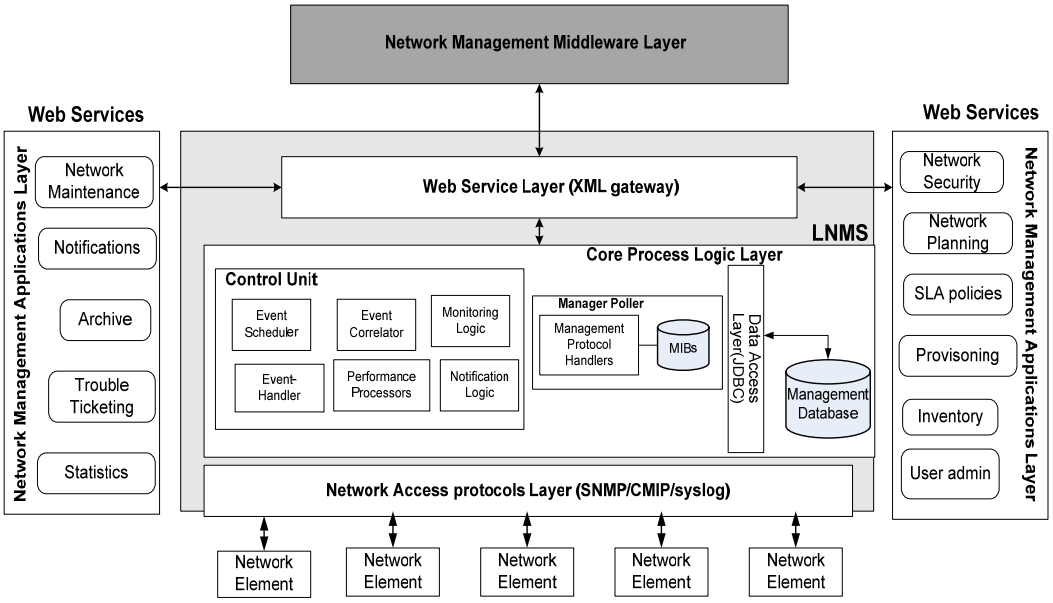


Figure 4.5: Local NMS Architecture

4.4.3 Core Process Logic Layer Development

The development of the Core Process Logic Layer is based on the SNMP framework [BLUM99]. Such development exploits and extends open-source software currently available. The major effort in developing the Core Process Logic Layer is to expose the LNMS management information, which can be used by other LNMSs and GNMS that reside on higher layers of the management architecture.

The following figure (figure 4.6) depicts the developed Core Logic.

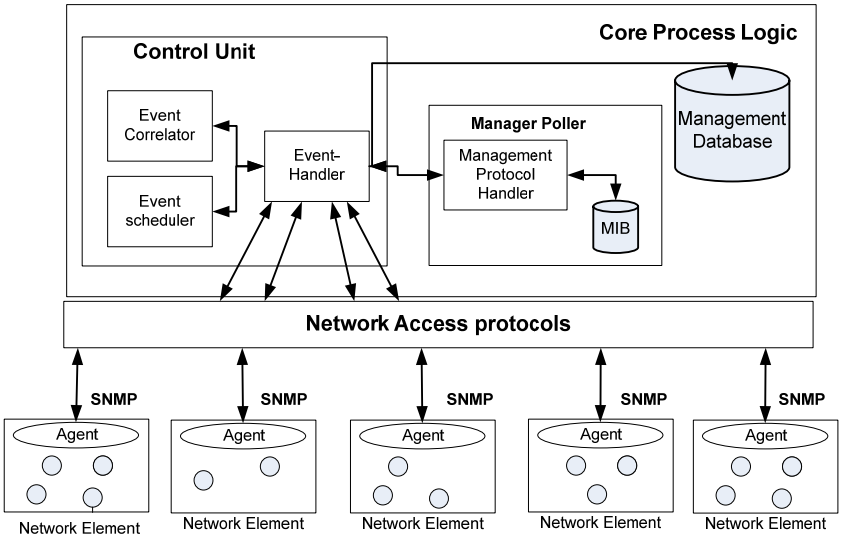


Figure 4.6: Core Logic functional architecture

4.4.3.1 Control Unit

The Control Unit is the core management component that provides event-handling, event correlation, and event schedule and archive. It consists of the following components:

a. The Event-Handler component listens for messages that the agents in the NEs send. It is also responsible for sending management requests to the appropriate agents, and the management information that it receives is stored into a database.

b. The Event Correlator component is used to match an incoming event to a specific notification or an action list and provides context to the data being collected.

c. The Event Scheduler component is responsible for scheduling and archiving events. Due to the large amount of information a relational database management system is deployed for storage.

The Control Unit is developed by using the open-source OpenNMS management tool [OPENNMS]. OpenNMS is a popular enterprise-grade network management tool that performs a number of functions including device discovery, service and performance monitoring and event management [OPENNMS]. The following packages of OpenNMS's back-end event management are used:

- opennms-correlator: is used for as the Event Correlator and Event Handler. Furthermore the performance functions are implemented in this component.
- opennms-reporting: Is used for implementing the event schedule and achieve functions.

For the management database, PostgreSQL has been used. PostgreSQL is an open source relational database that is used for storing management information captured by the LNMS. This relational database forms a persistence tier in the LNMS architecture [POSTGRE].

4.4.3.2 Manager Poller

The Manager Poller is an SNMP enabled component that is based on the concept of the SNMP session. A session is a communication channel between the LNMS and the remote agents. A session encapsulates various parameters, such as community strings, protocol version, and packet encoding. Once a session has been established, the LNMS can communicate with the remote agents by sending requests and waiting for responses through the Event Handler component. The Manager Poller processes the SNMP protocol. It encodes and decodes messages from

ASN.1 to usable internal formats. It creates requests by making GetRequest-PDU and SetRequest-PDU. Furthermore, this component processes the GetResponse-PDU, handles errors, receives and takes actions from traps (trap-PDU) that have been sent from the remote agents. Each network management application requires an object that implements the interface for the Management Protocol Handler. Since SNMP is used as the management protocol, the Management Protocol Handler is now referred to the SNMP-handler for simplicity. The SNMP-handler interface is responsible for processing received SNMP-PDU on behalf of the network management application. If an error occurs with the session, the handler is informed of the error. The Manager Poller interrogates its MIB to obtain information about the proper set of managed objects that can be monitored and controlled. The Poller must interface to the UDP layer through the event-handling component in order to send and receive the SNMP messages. The MIB of the LNMS contains a master list of the MIBs from all of the agents in its community. If an LNMS is to control each agent's MIB variables, it must know those variables.

The development of the Manager Poller is based on the open-source API SNMP4J [SNMP4J]. The Manager Poller uses the Event-Handler component in order to open the SNMP ports. SNMP protocol occupies two UDP network ports: the 161 port, which sends and gets SNMP messages from the agents, and the 162 port, which only receives notifications from the agents.

The processes and the functions performed by the Core Logic layer are described in detail in Section 4.5.

4.4.4 Agent Development

4.4.4.1 SNMP Agent

The management information exchange pattern used for the Low Level Management Communication is based on the manager-agent model of the SNMP framework [HARR02]. As mentioned in chapter 2, SNMP faces limitations such as scalability and efficiency that will not be able to meet the demands of the NGNs. On the other hand, SNMP is already a well established management protocol that most of the network and service providers are using today for managing their infrastructure due to its simplicity [MAUR01]. For instance, MPLS network switches, which are used in NGN for creating virtual links between NEs, have defined MIB structures and use SNMP as a management protocol [CISC07].

One reason that SNMP is used for the implementation is to minimize the complexity of management functions performed by the agents [SUBR00]. This means that the agents can be simple and lightweight and as a result agents with small footprint can be embedded in virtually any NE with low processing power [STAL99].

SNMP4J API [SNMP4J] has been used for the development of the agent. SNMP4J API is an open source API based on object-orientation used for developing Java-based managers and agents. Java has the advantage of platform independence with built in support for network sockets and threading [MAUR01]. Another advantage to Java is that creating

multithreaded applications is very easy. SNMP4J API provides all PDU types (supports all SNMP versions V1, V2 and V3), transport mapping with UDP, synchronous and asynchronous communication (traps), row-based efficient asynchronous table retrieval with GETBULK and multithreading support. SNMP4J has a built-in thread pool model so that we can specify the number of threads that respond to and process incoming request, making SNMP applications highly efficient.

The SNMP4J Command Line Tool (CLT) has been used for sending SNMP requests to the agent. Agents have been installed on a server and Linksys WRT54G wireless router installed with open source firmware, DD-WRT, a Linux-based open source firmware [DDWRT]. DD-WRT is designed to replace the firmware that ships pre-installed on many low cost commercial routers as it provides many features that are not supported by those commercial routers, e.g. the IPv6, Wireless Distribution System, RADIUS, and advanced quality of service, The server and the wireless routers have been used as an example to illustrate the information that is required for performing FCAPS functions. The same management information could be extracted from other NEs such as IMS, network bridges, etc. because the variable names are standardized by the MIB RFCs [KAVA00].

The SNMP agent that resides in each NE, must be able to read and write the management information, receive and transmit messages through the UDP transport interface, and should be able to generate trap messages. Figure 4.7 depicts the architecture of the agent showing the interactions with external entities. There are two external components (UDP,

Instrumentation Routines) and two data stores (Configuration Data Store, MIB Data Store).

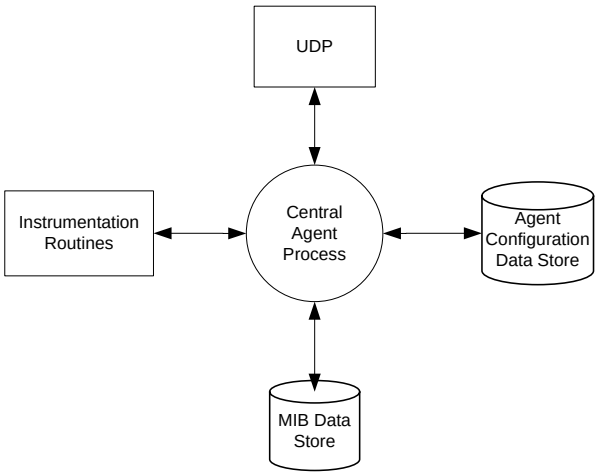


Figure 4.7: Architecture of Software Agent for Network Management

As it is specified by the SNMP specification [HARR02], the Central Agent uses the UDP as the transport protocol. The agent uses UDP protocol instead of TCP protocol due to the fact that each UDP packet does not need to be acknowledged and as a result, it adds less overhead to the network. When the agent has been successfully initialized, it listens and receives requests from the LNMS at port no. 161. When the agent receives a request, it processes it, and sends the response to the LNMS. Moreover, the agent can send asynchronous trap events to the LNMS informing it of some predefined condition that has occurred.

The Instrumentation Routines reside on the agent's network device. These routines determine if a requested object is in the agent's MIB, verify the access mode (read-only mode or read-write mode), know the location of the object, and determine if the agent can retrieve or set the value.

The Agent Configuration Data Store holds information such as the agent's community name, the collection of managed objects, IP addresses of the LNMS for sending the traps (or notification), and the agent's system variables (description, location, community name). The agent retrieves this information during the initialization in order to start up operations and enter the listening stage to read and write messages. The MIB Data Store contains all the objects that can be managed by the agent. MIB is a collection of information that is organized hierarchically and contains information about the system, such as temperature, location, interface status and interface queue utilization. Any sort of status or statistical information that can be accessed by the LNMS is defined in the MIB Data Store.

4.4.4.2 Agent Processes

Figure 4.8 illustrates the agent processes. These are: Initialization Process, Main Protocol Process and Trap Handler Process.

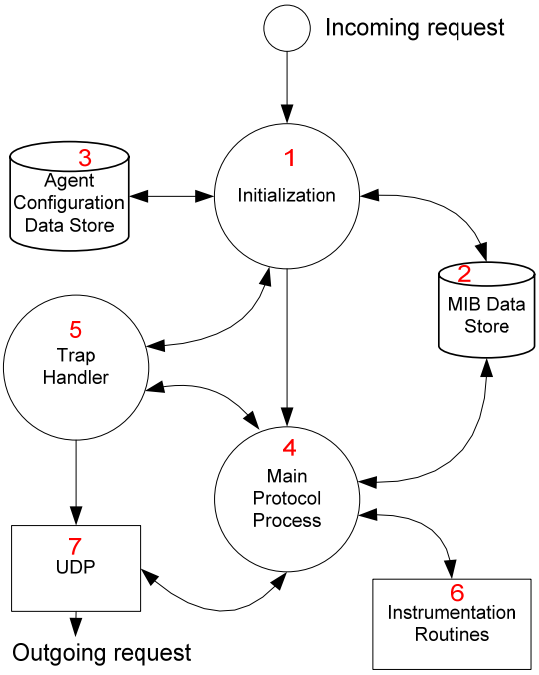


Figure 4.8: Agent Functional Architecture

4.4.4.3 Initialization Process

During the initialization process, the agent gets parameters from the Configuration Data store and MIB data store. In order to connect the UDP interface, the agent makes a socket call to get the socket descriptor and then binds to the socket ports and is ready to receive data. Figure 4.9 shows the implementation code of the agent's initialisation process.

```
protected void initTransportMappings() throws IOException {
   transportMappings = new TransportMapping[1];
   transportMappings[0] =
      new DefaultUdpTransportMapping(new UdpAddress("127.0.0.1/161"));//indicates
the localhost
}
```

Figure 4.9: Initialization Process

4.4.4.4 Main Protocol Process

The Main Protocol Process performs the following functions:

- Receives incoming requests.
- Performs requested Get/Set operation.
- Sends response to requesting client.

The Main Protocol Process is in charge of receiving the incoming message requests from the NEs. The LNMS sends a request to the agent that resides in the NE. The incoming request is read from the transport interface. It is then validated by the Agent Main Protocol Process. For instance, the protocol process checks if the type of the request is an integer, an octet string, or a counter type with length of 1, 128 or 256 bits. Furthermore, this process validates the version of the incoming message request and the community name. The validation of the version process compares the received version number value with the agent's configured version value to be sure that they are the same. The mismatch of the version numbers can cause the received message to be discarded. The validation of the community name process compares the received community name with the community name for which the agent is configured. If the community names do not match the message is discarded.

The Main Protocol Process handles the Protocol Data Unit (PDU) requests. For example, it determines the PDU type of the message request and calls the appropriate function to process that particular PDU type. The PDU type denotes the operations that are embedded in the message request (GetRequest, GetNextRequest, GetResponse,

SetRequest, and Trap). The requested MIB variables that are carried in the message request are mapped into an internal, local format. If the MIB objects are present in the MIB Data Store, the Main Protocol Process verifies the access mode (read-only or read-write mode) via the Instrumentation Routines process and performs the requested Get, GetNext or Set PDU operation on all of the objects in the message's request. When the command has been carried out, the message is transmitted to the LNMS. The main protocol process sends the created packet to the UDP layer for transmission back to the LNMS.

4.4.4.5 Trap Handler

The Trap Handler is responsible for sending traps, i.e. notifying events to the LNMS. It contains two processes, each of which is responsible for a specific function. These processes are:

- Process Trap request called by the agent when a trap needs to be sent to the LNMS.
- Send response to the requesting client.

The Process Trap request sends linkUp and linkDown traps, if this condition is detected. The agent needs to be configured in order to send these traps. The configuration profiles are stored into the Configuration Data Store. When a trap has been initiated, the message is passed to the Send Response process to the requesting client. This process sends the trap message to the UDP layer for transmission back to the LNMS. The agent must always know the IP address of the LNMSs to which this trap will be sent.

4.4.5 XML-gateway component

4.4.5.1 XML-Gateway Functions

The XML-gateway maps the management information into XML-based messages and through the SOAP protocol it transmits the information to other Web Service applications. It implements the following functionalities:

- Standardized communication protocol (SOAP) for information exchange.
- A service contract based on WSDL that can be used by other Web Service applications in order to bind to the XML-gateway.
- Exposure of the management information over the internet.
- Management information expressed in XML.

The XML-gateway provides network management information to the Network Management Middleware Layer above. Through information obtained from the XML-gateway, the Network Management Middleware Layer enables communications and co-ordination between different LNMS to provide global network management functions. This layer will be studied thoroughly in the next chapter.

XML is used for many reasons. First, XML technology is standardized, endorsed by software industry market leaders. It is simple, easy to be read and understood. XML syntax consists of text-based mark-up that describes the data being tagged; it is both application-independent and human readable. This simplicity and interoperability features have helped XML achieve widespread acceptance and adoption as a standard for exchanging information between heterogeneous systems in a wide variety

of applications, including Web Services. XML is currently the most sophisticated format for distributed data that can cover all existing data structures [HARO04], [CARE02a].

In the proposed LNMS architecture, the Network Management Applications are also expressed as Web Services. These Web Service applications are decoupled from the LNMS and use the WSDL interface definition contract in order to bind to the LNMS. The Web Service applications call the LNMS by using SQL calls encapsulated in SOAP requests. As a result, in the LNMS, Web Service applications such as archive, network planning, inventory etc. can be distributed (reside in different hosts) and implemented on different software platforms. This allows applications to be loosely coupled with the LNMS and could be a collection of different softwares provided by different vendors. These applications are used as local network management services operating under the network provider's own boundaries.

The XML-gateway performs SQL requests required in order to retrieve, update, and delete information from the management database. These functions are presented in the table below:

Table 4-1: functions performed by the XML-gateway

Function	Functional description
SELECT	Selects data to be presented from one or more table in the management database
UPDATE	Updates data in the management database
DELETE	Deletes data from the management database table

Figure 4.10 presents the XML-gateway component that has been created for converting and representing management data derived from LNMSs into XML-based format.

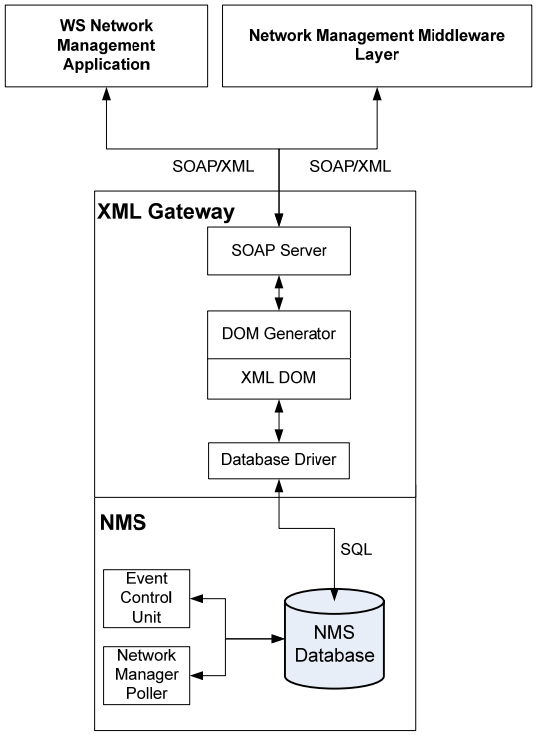


Figure 4.10: XML-Gateway Architecture

The database driver in the XML-gateway is a standard SQL-level API intermediary for accessing the LNMS database. It allows the construction of SQL statements and embedding them into API calls in order to query the LNMS database. The commands that the database driver uses in order to query data from the LNMS database are standardized SQL commands. This gives the ability to the XML-gateway to use different LNMS databases without changing anything to its logic.

Document Object Model (DOM) is an API that provides an object representation of an XML document. It provides a programmatic paradigm for giving access to objects represented by the document [W3C98]. DOM is a standardized technology supported W3C [W3C98]. It can be used in order to create, read, update, and delete elements. The XML-gateway uses DOM technology in order to create a new XML document, and then add elements to this document from the LNMS database table (Event table). The XML DOM component provides data structure for data conversion. The DOM generator function creates a DOM tree using management information stored in the LNMS database table. It generates an XML document on the basis of the DOM tree and delivers it to the Network Management Middleware Layer.

The XML message created by the XML-gateway component has one entity called root entity. All other entities must belong to that root entity. Entities are defined with a start tag and an end tag. For example, a start tag in the message that the XML-gateway component produces is <eventid>, and the end tag is </eventid>.

When one entity is embedded in another entity, the start and end tags of the embedded entity must both reside within the start and end tags of the embedding entity. The most fundamental concept of XML is that the tag set is not fixed but rather extensible [CARE02a]. This means that different LNMSs can define their own tag set and in effect create a new language for describing elements in a certain domain. This is a very powerful concept, because instead of having every type of information using the same set of descriptive rules, the information existing in each LNMS can

have their own type of expressing the information according to their own particular descriptive rules. Figure 4.11 illustrates the XML-based management information that is extracted by the XML-gateway component. This management information is extracted from agents by the LNMS and resides in the LNMS database.

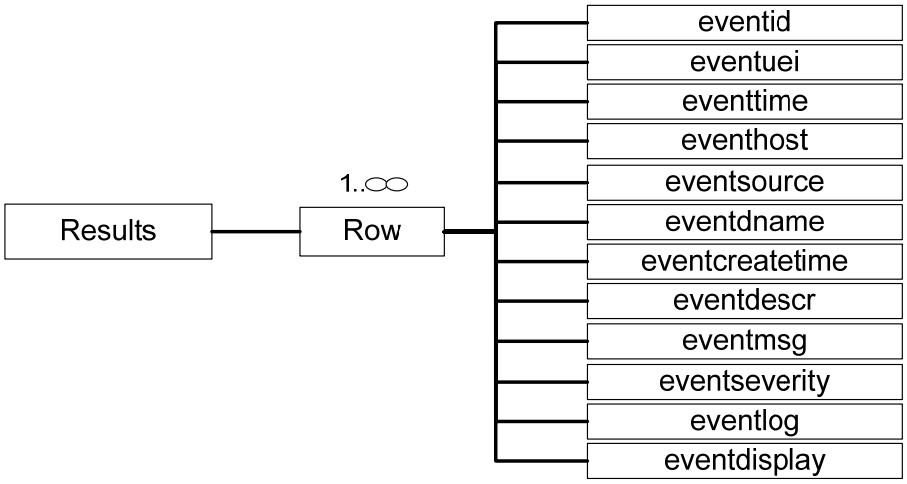


Figure 4.11: Representation of the XML-based management information created by the XML Gateway

4.4.5.2 Process for Converting SQL data into XML-based message

Several models have been proposed over the last few years related to the conversion of management information into XML. [YOON06] proposed a gateway that translates standard DOM interfaces to SNMP operations, which provides a method for XML-based manager to directly access management information through the DOM interfaces. In a message level translation, it translates HTTP messages through URI extension with XPath and XQuery, which provide methods to define detailed request message for XML/HTTP communication. The gateway uses the SOAP protocol, which is accepted as a standard protocol for XML.

[MART00] and [MART02] presented an idea to use XML for integrated management on Web-based Integrated Network Management Architecture (WIMA). The advantages of HTTP/XML-based communication are described, and the basic idea concerning SNMP MIB to XML conversion is also presented.

[STRA99] presented a library to access SMI MIB information, “libsmi”, which translates SNMP MIB to other languages, such as JAVA, CORBA, C, XML etc. This library provides a tool for MIB dump into an XML document based on metamodel-level schema mapping.

The above mentioned models focus on converting SNMP information to XML. These approaches can be used within a homogeneous network management environment, where the only protocol that can be used is the SNMP. Within the scope of the NGN management, the networks are heterogeneous and a variety of different management protocols will be used. As a result, these approaches cannot fulfil the NGN requirements.

The approach proposed in this thesis allows each LNMS to use its own methods and management protocols for collecting management information and uses the management information stored into the management database to express it in XML, providing the required interoperability functions between heterogeneous LNMSs.

The XML-gateway is implemented as a Java project that performs the following process in order to convert SQL data into an XML message. The process involves the following steps, as illustrated in figures 4.12, 4.13, 4.14, 4.15, 4.16 and 4.17:

1. Create a new document by using the standard Java API for XML Processing (JAXP) [JAXP], which provides the parsing and transformation of documents (figure 4.12).

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document xml = builder.newDocument();
```

Figure 4.12: Step 1

2. Retrieve the data from event_table of the management database (figure 4.13).

```
Statement st = conn.createStatement();
Set st = st.executeQuery("SELECT * from event_table");
```

Figure 4.13: Step 2

3. Store data in a document object. Once the data is successfully extracted from the LNMS database, it is stored in a temporary document. A method creates a row element (<row>…</row>) for each row of data, with each column represented as an element named after that column, and with the data itself as the content of the element (figure 4.14).

```
while (st.next()) {
      Element rw = xml.createElement("Row");
      results.appendChild(rw);
      for (int i = 1; i <= colCount; i++) {
      String columnName = rmd.getColumnName(i);
      Object value = st.getObject(i);
```

Figure 4.14: Step 3

4. Perform data mapping. Once the data is stored in the temporary document, DOM generator works on mapping the temporary document to a new XML structure. First, it retrieves the information on the root and row elements, and then retrieves the element

mappings. Each row of the data is analyzed and re-mapped to the new structure (figure 4.15).

```
public static String serialize(Document xml) throws
IOException {
      StringWriter writer = new StringWriter();
      OutputFormat format = new OutputFormat();
      format.setIndenting(true);
      XMLSerializer serializer = new XMLSerializer(writer,
format);
      serializer.serialize(xml);
      return writer.getBuffer().toString();
```

Figure 4.15: Step 4

5. Perform element mappings. This step determines what data is pulled from the temporary document and in what order (figure 4.16).

```
node.appendChild(xml.createTextNode(value.toString()));
row.appendChild(node);
```

Figure 4.16: Step 5

6. Add elements and data to the new XML document. The column count from the meta-data gives the quantity of the columns in each row element. The rmd.getColumnName() method gives the name of a given column. The value of the column object is accessed via the ResultSet.Metadata() method. An element is added for each column and is placed under its row (figure 4.17).

```
    Element node = xml.createElement(element);
    ResultSetMetaData rmd = st.getMetaData();
    return xml;
```

Figure 4.17: Step 6

A short version of an XML message that has been created by the XML-gateway is illustrated in figure 4.18 below:

```
<?xml version="1.0" encoding="UTF-8"?>

<Results xmlns=" http://esb.nms1">
<Row>
      <eventid>1</eventid>
      <eventuei> uei.opennms.org/reporting </eventuei>
      <eventtime>2010-03-16 12:10:50.0</eventtime>
      <eventhost>143.53.36.72</eventhost>
      <eventsource>ProCurve J8697A Switch </eventsource>
      <eventdpname>undefined</eventdpname>
      <eventcreatetime>2010-03-16 12:10:51.421</eventcreatetime>
      <eventdescr> High alert for interface 143.53.36.72 ;</eventdescr>
      <eventlogmsg>The Accuracy level is : 99%</eventlogmsg>
      <eventseverity>3</eventseverity>
      <eventlog>Y</eventlog>
      <eventdisplay>N</eventdisplay>
   </Row>
<Row>
      <eventid>2</eventid>
      <eventuei> uei.opennms.org/reporting </eventuei>
      <eventtime>2010-03-16 12:10:51.0</eventtime>
      <eventhost>127.0.0.1</eventhost>
      <eventsource>Host </eventsource>
      <eventdpname>Lab-PC</eventdpname>
      <eventcreatetime>2010-03-16 12:10:51.463</eventcreatetime>
      <eventdescr> IP packet loss 0</eventdescr>
      <eventlogmsg>The Total IP packet loss is 0</eventlogmsg>
      <eventseverity>1</eventseverity>
      <eventlog>Y</eventlog>
      <eventdisplay>N</eventdisplay>
   </Row>
<Row>
      <eventid>7</eventid>
      <eventuei> uei.opennms.org/reporting </eventuei>
      <eventtime>2010-03-16 15:09:23.53</eventtime>
      <eventhost>127.0.0.1</eventhost>
      <eventsource>Host </eventsource>
      <eventdpname>Lab-PC</eventdpname>
      <eventcreatetime>2010-03-16 15:09:23.124</eventcreatetime>
      <eventdescr> hrSWRStatus.200 = INTEGER: running(1)</eventdescr>
      <eventlogmsg>Application 200 is running</eventlogmsg>
      <eventseverity>1</eventseverity>
      <eventlog>Y</eventlog>
      <eventdisplay>N</eventdisplay>
   </Row>
<Row>
</Row>
</Results>
```

Figure 4.18: xml management message

The output of the XML-gateway is XML document-based, which in turn can be processed by the Network Management Middleware. The XML-based message contains information related to faults and performance measurements from multiple components residing on the network infrastructure. The interaction between LNMS and Middleware Layer is kept to the minimum by exchanging large amount of management information per message exchange. As explained in earlier chapters, multiple invocations results in overhead to the network. In other architectures such as CORBA-based architectures, the applications are required to exchange small amount of functionality due to the use of objects. For example, each interaction between LNMS and another application would contain one fault or one event per interaction. In contrast, in the XML-gateway, a single XML message contains information from multiple devices. In the short version of the XML management message depicted in figure 4.18, three events are illustrated.

The XML-gateway is implemented as a Web Service. A WSDL service contract has been created in order to define the description of its interfaces. Apache Axis [AXIS] has been used as the SOAP server. Axis provides the SOAP communication protocol and supports WSDL service contracts. Axis uses Tomcat application server [TOMCAT] as a container in order to expose Web Services over the internet and supports SOAP version 1.1 as a lightweight protocol for communication and WSDL 1.1 for the description of the Web Services Interfaces [AXIS]. The following figure (figure 4.19) illustrates the WSDL service contract of the XML-gateway. The WSDL describes the parameters of the XML-gateway such as the

operations names and types (in, out), the message names, binding name and address and the SOAP address.

```
wsdl:types>
   <xsd:schema targetNamespace="http://www.management.org/XML-gateway/">
    <xsd:element name="Get or Set">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="in" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="Response">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="out" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:schema>
  </wsdl:types>
  <wsdl:message name="OperationRequest">
   <wsdl:part element="tns:Operation" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="OperationResponse">
   <wsdl:part element="tns:OperationResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="XML-gateway">
   <wsdl:operation name="Operation">
    <wsdl:input message="tns:OperationRequest"/>
    <wsdl:output message="tns:OperationResponse"/>
   </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="XML-gatewaySOAP" type="tns:XML-gateway">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="Operation">
    <soap:operation soapAction="http://www.management.org/XML-gateway/Operation"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="XML-gateway">
   <wsdl:port binding="tns:XML-gatewaySOAP" name="XML-gatewaySOAP">
    <soap:address location="http://www.management.org/"/>
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 4.19: XML-gateway WSDL file

Web Service Network Management Applications can acquire management information from the LNMS. A test case has been created in the eclipse IDE [ECLIPSE] in order to act as Web Service Network Management Application. Figures 4.20 and 4.21 illustrate the input statements that are used and the results of each statement. In figure 4.20, the Web Service application that acts as a client requests information concerning the network nodes of the managed network. In figure 4.21, the Web Service application requests the services that are running on a server. This test case shows that the management information can be retrieved over the internet and the applications that consume this management information can be located on different hosts. As a result, the management architecture can be loosely coupled and distributed.

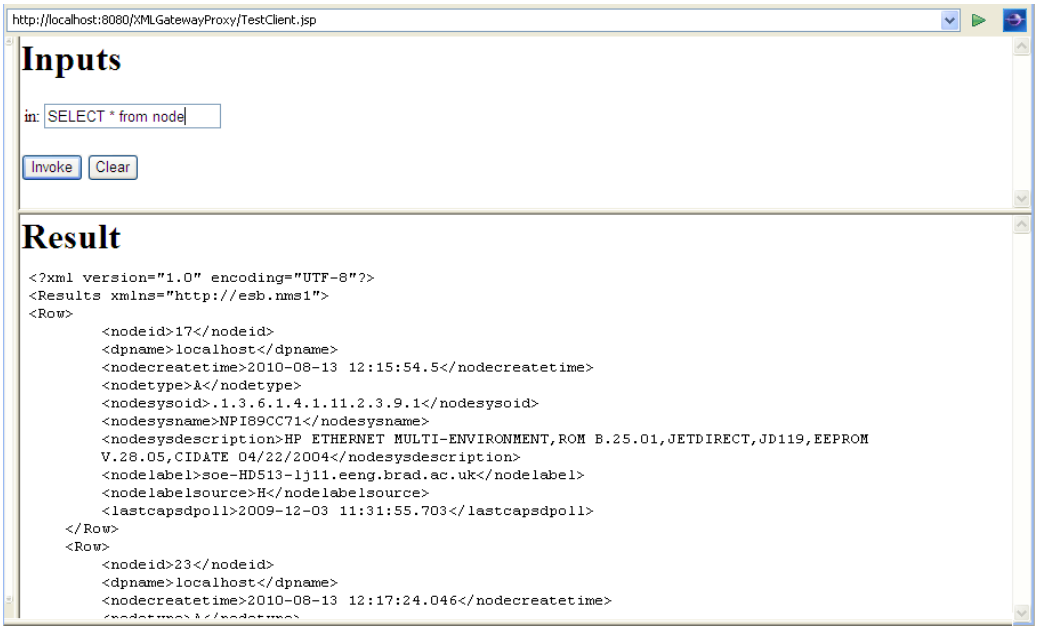


Figure 4.20: Web Service application requesting list of the network devices from the LNMS

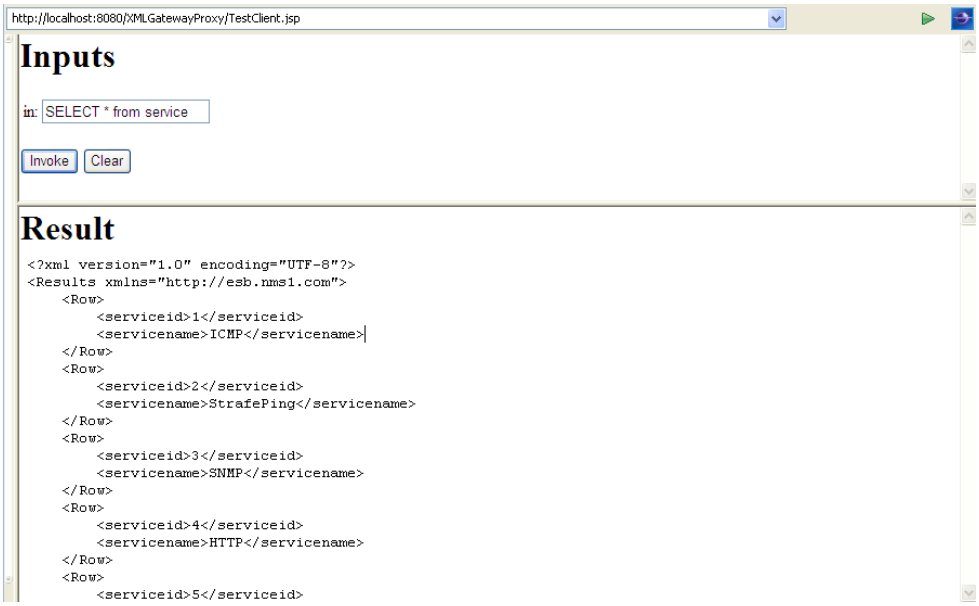


Figure 4.21: Web Service application requests the server's running process from the LNMS

The XML-gateway can be embedded in the Network Management Middleware layer instead of the LNMS. This ability is beneficial for legacy LNMSs that are not open source software or do not have integration capabilities. In this case, the XML-gateway will perform remote SQL-based calls instead of SOAP calls. SOAP calls provide more advanced security mechanisms (i.e. WS-Security) compared to remote SQL-based calls. Even though the remote SQL-based calls are Remote Procedure Calls, the security is an additional mechanism that has to be implemented for authentication and authorization.

The XML-gateway has been embedded in another open-source NMS that will be used later in chapters 5 and 6 as a second LNMS. NINO network management system [NINO] has been used for sending management information to the Network Management Middleware Layer. NINO is using different data representations compared to the LNMS presented in this

chapter. The two LNMSs are considered heterogeneous since their Core Process Logic as well as their management databases are different. NINO's representation of management information is being transformed into XML by the XML-gateway as depicted in figure 4.22. The representation of the management information is different compared to the management information in figure 4.18.

```
<?xml version="1.0" encoding="UTF-8"?>

<Results xmlns=" http://esb.nms2">
<Row>
      <id>2</id>
      <host>143.53.36.43 </host>
      <name>Intel<R> Gigabit Network connection</name>
      <location>Lab_Horton</location>
      <devicetype>Server </devicetype>
      <statuscheck>yes</statuscheck>
      <collect>16-03-2010 10:10:50</collect>
      <severityid>21</severityid>
      <severity>High</severity>
      <operatingsystem>x86 Family 6 Windows version 5.1</operatingsystem>
      <description>Interface: 143.53.36.43 is down</description>
</Row>
<Row>
<id>2</id>
      <host>143.53.36.43 </host>
      <name>Intel<R> Microsoft Service</name>
      <location>Lab_Horton</location>
      <devicetype>Server </devicetype>
      <statuscheck>yes</statuscheck>
      <collect>16-03-2010 10:10:52</collect>
      <severityid>13</severityid>
      <severity>Notification</severity>
      <operatingsystem>x86 Family 6 Windows version 5.1 </operatingsystem>
      <description>Interface: MySQL activated</description>
   </Row>
<Row>
</Row>
</Results>
```

Figure 4.22: XML-gateway output management information acquired from NINO LNMS

4.4.6 Performance Management

The performance of a network domain can be assessed by interrogating all the NEs in that domain. The implemented Event Control Unit performs

calculations regarding performance management. Polling is a frequent operation in the LNMS as there are several object values that require constant monitoring. MIB variables are not a representative indicator of the network's state, thus, an aggregation of multiple variables is required [GOLD93]. The following section presents the calculations that have been performed by the Control Unit. These performance functions are recommended by the ITU-T as well as by other RFCs [KAVA00], [WALD95], [M.2301].

4.4.6.1 Performance management Parameters

The following table (table 4-3) lists the messages that are required for monitoring performance functions. The messages as well as the values have been defined and stored in the agent's MIB data store. Each message has its own Object Identifier (OID) value in the MIB tree. This means that whenever a message request is received by the agent, the message name is mapped into the OID equivalent and the agent reads the stored value from the MIB. There are different fixed numbers of datatypes which are used by the values of OIDs [McCL99]. These value types have been defined by the SMI and the ones that have been used for the performance variables are TimeTick, Counter32 and Gauge32. TimeTick represents an unsigned integer that represents the time. The range of the TimeTick is 0 to 2^{32} in hundredths of a second (centisecond). Counter32 represents a non-negative integer, which monotonically increases until it reaches a maximum value of 4294967295. When the counter reaches the maximum value, then it starts increasing again from zero. Gauge32

represents an unsigned integer, which may increase or decrease, but cannot exceed a maximum number.

Table 4-2: The essential Variables required for performance management

Variable name	Object Identifier	Description	Value Type
SysUpTime	1.3.6.1.2.1.1.3	The time since the managed node was last re-initialized (measured in hundred of a second)	TimeTicks
ifInErrors	1.3.6.1.2.1.2.2.1.14	the no. of inbound packets with an error	Counter32
ifOutErrors	1.3.6.1.2.1.2.2.1.20	the no. of outbound packets with an error	Counter32
ifInUcastPkts	1.3.6.1.2.1.2.2.1.12	the count of inbound unicast packets	Counter32
ifOutUcastPkts	1.3.6.1.2.1.2.2.1.17	the count of outbound unicast packets	Counter32
ifInNUcastPkts	1.3.6.1.2.1.2.2.1.11	the count of inbound non-unicast packets (multicast and broadcast)	Counter32
ifOutNUcastPkts	1.3.6.1.2.1.2.2.1.18	the total outbound number of non-unicast packets (multicast and broadcast)	Counter32
IfInOctets	1.3.6.1.2.1.2.2.1.10	Total number of octets received on an interface, including framing characters	Counter32
ifOutOctets	1.3.6.1.2.1.2.2.1.16	Total number of octets transmitted out of an interface, including framing characters	Counter32
ifSpeed	1.3.6.1.2.1.2.2.1.5	Interface's current bandwidth in bits per second.	Gauge32
ifInDiscards	1.3.6.1.2.1.2.2.1.13	Number of inbound bytes that have been discarded to free up the buffer space.	Counter32
ifOutDiscards	1.3.6.1.2.1.2.2.1.19	Number of outbound bytes that have been discarded to free up the buffer space.	Counter32
ifInUnknownProtos	1.3.6.1.2.1.2.2.1.15	For Packet-oriented interface, the number of packets received via an interface which were discarded because of an unknown or unsupported protocol.	Counter32
ipOutDiscards	1.3.6.1.2.1.4.11	The number of output IP	Counter32

		packets for which no problem was encountered to prevent their transmission to their destination, but which discarded (i.e. lack of buffer space)	
ipOutNoRoutes	1.3.6.1.2.1.4.11	The number of IP packets discarded because no route could be found to transmit them to their destination	Counter32
ipFragFails	1.3.6.1.2.1.4.18	The number of IP packets that have been discarded because they needed to be fragmented at this node but could not be (e.g. their Don't Fragment flag was set)	Counter32
ipOutRequests	1.3.6.1.2.1.4.10	Total number of IP packets which local IP user-protocols supplied to IP in requests for transmission	Counter32
ipForwDatagrams	1.3.6.1.2.1.4.6	The number of input packets that request to find a route to forward them to their final destination.	Counter32

4.4.6.1.1 Total IP received packets calculation

Total IP packets received (TR_IP) is a count of the number of packets received at a NE's interface. The total number of packets received across an interface is given by the sum of all inbound packets. In more detail, the inbound packets consist of the following packets: unicast packets, non-unicast packets, discarded packets, packets with errors and packets that have been discarded for unknown reasons.

The above mentioned packets can be acquired by the following SNMP OID variables:

- ifInUcastPkts;
- ifInNUcastPkts;

- ifInDiscards;
- ifInErrors;
- ifInUnknownProtos.

The equation for calculating the total IP packets received by a NE's interface is depicted in equation (4.1):

$$TR\_IP = ifInUcastPkts + ifInNUcastPkts + ifInDiscards + ifInErrors + ifInUnknownpotos \quad (4.1)$$

4.4.6.1.2 Total IP transmitted packets

The total IP packets transmitted (TT_IP) through an interface in a NE is given by the sum of (formula 4.2):

- ifOutUcastPkts;
- ifOutNUcastPkts.

$$TT\_IP = ifOutUcastPkts + ifOutNUcastPkts \quad (4.2)$$

The number of successfully transmitted packets (TT_OK) over the link is given by the following equation (formula 4.3):

$$TT\_OK = TT\_IP - (ifOutDiscards + ifOutErrors) \quad (4.3)$$

4.4.6.1.3 IP Packet Loss Ratio

The IP packet Loss Ratio (IPLR) is the ratio of total lost IP packet outcomes to total transmitted IP packets. The variables that are required to be aggregated in order to calculate the IPLR are:

- ifInUcastPkts,
- ifInNUcastPkts,
- ifOutUcastPkts and

- ifOutNUcastPkts.

The following formula (formula 4.4) calculates the IP Packet Loss Ratio:

$$IPLR = \frac{(ifInUcastPkts + ifInNUcastPkts)}{TT\_IP} \quad (4.4)$$

4.4.6.1.4 Error Rate and Accuracy

Accuracy determines the traffic of an interface that does not result in error and is expressed in terms of percentage, comparing the success rate to total packet rate over a period of time. First, the Error Rate (ER) needs to be calculated and then the accuracy can be determined. For instance, if three of every 100 packets result in error, the ER would be 3% and the accuracy would be 97%. In order to calculate the ER and the accuracy formulas, the delta (Δ) function is used. This means that instead of one two poll cycles are required and the difference between the two cycles is calculated. In formulas 4.6, 4.9 and 4.10, the variable t indicates the polling cycle and the variable x indicates the previous polling time at polling cycle.

With earlier network technologies, a certain level of errors was acceptable. However, today's high-speed networks are considerably more accurate and the ERs are close to zero, unless there is an actual problem. The most common causes of interface errors are [CISC07]:

- Electrical interference.
- Out-of-specification wiring.

- Faulty hardware or software.
- Incorrect configuration.

The ER is expressed as a percentage. The formula for determining the ER of an interface is given by formula (4.5 or 4.6):

$$ER = \frac{\Delta ifInErrors * 100}{(\Delta ifInUcastPkts + \Delta ifInNUcastPkts)}\% \quad (4.5)$$

or

$$ER = \frac{(ifInErrors_{(x+t)} - ifInErrors_{(x)} * 100}{(ifInUcastPkts_{(x+t)} - ifInUcastPkts_{(x)}) + (ifInNUcastPkts_{(x+t)} - ifInNUcastPkts_{(x)})} \quad (4.6)$$

The outbound errors are not considered in the ER formula. The reason is that the NE should never place packets with errors on the network, and the outbound interface ERs should never increase. Thus, inbound traffic and errors are the only measures of interest for interface errors.

The Accuracy Rate (AR) is expressed as a percentage. The formula for accuracy calculates the ER of the interface and subtracts it from 100. The formula for determining the accuracy of an interface is given by the following formula (formula 4.7 or 4.8):

$$AR = 100 - \frac{\Delta ifInErrors * 100}{(\Delta ifInUcastPkts + \Delta ifInNUcastPkts)} \quad (4.7)$$

or

$$AR = 100 - ER \quad (4.8)$$

4.4.6.1.5 Utilization of an interface

Utilization measures the use of a particular resource over a time period. It is expressed in the form of a percentage in which the usage of a resource is compared to its maximum operational capacity. Utilization is the principle for determining how full the network pipes (links) are. Through utilization measurement, the congestion throughout the network can be identified.

For serial link utilization rate, there are two possible transmission modes: The half-duplex mode and the full-duplex mode. Shared LAN connections are usually based on half-duplex mode, mainly because connection detection requires that a network interface listens before it transmits. WAN connections are typically full-duplex mode allowing the communication in both directions simultaneously. Land-line telephone networks are full-duplex, since they allow both callers to speak and be heard at the same time. If the utilization rate is over 90%, the network is regarded as overloaded. It is not serious for the utilization rate to exceed 90% temporarily, but if the average utilization rate is over 90%, the entire network is overloaded and usually needs to be reconstructed or equipment need to be upgraded. To calculate the utilization rate, a single poll will not give any useful information. Sampling the interface over a time interval can show the traffic in and out of the interface over a period of time. In this case, the sampling will require two polling cycles. Furthermore, the values acquired from the agent are octets meaning that the each octet is one byte. So the formulas need to be multiplied by 8, eight times to get the rates in bit per second. Each interface contains a variable value that

indicates the mode of operation. This variable cannot be modified due to the fact that the access status is read-only. The following table (table 4-4) describes the DuplexStatus variable that is an essential parameter for indicating the interface's mode in order to be able to calculate the utilization rate according to the specified formula.

Table 4-3: DuplexStatus variable

Variable name	Object Identifier	Description and value
DuplexStatus	1.3.6.1.2.1.10.7.2.1.19	The value of the OID indicates the status of the interface's mode. halfduplex(1) indicates the Half-duplex mode and The fullduplex(2) indicates the full-duplex mode.

The Half-Duplex Utilization (HDU) is expressed as a percentage. The formula for the HDU of an interface is calculated by adding the total sent bits and the total received bits divided by the bandwidth. The following formula depicts the utilization of a half-duplex interface (formula 4.9):

$$HD = \frac{[(ifInOctets_{(x+t)} - ifInOctets_{(x)}) + (ifOutOctets_{(x+t)} - ifOutOctets_{(x)})]*8*100}{(sysUpTime_{(x+t)} - sysUpTime_{(x)})*ifSpeed} \quad (4.9)$$

For a Full-Duplex Utilization (FDU) media, the utilization formula calculates the larger value of the input and output octets (bytes) and generates the utilization percentage. An example for full-duplex connection is the T-1 serial connection. In this case, the line speed is 1.544 Mbps. This means that a T-1 interface can both receive and transmit 1.544 Mbps for a combined bandwidth of 3.088Mbps. The utilization of a full-duplex interface can be calculated using equation (4.10):

$$FD = \frac{Max[(ifInOctets_{(x+t)} - ifInOctets_{(x)}), (ifOutOctets_{(x+t)} - ifOutOctets_{(x)})]*8*100}{(sysUpTime_{(x+t)} - sysUpTime_{(x)})*ifSpeed} \quad (4.10)$$

4.4.6.1.6 IP output datagrams discard rate

The Discard Rate (DR) defines the IP output datagrams discarded over the total number of datagrams sent during a specific time interval. DR is expressed as a percentage by combining five MIB variable objects. Formula 4.11 identifies the IP output datagrams DR:

$$DR = \frac{(\Delta ipOutDiscards + \Delta ipOutNoRoutes + \Delta ipFragFails)*100}{\Delta ipOut\ \mathrm{Re}quests + \Delta ipForwDatagrams} \quad (4.11)$$

4.4.6.2 Performance function process flows

4.4.6.2.1 Initialisation

The Event Correlator component of the Control Unit is responsible for calculating the aforementioned formulas. Through the event handling, the Event Correlation component initializes the Manager Poller and requests specific OID variables to be acquired by a specific IP address when carrying some performance functions. Figure 4.23 shows the software code for implementing the agent's target address (143.53.36.23) and the socket number (161) are specified.

```
Address targetAddress =
GenericAddress.parse("udp:143.53.36.23/161");
   TransportMapping transport = new DefaultUdpTransportMapping();
   snmp = new Snmp(transport);
transport.listen();
```

Figure 4.23: defining the agent's address and UDP port number

The Manager Poller implements a time function specifying how often the agents should be polled, as shown in Figure 4.24 below.

```
private Timer timer = new Timer(true);
    public void schedule(TimerTask task, int milsec) {
      timer.schedule(task, 10000);
    }
```

Figure 4.24: Timer method

The Poller sends an SNMP BULK request including the required variables for calculating each formula. The BULK request is used to acquire multiple variables with just one request. If the connection fails for a reason, then the poller initiates a new request. Once the connection between the LNMS and the agent is established, the Event-Handler sends the SNMP message to the agent. The following code in figure 4.25 illustrates the SNMP GetBulk operation requesting: ifInUcastPkts ifInNUcastPkts, ifInDiscards, ifInErrors, and ifInUnknownProtos from the agent. The target address has been specified in the targetAddress variable.

```
PDU request = new PDU();
request.setType(PDU.GETBULK);
request.add(new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.11")));
request.add(new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.12")));
request.add(new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.13")));
request.add(new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.14")));
request.add(new VariableBinding(new OID("1.3.6.1.2.1.2.2.1.15")));
ResponseEvent responseEvent = snmp.send(pdu, target);
```

Figure 4.25: OID requests

The Event-Handler waits for the agents to respond within a predefined time interval. If the agent cannot respond for a reason (i.e. the agent is deactivated), within 20 sec, the session fails and an error message is created and sent to the event schedule and archive component (figure 4.26).

```
target.setTimeout(20000);//20*1000ms
target.setRetries(5);
public long getRetryTimeout(int Retries, long targetTimeout); {
    return targetTimeout;
  }
```

Figure 4.26: Method for Time interval and number of retries

If the Control Unit receives the SNMP response from the agent within 20 seconds, the respond values are consumed by the Event Correlation component.

4.4.6.2.2 Process flow for TT_IP, TR_IP, TT_OK, IPLR Measurements

Four different options can be chosen, as shown in figure 4.27. The first option shows the process flow of the total received packets in the agent's interface (TR_IP). The second option illustrates the process of the number of packets transmitted from the agent's interface (TT_IP). The third option shows the packets that have been successfully transmitted over the agent's interface (TT_OK) and the fourth option shows the packet loss of the agent's interface (IPLR).

When the Event Correlator component receives the agent's values, it calculates them based on equations (4.1) to(4.4). A process of storing the calculated information takes place after the calculation process, where the result of each calculation is stored in a database table (Event_table). Each result is accompanied by the IP address of the interface, the specific number of the interface (in case of multiple interfaces) and a timestamp of every insertion.

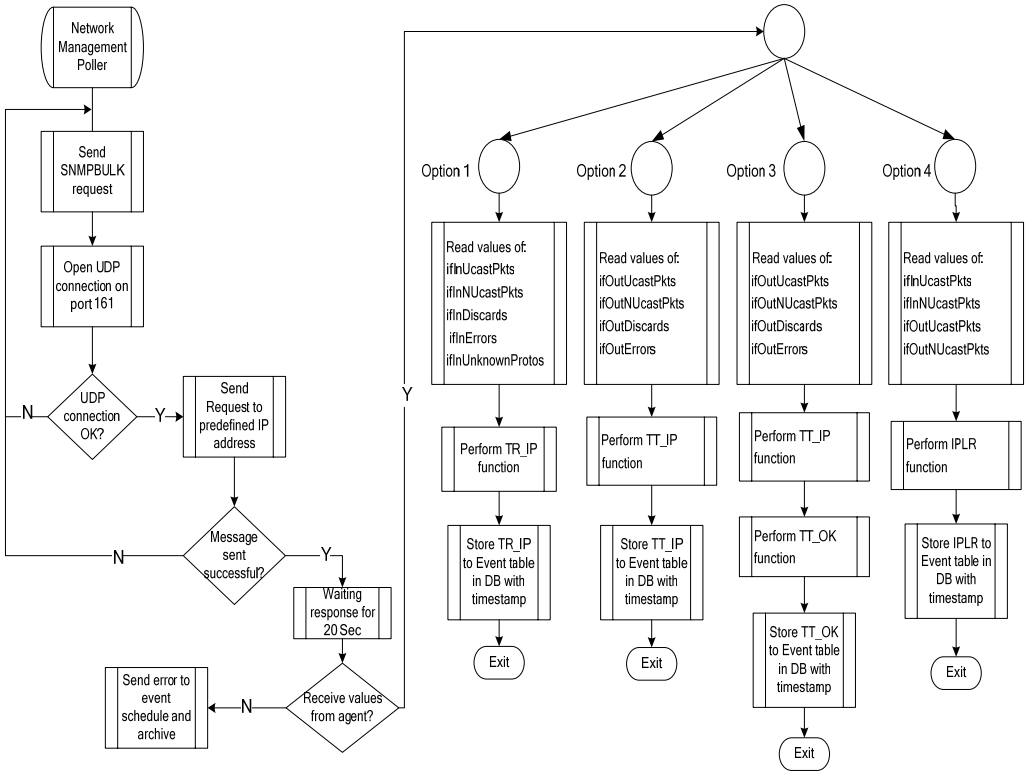


Figure 4.27: LNMS flow diagram for performing TR_IP, TT_IP, TT_OK and IPLR

4.4.6.2.3 Process flow for Error Rate and Accuracy Rate Measurement

Figure 4.28 also includes the process flow for measuring the ER and the AR of the NE's interface.

The calculation of the ER in the NE's interface takes place when the Event Correlator component reads the agent's values. Due to the delta function calculation, the ER function requires two poll cycles to be sampled. The first cycle (Cycle 1) involves storing the agent's values to the database table (Error Rate_table). The timer in the Network Manager Poller has been set to initiate another poll after one minute, which is when the poller sends again a BULK request to the agent following the same process as

before, but this time the Event Correlation component executes the second cycle of the ER process flow (Cycle 2).

At this point, the Event Correlation component reads the new values acquired by the agent to perform the ER function, as well as the previous stored values from the Error Rate_table. The next process is the calculation of the ER. In this step, the agent's values ifInUcastPkts, ifInNUcastPkts and inInErrors are calculated according to equation (4.6). An 'if' function has been created in order to initiate an alert for ER values that are higher than 2 (2%). In this decision point, if the ER value exceeds 2%, an alert is created with a printed value *"High alert for Interface" +IP ""the error rate is" +ER "Severity level" +severity "on date and time" +timestamp*. This information is stored in the event table in the LNMS's database. If not, the ER is stored in the event table with the calculated value, the IP address of the interface and a timestamp. A process of deleting the temporary values stored in the Error Rate_table takes place in both cases before the application exits. This process has been implemented due to the fact that the ER function will be executed continuously, thus the variables and the values of the first poll stored in the Error Rate_table need to be constantly updated. Hence, the temporary management information gathered and stored in the Error Rate_table needs to be deleted after every process execution.

In the accuracy process flow, the Event Correlation component reads the agent's values. Similar to the ER process, the Event Correlation component samples twice the agent. In Cycle 1, the values are stored in the Accuracy_table, and the algorithm exits. In Cycle 2, the Event

Correlation component first reads the agent's values, next, reads the values stored in the Accuracy_table and uses these values to feed the next process, which performs the accuracy calculation (function (6)). If the accuracy has value less than 98 (98%), an alert is created with a printed value *"High alert for Interface" +IP "the Accuracy is" +AR "Severity level" +severity "on date and time" +timestamp*. If the value is higher than 98 (98%), then it is stored in the database's event table accompanied with the IP address of the NE's interface and a timestamp. The temporary stored data is deleted from the Accuracy_table before the processes can exit. The ER and accuracy functions are performed separately, because it allows the user to decide if both functions are going to be performed.

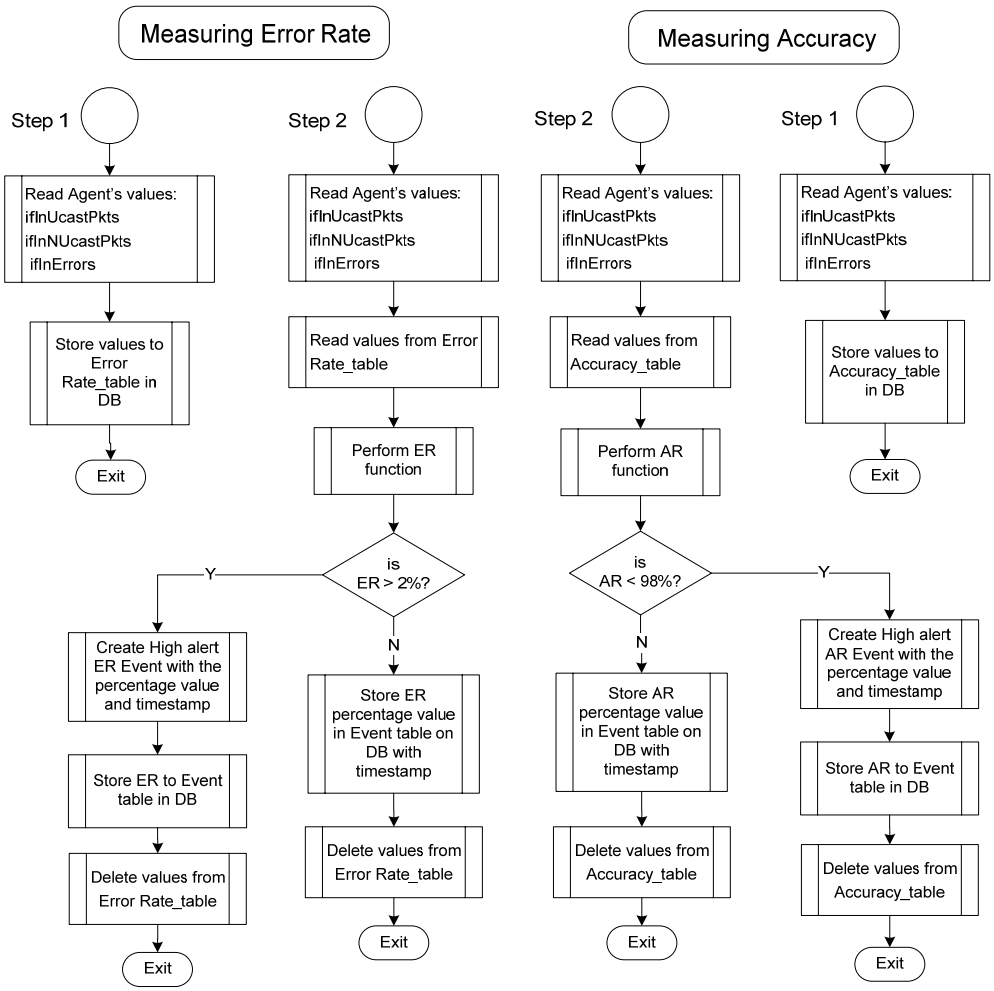


Figure 4.28: Process flow for performing ER and AR functions

4.4.6.2.4 Process flow for Discard Rate Measurement

Figure 4.29 demonstrates the process for measuring the DR and the utilization of the NE's interface. In the DR process, the Event Correlation component reads the agent's values and stores the values in the DR_table (Step 1). Next, the Manager Poller, after a predetermined time period, initiates a second SNMP BULK request for requesting management information from the agent. The Event Correlation component (Step 2)

reads the agent's values and feed the values to the next process that performs the DR function (formula 4.11). The output of the process is stored in the database table 'event table' including the IP address of the NE's interface and a timestamp. Before the application can stop, it deletes the data stored in the specific rows of the DR_table.

4.4.6.2.5 Process flow for Utilisation Rate Measurement

As mentioned above, the utilization rate of an interface has two options that are calculated according to two equations: the half-duplex and the full-duplex equations. The Event Correlation component reads the agent's values and stores the values in the Utilization_table (Step 1). Next, the Manager Poller, after a predetermined time period, polls again the agent requesting the same variables (Step 2). The most important value of agent's retrieval is the DuplexStatus. The Event Correlation component reads the agent's values. If the value of the DuplexStatus is fullDuplex(2), meaning that the interface works as full-duplex interface, then the event correlation component performs the FD function (formula 4.10).

If not, it performs the HD function (formula 4.9). In the full-duplex process, if the calculated value exceeds 0.9 (90%), then an alert is created with a printed value *"High alert for Interface" +IP "the Utilization is" +FD "Severity level" +severity "on date and time" +timestamp*. This information is stored in the LNMS's database (event table) and the application deletes the data stored in the Utilization_table before exiting. If the value of the FD function is less than 0.9 (90%), the value is stored in the database's event table accompanied with the IP address of the NE's interface and a timestamp.

Again, the application deletes the data stored in the Utilization_table before it exits.

In the half-duplex option, an ‘if’ function creates a decision point, where if the calculated value is higher than 0.9 (90%), then an alert is created with a printed value *“High alert for Interface” +IP “the Utilization is” +HD “Severity level” +severity “on date and time” +timestamp*. This information is stored in the LNMS’s event table and before the application exits, it deletes the data stored in the Utilization_table. If the HD function gives a value that is less than 0.9 (90%), the value is stored in the database’s event table accompanied with the IP address of the NE’s interface and a timestamp. The application deletes the data stored in the Utilization_table before it exits.

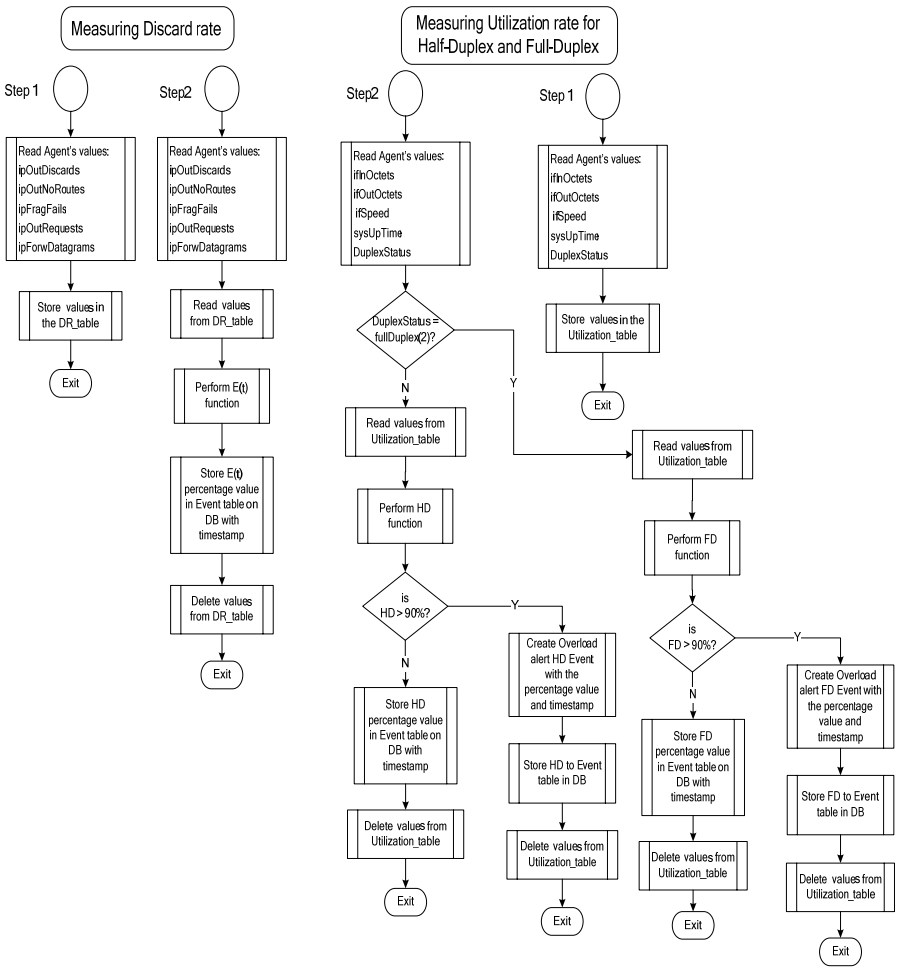


Figure 4.29: Process flow for performing DR and HD or FD functions

4.4.6.3 Performance Information Retrieval

The agent can provide information concerning performance measurements. These message requests can be handled individually by the agent or can be sent altogether at once. To minimize the traffic in the network, it is reasonable to have one request that contains all the appropriate performance requests, instead of performing multiple request-response operations. Figure 4.30 depicts the interaction between LNMS and the agent when using the BULK request for acquiring multiple management information from the agent.

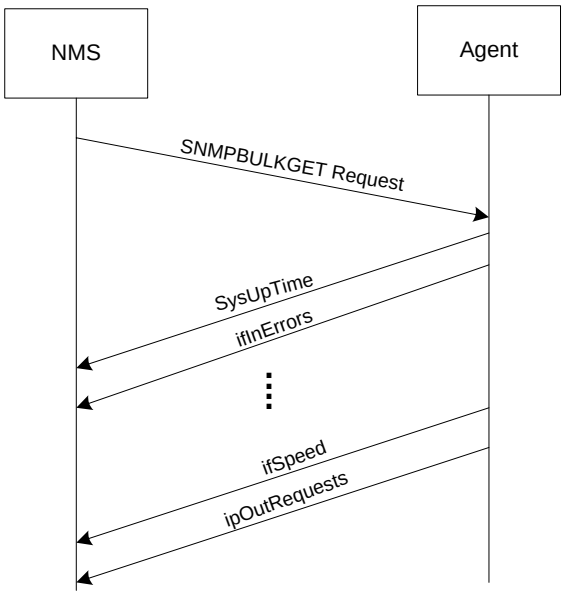


Figure 4.30: Agent's multiple responses

The output of the SNMPBULKGET operation acquiring performance measurement information is shown in figure 4.31.

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (10395090) 1 day, 4:52:30.90
IF-MIB::ifInErrors.1 = Counter32: 0
IF-MIB::ifInUcastPkts.1 = Counter32: 64966
IF-MIB::ifInNUcastPkts.1 = Counter32: 0
IF-MIB::ifInOctets.1 = Counter32: 2872653
IF-MIB::ifOutOctets.1 = Counter32: 2872653
IF-MIB::ifSpeed.1 = Gauge32: 10000000
IP-MIB::ipOutDiscards.0 = Counter32: 0
IP-MIB::ipOutNoRoutes.0 = Counter32: 0
IP-MIB::ipFragFails.0 = Counter32: 0
IP-MIB::ipOutRequests.0 = Counter32: 124205
IP-MIB::ipForwDatagrams.0 = Counter32: 0
```

Figure 4.31: Agent's response messages

4.4.7 Fault and Configuration Management

4.4.7.1 Fault and Configuration Management Process

For fault and configuration management, the process is simpler compared to performance management. Fault management requires system's status

only, whereas performance management measures different aspects of the network.

Table 4-4 presents the information that is required in order to perform fault management functions. Those variables indicate the operational status of the element's network interfaces, hardware components and software components and all have read-write mode. This means that the values not only can be read but they can also be modified. Hence, the LNMS is able to perform configuration management by configuring the values thus, changing the behaviour of the NE.

Table 4-4: Variables indicating faults in network elements

Variable name	Object Identifier	Description and Value
ifadminStatus	1.3.6.1.2.1.2.2.1.7	The state of an interface in the network node. • Up(1): the interface is up and running. • Down(2) : the interface is down and • Testing(3) no operational packets can be passed through this interface.
hrDeviceStatus	1.3.6.1.2.1.25.3.2.1.5	The operational status of a hardware component of the network node. • unknown(1): the current state of the device is unknown. • running(2): the device is up and running and that no unusual error conditions are known • warning(3): an unusual error condition by the operational software (e.g., a disk device driver) but that the device is still 'operational'. • testing(4):, the device is not available for use because it is in the testing state. • down(5): device is not available for any use.

hrSwRunStatus	1.3.6.1.2.1.25.4.2.1.7	the status of software running in the network node. • Running(1) the software is running. • Runnable(2) the software is waiting for resources (i.e. waiting for memory or CPU resources). • Runnable(3) the software is loaded but is waiting for an event to start the process, • invalid(4) the software is not running.

Fault management variables described in the table 4-5 are collected by activating the Event Correlation component, which in turn initiates a process to acquire the status of a NE. The Network Manager Poller initiates an SNMP session with a SNMP GET requests and stores these values status to the LNMS's database. For configuration management, the Poller uses the SNMP SET operation in order to modify the status variables described in table 4-4.

4.4.7.2 Status information retrieval

The status information indicates the current condition of the NE. Figure 4.32 illustrates the hardware status information as well as the agent, who is capable of terminating or starting a hardware resource on the server. In figure 4.32, the status of hardware resources with OID values 1.3.6.1.2.1.25.3.2.1.5.9 (Intel processor CPU_1) and 1.3.6.1.2.1.25.3.2.1.5.10 (Intel processor CPU_2) are running properly. If the value's current state changes, the LNMS will create a fault indication. Moreover, the LNMS can perform an SNMP SET operation in order to force the agent to change the state of the components.

```
HOST-RESOURCES-MIB::hrDeviceDescr.9 = STRING: Intel
HOST-RESOURCES-MIB::hrDeviceDescr.10 = STRING: Intel
HOST-RESOURCES-MIB::hrDeviceDescr.11 = STRING: MS TCP Loopback interface
HOST-RESOURCES-MIB::hrDeviceDescr.12 = STRING: Intel(R) PRO/1000 EB Network Connection with I/O Acceleration #2
HOST-RESOURCES-MIB::hrDeviceDescr.13 = STRING: Intel(R) PRO/1000 EB Network Connection with I/O Acceleration -
HOST-RESOURCES-MIB::hrDeviceDescr.14 = STRING: D:\
HOST-RESOURCES-MIB::hrDeviceDescr.15 = STRING: Fixed Disk
HOST-RESOURCES-MIB::hrDeviceDescr.16 = STRING: IBM enhanced (101- or 102-key) keyboard, Subtype=(0)
HOST-RESOURCES-MIB::hrDeviceDescr.17 = STRING: 3-Buttons  (with wheel)
HOST-RESOURCES-MIB::hrDeviceDescr.18 = STRING: COM1:
HOST-RESOURCES-MIB::hrDeviceDescr.19 = STRING: COM2:
HOST-RESOURCES-MIB::hrDeviceStatus.9 = INTEGER: running(2)
HOST-RESOURCES-MIB::hrDeviceStatus.10 = INTEGER: running(2)
HOST-RESOURCES-MIB::hrDeviceStatus.11 = INTEGER: unknown(1)
HOST-RESOURCES-MIB::hrDeviceStatus.12 = INTEGER: unknown(1)
HOST-RESOURCES-MIB::hrDeviceStatus.13 = INTEGER: unknown(1)
HOST-RESOURCES-MIB::hrDeviceStatus.14 = INTEGER: unknown(1)
HOST-RESOURCES-MIB::hrDeviceStatus.15 = INTEGER: running(2)
HOST-RESOURCES-MIB::hrDeviceStatus.16 = INTEGER: running(2)
HOST-RESOURCES-MIB::hrDeviceStatus.17 = INTEGER: running(2)
HOST-RESOURCES-MIB::hrDeviceStatus.18 = INTEGER: unknown(1)
HOST-RESOURCES-MIB::hrDeviceStatus.19 = INTEGER: unknown(1)
```

Figure 4.32: Server's hardware resources retrieved by the agent

Figure 4.33 demonstrates the software resources that are installed in the server. The agent accessed the MIB Data Store and retrieved the values of the softwares running on the server. Same as before, the agent has the rights to modify the states of the softwares.

```
HOST-RESOURCES-MIB::hrSWRunStatus.1 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.4 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.200 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.224 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.236 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.328 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.356 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.416 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.460 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.580 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.632 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.768 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.804 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunStatus.912 = INTEGER: running(1)
HOST-RESOURCES-MIB::hrSWRunName.1 = STRING: "System Idle Process"
HOST-RESOURCES-MIB::hrSWRunName.4 = STRING: "System"
HOST-RESOURCES-MIB::hrSWRunName.200 = STRING: "ctfmon.exe"
HOST-RESOURCES-MIB::hrSWRunName.224 = STRING: "svchost.exe"
HOST-RESOURCES-MIB::hrSWRunName.236 = STRING: "wlcomm.exe"
HOST-RESOURCES-MIB::hrSWRunName.328 = STRING: "pctsSvc.exe"
HOST-RESOURCES-MIB::hrSWRunName.356 = STRING: "svchost.exe"
HOST-RESOURCES-MIB::hrSWRunName.416 = STRING: "rundll32.exe"
HOST-RESOURCES-MIB::hrSWRunName.460 = STRING: "spoolsv.exe"
HOST-RESOURCES-MIB::hrSWRunName.580 = STRING: "svchost.exe"
HOST-RESOURCES-MIB::hrSWRunName.632 = STRING: "msnmsgr.exe"
HOST-RESOURCES-MIB::hrSWRunName.768 = STRING: "cisvc.exe"
HOST-RESOURCES-MIB::hrSWRunName.804 = STRING: "jusched.exe"
HOST-RESOURCES-MIB::hrSWRunName.912 = STRING: "wcescomm.exe"
```

Figure 4.33: Server's software resources retrieved by the agent

The most important parameter that a router should monitor is the status of its interfaces. In this example, the agent monitors the status of the LinkSys router's interfaces. The router may operate as a NE indicating that

there is no problem related to its operation but that one interface could be faulty or deactivated. Figure 4.34 demonstrates the output of the router's interfaces obtained by the agent. The SNMP operation that has been used for acquiring the status of the router's interfaces is the ifAdminStatus. The status of the second interface has value down(2) indicating that the particular interface is deactivated. By using an SNMP SET operation the status of the interface could be modified.

```
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifAdminStatus.2 = INTEGER: down(2)
IF-MIB::ifAdminStatus.3 = INTEGER: up(1)
IF-MIB::ifAdminStatus.4 = INTEGER: up(1)
IF-MIB::ifAdminStatus.5 = INTEGER: up(1)
IF-MIB::ifAdminStatus.6 = INTEGER: up(1)
IF-MIB::ifAdminStatus.7 = INTEGER: up(1)
```

Figure 4.34: Router's network interfaces obtained by the agent

4.5 Conclusion

The FCAPS functions of existing LNMSs are typically implemented as stovepipe systems. This means that each FCAPS function operates in isolation. For example, faults collected for fault management and statistics collected for performance management are processed and analysed by fault and performance management components respectively. In order to have a comprehensive view and be able to diagnose a network problem, management information has to be exchanged between management systems. Due to this isolation, the management information that carries valuable information concerning the health of the network cannot be shared and processed by other management systems.

This chapter has presented the management communication (Low Level Management Communication) between network devices. The design and development of a network management agent that collects management information from the NE has been described. Moreover, the design of an XML-gateway component that connects to the Network Management Middleware Layer has been presented.

The design and the development of an LNMS that performs fault, performance and configuration management based on data acquired from the agent have been explained. Messages required for performing performance, fault and configuration management have also been defined.

Chapter 5 : DESIGN OF THE NETWORK MANAGEMENT MIDDLEWARE LAYER

5.1 Introduction

NGN management requires multiple NMS systems to be able to operate as one integrated entity [WEIS07]. This requires interoperability between these distributed systems. A Middleware Layer can provide mediation mechanisms that can simplify the task of bridging the distributed systems. Middleware in general can be seen as a layer between applications and operating systems. The role of the middleware is to provide a simple, consistent way for integration in a distributed programming environment.

In the previous chapter we presented how to collect and process management information from the network infrastructure. The amount of information extracted from different network elements can be enormous depending on the scale of the network infrastructure. In an integrated environment several management systems are required to use this management information for different purposes. As a result, an optimal way is required to categorize and make available the appropriate management information to multiple management systems.

The previous chapter proposed that management information should be expressed in XML-based data format due to its all-encompassing capability of being able to include data from many different databases distributed over multiple servers.

This chapter proposes a Network Management Middleware Layer based on messaging and asynchronous communication that will remove the integration complexity from the management systems. Moreover, the proposed middleware will handle the heterogeneity on the information expressed by different systems and will address the following questions:

- If management information is required to be consumed by other management systems how this information can be used?
- How to deal with management information that is heterogeneous?
- How to connect different management systems together?
- How to route the information to the appropriate management system?
- Where these functions should be performed?

The chapter first addresses the functional view (what should the solution do?) and the technical view (How should the solution work?).

5.2 The Network Management Middleware Functional Architecture

As mentioned in chapter 4, in an NGN, two levels of network management are possible. At the local level, each transport network will have its own Network Management System (NMS), each NMS may use different platforms, technologies, protocols and information model. NMS at this level will be called Local Network Management System (LNMS). At the

NGN or the global network level, a global network management system (GNMS) to provide the overall management of the network. To enable the GNMS to interact with the heterogeneous LNMSs, a network management middleware (NMM) architecture needs to be defined in order to provide interoperability between different LNMSs. Before deriving the functional architecture of the NMM, its functional requirements need to be defined.

5.2.1 Middleware Requirements

Middleware requirements can be generalized into five categories [PINU04], [EMME00], as shown below:

- **Heterogeneity:** Middleware should support heterogeneous hardware and software platforms.
- **Network communication:** The middleware should enable communications between heterogeneous network components, regardless of their underlying transport protocols.
- **Coordination:** Middleware should enable coordination of information exchange between heterogeneous applications and services.
- **Reliability:** Middleware should ensure that information are guaranteed to reach their destination complete and uncorrupted and in the order they were sent.
- **Scalability:** The Middleware should accommodate future network and service expansion.

The middleware requirements, combined with the network management requirements defined in Chapter 4, should govern the type of services to be supported in the middleware to allow communications between applications and services in the GNMS and those in the LNMS. For clarity, applications/services supported by the GNMS will hereafter be called the global network management applications (GNMA) and those by the LNMS the local network management applications (LNMA).

5.2.2 The Middleware Functional Architecture

Figure 5.1 illustrates the functional architecture of the Network Management Middleware, which is based on the service oriented architecture (SOA) and the message-oriented middleware (MOM) concepts supported by the Core NMS Service Bus.

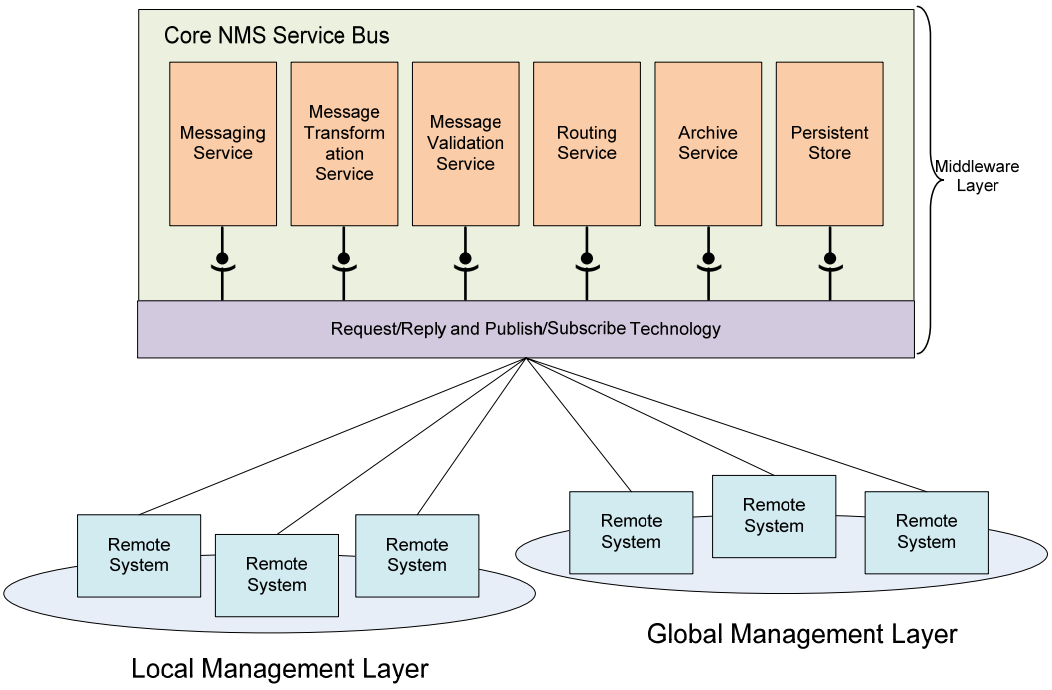


Figure 5.1: Functional Architecture of the Network Management Platform

The GNMS layer provides global network management applications and services to support generic network management FAB functions for the NGN. These applications and services are provided the management and co-ordination of the underlying heterogeneous transport networks. For example, functions to enable handover between two different transport networks requires information of the performance and configurations of these two networks. The LNMS layer exposes local network management FCAPS functions as a set of network management services of individual transport networks in the transport stratum of the NGN, each provided by its own network management system. Services provided by the GNMS and LNMS will interact with each other through the middleware offered by the NMM.

Two main categories of services can be considered in the NMM:

- *Interface Management Services (IMS)*
- *Core Messaging Services (CMS)*

Interface Management Services include service registration, service lookup, service invocation. While service registration and service look up are integral parts of interface management, this thesis concentrates on service invocation.

Service invocation includes message validation, message transformation, message routing, protocol bridging. These components within the service invocation framework ensure that the requirements for heterogeneity, network communication, coordination, reliability are met.

Core Messaging Services include services that will enable service publication and subscription through different messaging models. It can also provide notification or topics alert services. Two storage services associated with the core massaging services will also be included, namely, Persistent Message Storage and the Message Archive Service. Table 5-1 summarises the services provided by the middleware.

Table 5-1: Services Provided by the Middleware

	Heterogeneity	Network Communication	Coordination	Reliability	Security	Scalability
Validation				*x*	*x*	
Transformation	*x*					
Routing			*x*			
Protocol adaptation	*x*	*x*				
Persistent Message Store				*x*		
Message Archive Service				*x*		
Message-based communication						*x*

These services are described below:

- *Transformation Service*: This service transforms management information into a common information model. This transformation should contain message decomposition with needed information (i.e. metadata)

- *Validation Service*: This service validates the information that Core NMS Service Bus receives from the remote services.
- *Routing Service*: The Routing Service determines the destination of each message. This service will be realized through the implementation of a normalized message router.
- *Protocol Adaptation Service*: This service will adapt heterogeneous communication protocols through a unified API.
- *Message Archive Service*: This service keeps a record of every message sent by remote services.
- *Persistent Storage Service*: A persistent store is developed in order to store management information consumed by services in the NMS Layer. The Core NMS Service Bus consumes, so that in the case of middleware failure the data can be recovered.

5.2.3 The Message Oriented Middleware (MOM) Concept

5.2.3.1 Message Producer, Message Consumer and Message Channels

In the MOM concept, message applications employ a message client API to communicate with each other through a messaging system, in this case, the Core NMS Service bus. In the MOM communication paradigm, an application can act as a message producer that produces (sends) the message or a message consumer that consumes (receives) the message. An application may have dual functionalities of being a producer and a consumer at the same time. In relation to the NMP, the NMSs will primarily be the message producers whereas the GNMS will be the message

consumer, which retrieves local network management information from the individual LNMSs.

Communications between producers and consumers are via virtual channels [ERL10]. Each application may have its own channel or multiple applications can share a single channel depending on the implementation. Figure 5.2 shows the relationship between the producers and the consumers. The Networks produce the management information and the Core NMS publishes the information into virtual channels, the global users are acting as consumers, requesting management information from the Core NMS Service Bus. The global users can be remote services residing on remote systems.

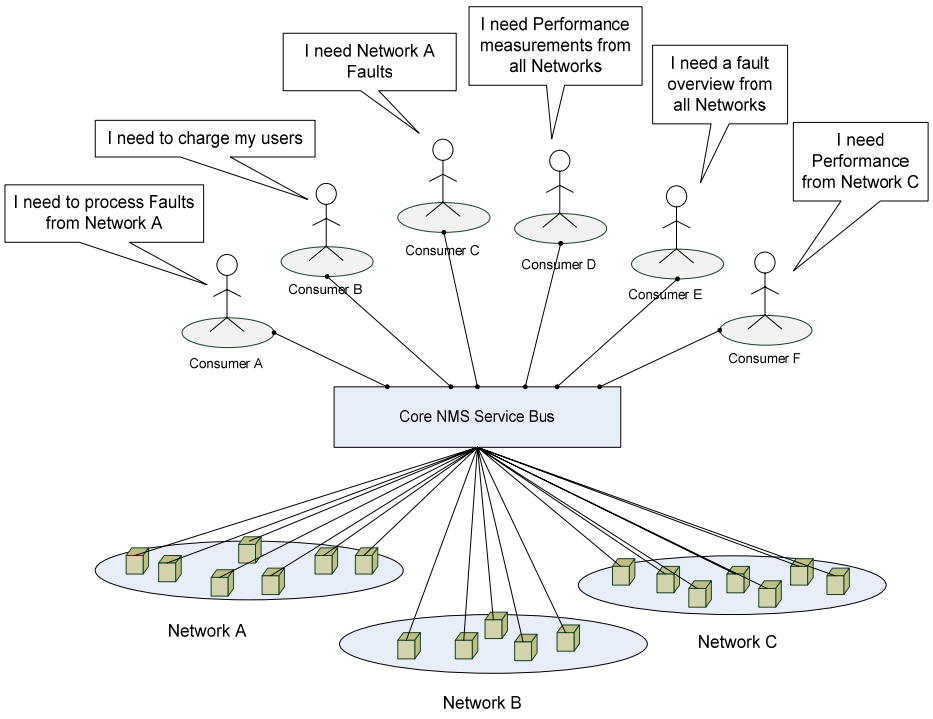


Figure 5.2: Communication Scenario between Core NMS Service Bus and consumers

Virtual channels can be expressed as queues or topics, depending on the messaging models for information exchange through them. Virtual channels can be further categorized into different groups according to the type of events. Consumers can subscribe to the group of interests and receive all messages sent to the groups. This categorization can help filtering messages accordingly.

5.2.3.2 Messaging Models

Two common models are used to exchange information through message virtual channels, namely, the point-to-point and publish/subscribe (pub/sub) models. For the point-to-point model, the channels are often referred to queues; for pub/sub model as topics.

5.2.3.2.1 Point-to-Point

The point-to-point messaging model allows message clients to send and receive messages asynchronously via virtual channels known as queues. Messages from the message producer are routed to the message consumer via a queue. While there is no restriction on the number of message producers who can publish to a queue, a message in the queue can only be received by a single message consumer.

Figure 5.3 illustrates the point-to-point messaging model.

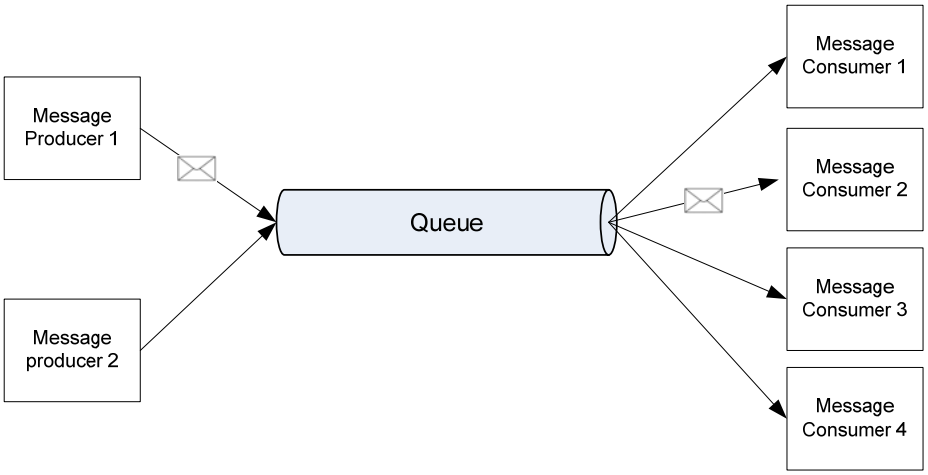


Figure 5.3: Point-to-point management messaging paradig

Even though multiple consumers are allowed to connect to a queue, each message will only be received by a single consumer. This property enables load balancing to be supported in the system. In this model, messages are always delivered and will be stored in the queue until a consumer is ready to retrieve them.

5.2.3.2.2 Publish/Subscribe

In the publish/subscribe model, messages are published to a virtual channel called topic. Unlike the point-to-point model which only supports one-to-one message distribution, the pub/sub model supports one-to-many and many-to-many distribution mechanism, allowing a single producer to broadcast a message to hundreds of thousands of consumers [BALD05].

Figure 5.4 illustrates the publish/subscribe communication paradigm.

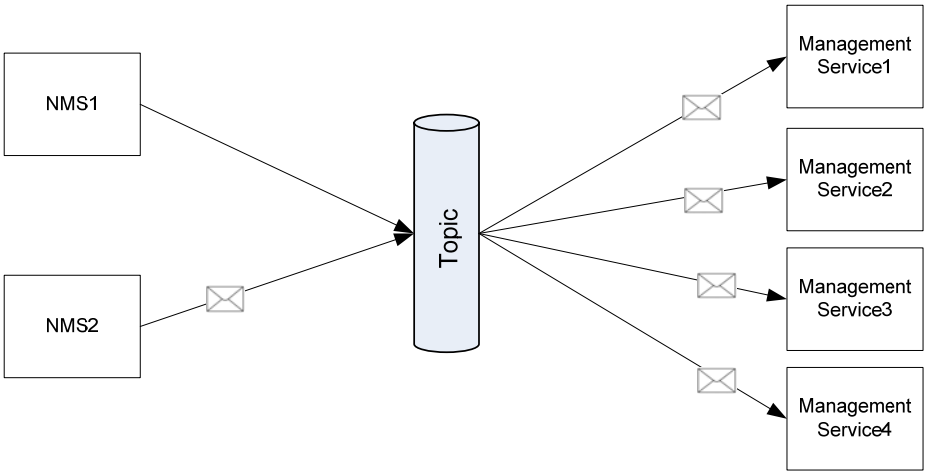


Figure 5.4: Publish/subscribe management messaging paradigm

There are two types of subscription within the publish/subscribe paradigm; the durable subscription and the non-durable subscription. The non-durable subscription allows temporary subscriptions to receive messages only when they are actively listening to the specific topic. Topics cannot hold messages except if the consumer use the durable subscription. In duration subscription, when a subscribing consumer is disconnected from the messaging server, the message server stores the message and holds the data until the consumer reconnects. Thus, durable subscription can survive the failure of the subscribing consumer.

5.2.3.2.3 Request/Reply

The request/reply model is used for the World Wide Web (WWW), where a client requests a page from a server and the server replies with the requested page. Any producer who sends a message (web page) must be ready to receive a reply from consumers at some stage in the future.

The publish/subscribe and point-to-point message models are primarily for asynchronous communications between message producers and consumers. However, synchronous interactions between these two parties are sometimes required. A request/reply message pattern can be built on top of the two MOM message models to perform both asynchronous and synchronous request/reply. MOM message channels (topics and queues) are not bidirectional. To perform a request/reply operation, a requester must use two channels: one for request and one for the response (reply). A correlation ID can be used to correlate the request message with the reply message.

5.2.3.2.4 Pull/Push

Pull and Push are methods used by a consumer to receive messages from a producer. In the pull method, a consumer can pull a message from the provider by polling the provider to check for any messages. In the push method, a consumer can request the provider to send on relevant messages as soon as the provider receives them, which effectively means that the consumer instructs the provider to push messages to the consumer application.

5.2.3.3 Message Composition

The message consists of three parts; a header, the properties and a body. Figure 5.5 illustrates the message composition.

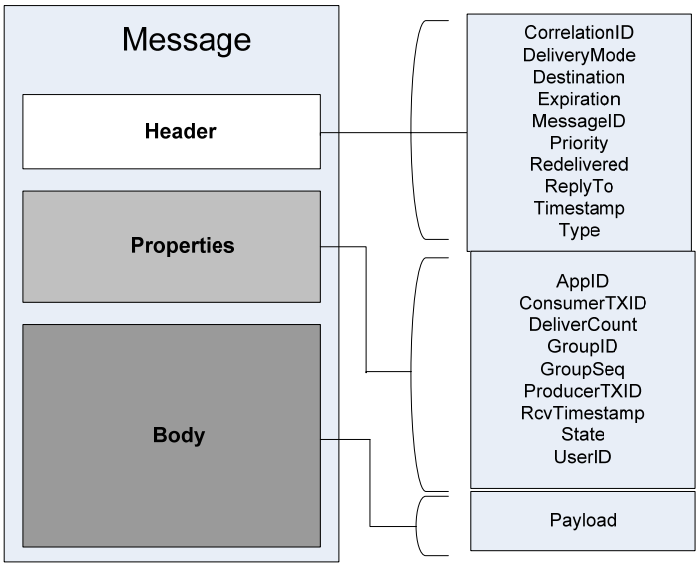


Figure 5.5: Message composition

The Header contains metadata information about the message used by producers and consumers. The header fields that are assigned in each message are described below:

- **CorrelationID:** Associates message with a previous message. This header is used in order to associate a response message with a request message.
- **DeliveryMode:** The messaging service supports two types of delivery modes for the messages. The first mode is the persistent mode and the second is the non-persistent mode. In the persistent mode, the messaging service stores the messages into a database so that if the messaging service fails, the data can still be retained. The messaging service uses the *once-and-only–once* function for sending the message, which means that if the messaging service fails, the message will not be lost and will be delivered once the message service resumes and only once to the message

consumer. Messages with persistent mode add more overheads due to the storage of the data. However, if reliability is more important than performance, such as the case of the Network Management Platform (NMP) where the messages provide crucial information concerning the health of the managed network. At the non-persistent mode, the messaging service delivers the message *at-most-once*. In other words, if the messaging service fails, the messages will be lost and will not be sent again. The non-persistent mode produces less overhead compared to the persistent mode. In considering the above, this thesis considers the use of persistent delivery in order to ensure the reliability of the NMP.

- **Destination:** Indicates the destination to which the message is being sent. This is an important header that is used from clients who consume messages from more than one destination.
- **Expiration:** Indicates the time that the message expires. It prevents the delivery of a message after it expires.
- **MessageID:** Uniquely identifies a message that is assigned by the messaging service. It is used for message processing or for historical purposes in a message storage mechanism.
- **Priority:** It assigns a level of importance to the message. The Priority header is assigned by the messaging service and it is applied to all messages that have been sent from the messaging service. There are 10 levels of message priority where zero is the lowest and nine is the highest.

- **Redelivered:** Indicates the likelihood that a message was previously delivered but not acknowledged. This can occur if a service fails to acknowledge delivery. If the messaging service has not been notified of the delivery an exception is being indicated.
- **ReplyTo:** Specifies a destination where a message should be sent.
- **Timestamp:** The Timestamp header denotes the time that the message is sent by the message producer to the message consumer. The value of this property uses a standard millisecond value.
- **Type:** This header is used to semantically identify the message type. This header field provides a reference to the message's definition in the messaging service repository.

The properties section in the message is an optional field that adds a set of additional custom information to the message. These properties occupy a section of the message so that filtering can be applied to the message. The properties headers are listed below:

- **AppID:** Identifies the service that sends the message.
- **ConsumerTXID:** This optional header is the transaction identifier that identifies the transaction which the message can be consumed.
- **DeliveryCount:** This header is a counter that stores the message's delivery attempts.
- **GroupID:** This filed is dedicated to the message group of which the message is a part.

- **GroupSeq:** This header indicates the sequence number of the message within the group.
- **ProducerTXID:** Is the transaction identifier that identifies the transaction which the message can be produced.
- **State:** This header is used in order to define a provider-specific state.
- **UserID:** Identifies the user sending the message.

The Body which is the actual payload of the message contains the data from the message producer. The body can contain XML message types that allow the message payload to be accessed using common XML parsing technologies. More specifically, there are two types of message body:

- **Message:** Is used in order to send a message with no payload, only header and properties. This type of payload is used for simple event notification.
- **TextMessage:** It is a message whose payload is a string. It is commonly used in order to send textual and XML data.

5.2.4 Reliability of Management Messages

In previous sections, the concepts, the actors, the message model and message composition underlying the MOM have been presented. They will be applied to the network management middleware architecture, the following applies:

- The durable publish/subscribe messaging model is used to ensure reliability.
- The messages of the NMP will follow the message format as described in section 5.2.3.2 above. The Body will contain management information produced by the LNMA.
- The message consumer will be the application clients in the GNMS. In other words the message consumer is the GNMA.
- The message producer will be the application clients in the LNMS, i.e. LNMA.

A message queue acts as a message store, accumulating messages that are ready for transmission. Queues can be ordered in various fashion, from first in first out (FIFO) to priority queues with messages of higher priority being moved to the front of the messaging queue. In Figure 5.6, the FIFO method for storing messages into the messaging queue is presented.

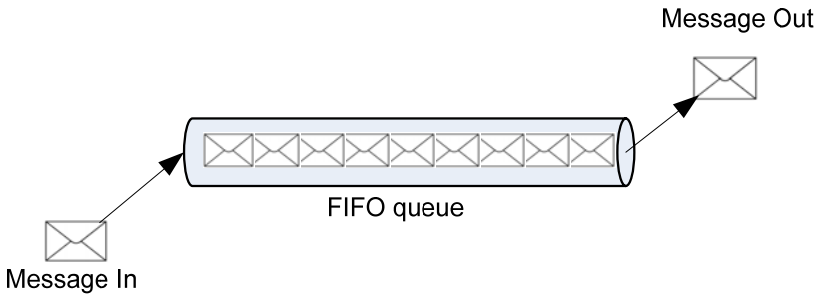


Figure 5.6: FIFO message storage for messaging queues

The GNMA uses a durable subscription with the publish/subscribe paradigm in order to subscribe messages to ensure reliability. If a message is subscribed with duration subscription, the message will then be marked as a persistent message. A persistent message in the message queue can only be deleted when it has been consumed and acknowledged, otherwise it will remain in the queue. In the case of the publish/subscribe paradigm for one-to-many subscriptions, a persistent message can only be deleted when all subscribers have consumed and acknowledged the message. Thus, a message queue acts as a message store and under duration subscription, the message queue acts as a persistent message store.

Figure 5.7 illustrates the process of storing messages to the topic. When a message is marked as persistent, the messaging service utilizes a store-and-forward mechanism to store persistent messages to ensure that they can be recovered if there is a failure of either the messaging service or the message consumer, i.e. the GNMA client. The steps involved in delivering a publish/subscribe message by using persistent messaging and durable subscription are explained below.

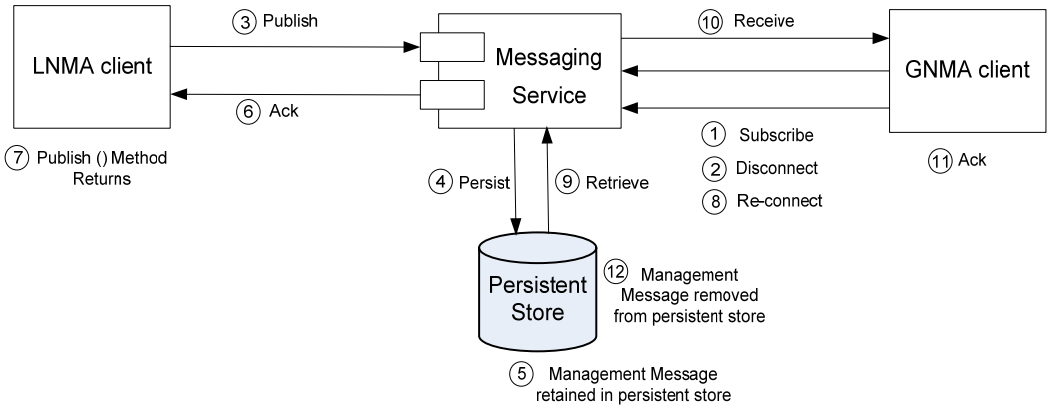


Figure 5.7: Reliable publish/subscribe with acknowledgments, persistence and durable subscription

The processes involved in the reliable publish/subscribe messaging are described below:

1. GNMA client subscribes and indicates that the subscription is durable.
2. Management Service disconnects from the messaging service, due to a failure.
3. LNMA client sends the message using publish() method. The publish() method will block and wait until it receives an acknowledgment from the messaging service.
4. Messaging service writes the message to the persistent storage entity which in this case is a MySQL database.
5. Message is held in the database.
6. Acknowledgment is sent back to the LNMA client indicating that the message is now stored in the persistent store.
7. publish() method returns.

8. GNMA client reconnects and re-establishes the subscription.
9. Message is retrieved from the persistent store.
10. Message is delivered to the GNMA client.
11. GNMA client acknowledges to the messaging service that it has successfully received the message.
12. Messaging service removes the message from the persistent store.

5.3 Design of MOM Services

To satisfy the middleware requirement, seven service components are created as indicated earlier. These services are explained in greater detail in the following sections.

5.3.1 Messaging Service

The messaging service is a service component that has been developed on the Middleware Layer in order to allow the communication and data transfer from one management system to another. This service uses Request/Reply and Publish/Subscribe technologies that are based on the Java Messaging Service (JMS) specification [SUNJMS] JMS provides a standardized API for sending and receiving messages using Java programming language in a vendor-neutral manner [SUNJMS]. The messaging operations that are performed in the messaging service are depicted in figure 5.8.

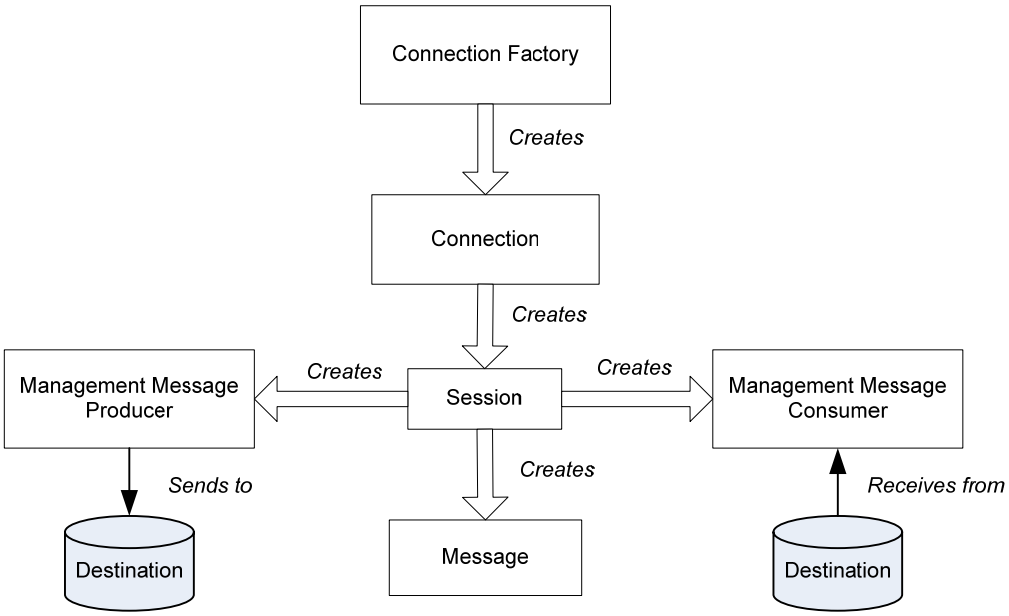


Figure 5.8: Messaging Service objects and their relationships

- **Connection Factory:** The Connection Factory encapsulates a set of configuration properties for a connection. The messaging service uses the Connection Factory to create a connection. Each Connection Factory is an instance of the QueueConnectionFactory or TopicConnectionFactory interface.

- **Connection:** The Connection object encapsulates the Management Service's active connection to the messaging service. The Management Service uses a Connection in order to create sessions.

- **Session:** A Session is a single threaded context for sending and receiving messages. The messaging service uses a Session in order to create messages, Message Produces and Message Consumers.

- **Message Producer:** Is the messaging service that sends the messages to a destination. The destination for the Message Producer is the Management Service and it is implemented by the MessageProducer interface.
- **Message Consumer:** Is the Management Service that receives the message from a destination. The destination for the Message Consumer is the messaging service and it is implemented by the MessageConsumer interface.
- **Message:** The message object encapsulates a message that is sent or received by the Message Producer or Consumer.

5.3.2 Message Validation Service

The NGN Transport Stratum consists of heterogeneous networks that act as one converged network [M.3060]. One major problem for managing the converged network is managing heterogeneous management information that the different networks produce. Generally, the management information extracted from different networks could contain errors regarding the content of the information that they share or could share messages that cannot be understood by other applications. A solution for this problem is to subjecting their information to reference validation. For this reason, a validation mechanism is proposed in order to eliminate the creation of unnecessary faults and errors in invalid messages which store invalid information regarding specific managed nodes that cannot be later processed by the GNMS. A validation service component has been developed for validating messages received from heterogeneous LNMSs.

5.3.2.1 Validation XML Schema for Management Messages

In general, a well-formed message is a message that conforms to the XML syntax rules [W3C09]. These rules are illustrated below:

- The message has to begin with an XML declaration such as <?xml version="1.0" encoding="UTF-8"?>.
- The message must have one unique root element.
- The start-tags must have matching end-tags (i.e. <Results></Results>).
- All elements in the message has to be case sensitive.
- Attribute values have to be quoted (i.e. xmlns="http://esb.nms1.org")
- All elements must be properly nested.

The message, even if it is well-formed, it can still contain errors. These errors are related to the content of the information that each message provides. In order to avoid errors related to the content of the message, the proposed Network Management Platform defines its own schema (Validation.xsd) that is stored in the metadata repository. The Core NMS Service Bus uses the Validation.xsd schema in order to describe the messages in a way that the GNMA client can understand.

The Validation.xsd schema contains elements and attributes from different LNMSs; each LNMS may have different element representation for expressing the same type of information from other LNMS. For instance, eventid element (<eventid>) defined in LNMS1 and id element (<id>) defined in LNMS2 both indicate an event identifier with an integer attribute

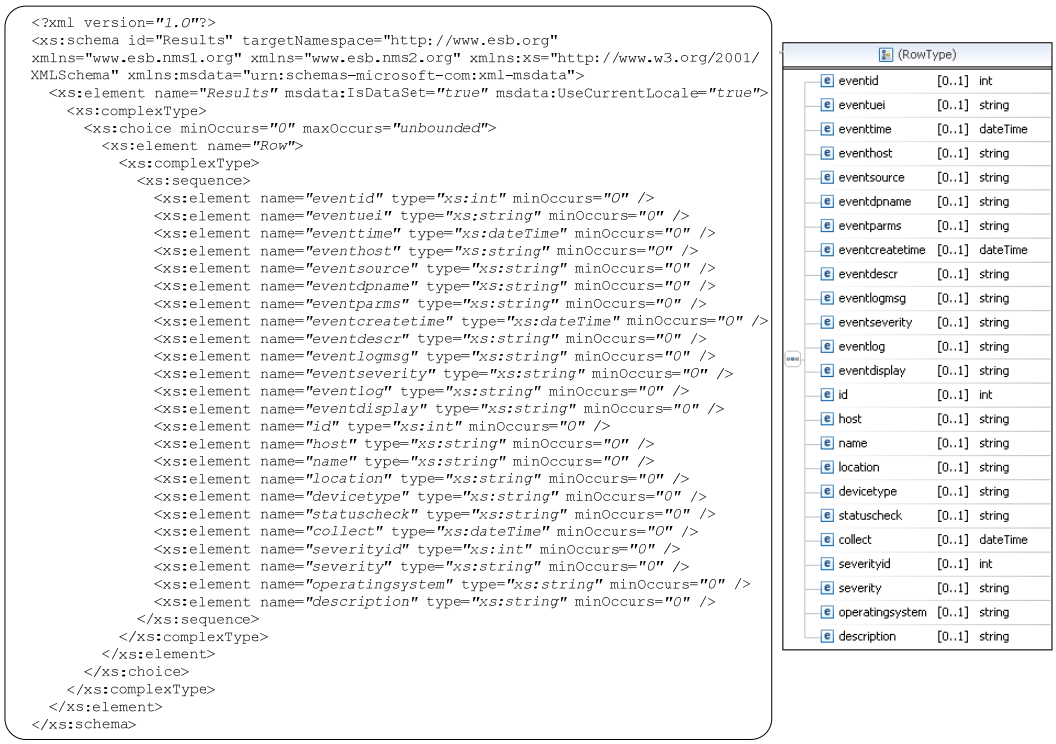
type. All valid elements and attribute types need to be included into the validation schema.

The XML schema (xsd) describes the structure of the XML-based message. The purpose of the schema is to define the legal building blocks of the message. The schema contains a formal description of what comprises a valid message. In more detail, the XML schema defines [W3C09]:

- Elements that can appear in the message.
- Attributes that can appear in the message.
- Data types for elements and attributes.
- Default and fixed values for elements and attributes.
- Child elements in the message.
- Order of the child elements.
- Number of the child elements.
- Whether an element can have an empty value or needs to include management data.

All the attribute names that have been used in the Validation schema were standardized and defined by the W3C recommendation [W3C09]. The namespace URI that defines the standardized attributes in the Validation schema is the xmlns:xs="http://www.w3.org/2001/XMLSchema".

Figure 5.9 shows the XML schema of the message defined in the Message Validation Service.



```
<?xml version="1.0"?>
<xs:schema id="Results" targetNamespace="http://www.esb.org"
xmlns="www.esb.nms1.org" xmlns="www.esb.nms2.org" xmlns:xs="http://www.w3.org/2001/
XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Results" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Row">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="eventid" type="xs:int" minOccurs="0" />
              <xs:element name="eventuei" type="xs:string" minOccurs="0" />
              <xs:element name="eventtime" type="xs:dateTime" minOccurs="0" />
              <xs:element name="eventhost" type="xs:string" minOccurs="0" />
              <xs:element name="eventsource" type="xs:string" minOccurs="0" />
              <xs:element name="eventdpname" type="xs:string" minOccurs="0" />
              <xs:element name="eventparms" type="xs:string" minOccurs="0" />
              <xs:element name="eventcreatetime" type="xs:dateTime" minOccurs="0" />
              <xs:element name="eventdescr" type="xs:string" minOccurs="0" />
              <xs:element name="eventlogmsg" type="xs:string" minOccurs="0" />
              <xs:element name="eventseverity" type="xs:string" minOccurs="0" />
              <xs:element name="eventlog" type="xs:string" minOccurs="0" />
              <xs:element name="eventdisplay" type="xs:string" minOccurs="0" />
              <xs:element name="id" type="xs:int" minOccurs="0" />
              <xs:element name="host" type="xs:string" minOccurs="0" />
              <xs:element name="name" type="xs:string" minOccurs="0" />
              <xs:element name="location" type="xs:string" minOccurs="0" />
              <xs:element name="devicetype" type="xs:string" minOccurs="0" />
              <xs:element name="statuscheck" type="xs:string" minOccurs="0" />
              <xs:element name="collect" type="xs:dateTime" minOccurs="0" />
              <xs:element name="severityid" type="xs:int" minOccurs="0" />
              <xs:element name="severity" type="xs:string" minOccurs="0" />
              <xs:element name="operatingsystem" type="xs:string" minOccurs="0" />
              <xs:element name="description" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Figure 5.9: Message Validation.xsd schema

The first line of the XSD (eXtensible Schema Definition) document indicates the version of the XML specification that the document uses since the Validation.xsd schema is based on the first version of the XML standard.

The schema specifies a unique ID attribute with the value “Results”. This ID value classifies the schemas of the Message Validation Service in case other schemas are needed to be stored in the metadata repository for other validation purposes. The namespaces (xmlns) attribute returns with Uniform Resource Identifier (URI) attributes to identify the domain where the event occurred. The targetNamespace attribute specifies the URI reference of the namespace of the schema. The ‘Results’ element is defined as a complex type in the validation schema and can be referred to as the parent element in this example schema. The parent element has

several child elements (or nested elements) as presented in the table below (Table 5-2).

Table 5-2: Nested elements in the Validation schema

Nested elements (NMS1 and NMS2)	Description
NMS1: eventid NMS2: id	Unique numeric value indicated in each event that occurs
NMS1: eventuei	Indicates the Network Management tool that has been used to initiate the event (i.e. OpenNMS, CACTI, OpenView, NINO, etc.)
NMS1:eventtime NMS2collect	Time of the event
NMS1: eventhost NMS2: host	Indicates the IP address of the Network Element that the event occurred
NMS1: eventsource NMS2: operatingsystem	Description of the network element
NMS1: eventdpname NMS2: location	Location of the network element
NMS1:eventcreatetime	Timestamp of the event stored in database
NMS1: eventdescr NMS2: Description	Description of the event
NMS1: eventlogmsg	Description of the event presented as a log message
NMS1:eventseverity NMS2:severity	A value indicating the severity of each event
NMS1:eventlog	Indicates the choice of storing the description of the event message
NMS1: eventdisplay	Indicates the choice of displaying the event message in the NMS
NMS2: Sverityid	Numeric value indicating the id of the severity
NMS2: name	Indicates the event element name
NMS2: operatingsystem	The platform type of the network element that the event occurred
NMS2: devicetype	Indicates the type of the network element (router, switch, etc.)
NMS2: statuscheck	Indicates the if the network element is registered for events

The nested elements specified in the Validation schema can occur more than once in the message to allow each message to contain more than one event in order to reduce message interactions between application

clients in the LNMSs and GNMS. As a result, the network's bandwidth consumption can be minimized. Table 5-3 illustrates the XSD attributes used in the validation schema.

Table 5-3: XSD attributes

XSD attributes	Description
xs:complexType	A complex type element is an XML element that contains other elements and/or attributes
xs:int	The integer data type is used to specify a numeric value without a fractional component
xs:string	The string data type can contain characters, line feeds, carriage returns, and tab characters
xs:dateTime	The dateTime data type is used to specify a date and a time. The dateTime can be specified in the following form "YYYY-MM-DDThh:mm:ss"
xs:sequence	The sequence element specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times
minOccurs	indicator that specifies the minimum number of times an element can occur
maxOccurs	indicator that specifies the maximum number of times an element can occur

The child elements defined in the Validation schema have the following values: integers, strings, and date and time. Moreover, the number of the element occurrence is defined in the schema by using the maxOccurs attribute. The value of maxOccurs is set to "unbounded" in order to indicate that the nested elements can have unlimited appearance in the message. The attribute name minOccurs indicates the minimum number of times an element can occur in the message. The xs:sequence attribute defines the appearance order of the nested elements in the message. The type attribute indicates the value type of the each nested element that is expected.

5.3.2.2 Message Validation Service Architecture

Applications in the GNMS, i.e. the GNMA, which are the message receivers, must be able to interpret the messages published by the applications in the LNMSs and understand their meaning. This is not always possible, because a message could be invalid. For example, the message body may cause parsing errors or lexical errors, or there are missing information in the message header, or the properties values in the message itself are wrong.

In other cases, when virtual channels are categorised into different groups for different management information type, if a message is put in the wrong category, the Message Validation Service should be able to detect such error.

In Figure 5.10, the messaging service creates an incoming messaging channel and two outgoing message channels. The incoming message channel receives the messages transmitted by LNMAs. As for the two outgoing channels, one is responsible for connecting the message validation service with the GNMA and the other with the error message handler.

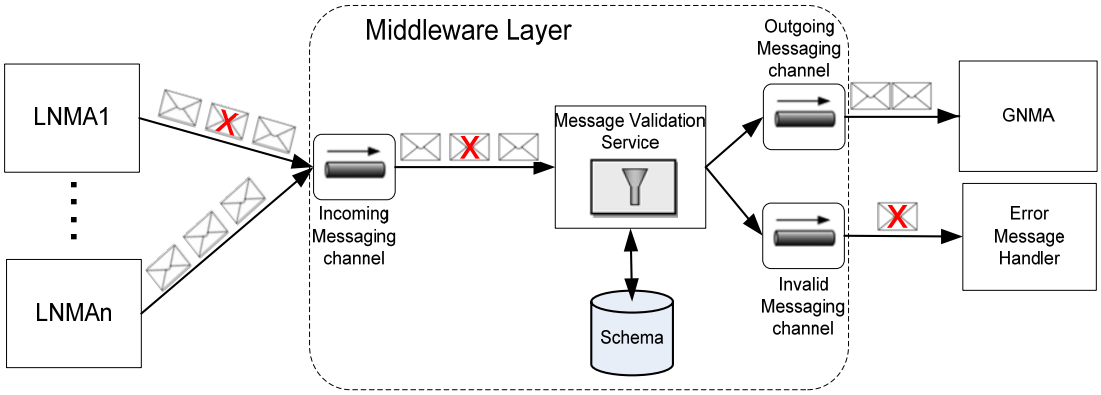


Figure 5.10: Message validation Service

A message sent by a LNMA will be validated before reaching its destined message consumer, the GNMA. This message contains management information regarding a fault in a network node. The message is passed to the virtual channel and is processed through the Message Validation Service, where it will be compared against a validated XML schema. If the messages satisfy the requirements of the XSD schema, then they can successfully proceed to the destination, which is the GNMA. If they fail, the Message Validation Service initiates an invalid fault alert and sends the invalid messages to the error message handler.

5.3.3 Message Transformation Service

5.3.3.1 Architecture

Legacy systems only understand their own proprietary protocols and messages and rarely agree on a common data format. This makes system and data integration virtually impossible. One solution for integrating heterogeneous systems is to modify the systems through data

transformation, where data of one system is transformed into the data format of the other. However, this is not the most efficient way to integrate systems due to the fact that it requires a lot of changes in the system's logic and data format changes are not economically feasible [CARE02a]. Furthermore, adjusting the data format of one system to match that of another system makes the overall architecture more tightly-coupled.

Another approach is to use XML-based messages to enable service interoperability. Transformation is performed using a stylesheet language called XSLT (eXtensible Stylesheet Language Transformation) to restructure XML documents from one format to another and to transform and/or enhance the content of the XML message. The stylesheet specifies how the XML data will be displayed. XSLT uses the formatting instructions in the stylesheet to perform the transformation. These instructions inform the transformation processor of how to process a source document in order to produce a target document that is understood by all systems.

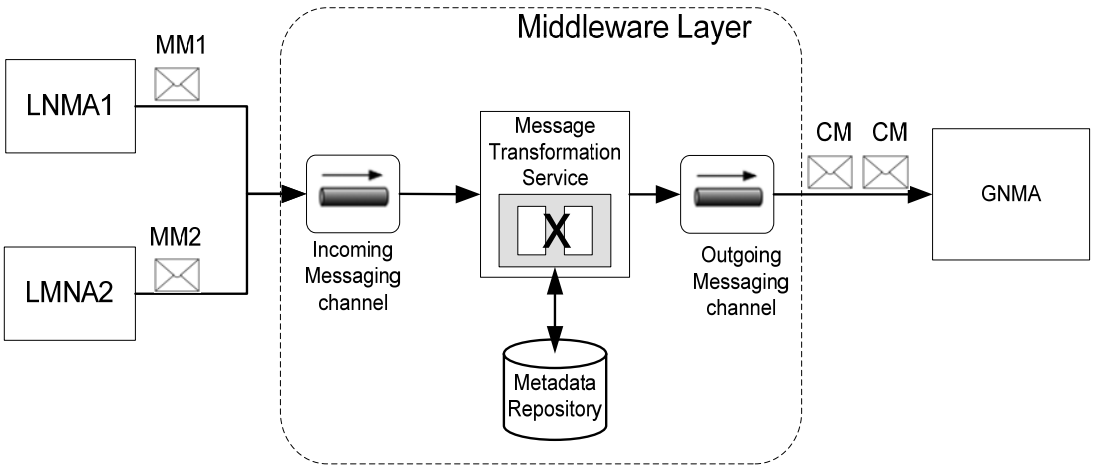


Figure 5.11: Message Transformation Service created in the Network Management Platform

Figure 5.11 demonstrates the Message Transformation Service that transforms messages from one format into a common format. In this scenario, messages are sent to the GNMA by two applications of two different LNMSs. The messages need to be transformed into a common information model to be understood by the GNMA.

Messages from LNMA1 (MM1) and messages from LNMA2 (MM2), each having its own proprietary data formats, are passed to a common message incoming channel created by the Message Service in order to be delivered to and processed by the Message Transformation Service. The Messaging Service also creates an outgoing messaging channel responsible for connecting the GNMA to the Message Transformation Service.

The Message Transformation Service has a central repository for storing metadata defining the appropriate message format understood by the GNMA. The metadata can be stored in a number of formats. A common format for XML messages is defined in the XSLT. The Message Transformation Service makes an external call to the metadata repository for a lookup (searching the data structure of the XSLT). The messages (MM1 and MM2) are compared against the XSLT schema and the content is being transformed according to the XSLT schema. Finally, the Messaging Transformation Service will place the transformed messages to the outgoing messaging channel for delivery to the GNMA.

In the case the Transformation Service component is required to transform information based on different information models, each XML namespace (xmlns) included in the messages, has to be mapped to a particular XSLT

stylesheet. In the proposed NMP, it is assumed that GNMAs follow a common information model and as a result, one XSLT stylesheet is required.

5.3.3.2 The XSLT Transformation Stylesheet

The XSLT stylesheet must be a well-formed XML document and should comply with XSLT specification [W3C99b], which describes the allowed syntax and vocabulary. The content of the stylesheet depends on the input document structure (schema) and the required output structure. The XSLT stylesheet consists of a set of rules referred to as templates. A template consists of template rules that have two parts: a pattern which is matched against nodes in the source tree and a template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to many documents that have similar source tree structure. Figure 5.12 illustrates the implemented stylesheet for the Message Transformation Service.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="Results">
   <xsl:apply-templates select="Row"/>
</xsl:template>

<xsl:template match="Row">
      <xsl:for-each select="Row">
            <event_id><xsl:value-of select="id"/></event_id>
            <event_IP><xsl:value-of select="host"/></event_IP>
            <event_source><xsl:value-of select="name"/></event_source>
            <event_Location><xsl:value-of select="location"/></event_Location>
            <Host><xsl:value-of select="devicetype"/></Host>
            <event_status><xsl:value-of select="statuscheck"/></event_status>
            <event_Time><xsl:value-of select="collect"/></event_Time>
            <severity_id><xsl:value-of select="severityid"/></severity_id>
            <Severity><xsl:value-of select="severity"/></Severity>
            <event_system><xsl:value-of select="operatingsystem"/></event_system>
            <event_Description><xsl:value-of select="description"/></event_Description>
            <event_id><xsl:value-of select="eventid"/></event_id>
            <NMS_name><xsl:value-of select="eventuei"/></NMS_name>
            <Time><xsl:value-of select="eventtime"/></Time>
            <event_IP><xsl:value-of select="eventhost"/></event_IP>
            <Host><xsl:value-of select="eventsource"/></Host>
            <Location><xsl:value-of select="eventdpname"/></Location>
            <event_Time><xsl:value-of select="eventcreatetime"/></event_Time>
            <event_Description><xsl:value-of select="eventdesc"/></event_Description>
            <Output Message><xsl:value-of select="eventlogmsg"/></Output Message>
            <Severity><xsl:value-of select="eventseverity"/></Severity>
            <eventlog><xsl:value-of select="eventlog"/></eventlog>
            <eventdisplay><xsl:value-of select="eventdisplay"/></eventdisplay>
      </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figure 5.12: Transformation.xslt

The transformation.xslt begins with an XML declaration indicating the XML version that has been used and the encoding style. The <xsl:stylesheet> element defines the start of the stylesheet and declares the document to be an XSLT stylesheet. This element must have a version attribute to indicate the version of the XSLT in which the stylesheet is based on. In addition, this element declares the URI XSLT namespace attribute to ensure the uniqueness of the elements. The template element <xsl:template> contains rules to apply when a specified node is matched in the input message. These rules describe the contribution that the matched elements make to the output message. The ‘match’ attribute in the template element specifies which node of the input message the template is instantiated for.

When the XSLT transformer reads the input message, the root is the first node it processes and the rules matched that root node are carried out. The <xsl:apply-templates> element has been used to apply template rules to node <Row> of the incoming message. By applying the <xsl:appy-template> element , the XSLT transformer is instructed to compare each child element of the matched element (<Results>) against the templates in the XSLT stylesheet, and if a match is found, it performs the template for the matched node. In other words, when the processor in the XSLT transformer comes across child nodes that have value <Row> from the root node <Results> then it will process it.

The process is performed by applying the second xsl template that transforms child nodes of the parent <Row> into different node values. In order to perform transformation to every Row node that the message will have, the use of 'for-each' function has been applied. For example, for every node <Row> that has element value <id>, the XSLT transformer will modify the value to <event_id>. All elements that are required to be transformed need to be included in this template.

5.3.4 Message Routing Service

5.3.4.1 Routing Interfaces

Network intermediaries such as routers are discouraged to provide value-added application aware functions in the network infrastructure to avoid violating the internet's design guideline that states "network elements should not process packets that are not addressed to them" [BALD05] As

a result of the advances in hardware, software and network technologies, intermediaries are capable of processing data and providing decisions according to the content of the information [CAPO02], [CAST02]. For instance, peer-to-peer networks (P2P) use content-based routing mechanisms for dispatching information from publishers to subscribers [BHOL02]. Driven by the peer-to-peer network processing methods, a Routing Service component has been developed to provide routing functions based on the content of the messages.

Routing functions target messages that have been sent by the LNMAs and need to be distributed to different application clients of the GNMS. By implementing the Routing Service in the Middleware Layer, neither the GNMA nor individual LNMAs need to be concerned with routing functions (i.e. the destination of the message, message priority etc). As a result, these services become more loosely-coupled and more reusable because they do not have to specify the number of consumers that will be attached to or how to prioritize the message exchange.

As an illustration, Figure 5.13 shows a tightly-coupled scenario where different applications in the GNMS, i.e. the GNMAs, need to know the intimate details of how every LNMA wants to be communicated with, the number of methods it exposes and the details of the parameters that each method accepts. As the number of service components of each NMS increases, the number of interface connections that need to be created and maintained increases to n(n-1)/2 where n is the number of applications (n=6) [CHAP04]. This formula makes two assumptions for calculating the number of interfaces. First, it assumes that each

application endpoint has only one interface, and second, it assumes that every application needs to interact with every other application.

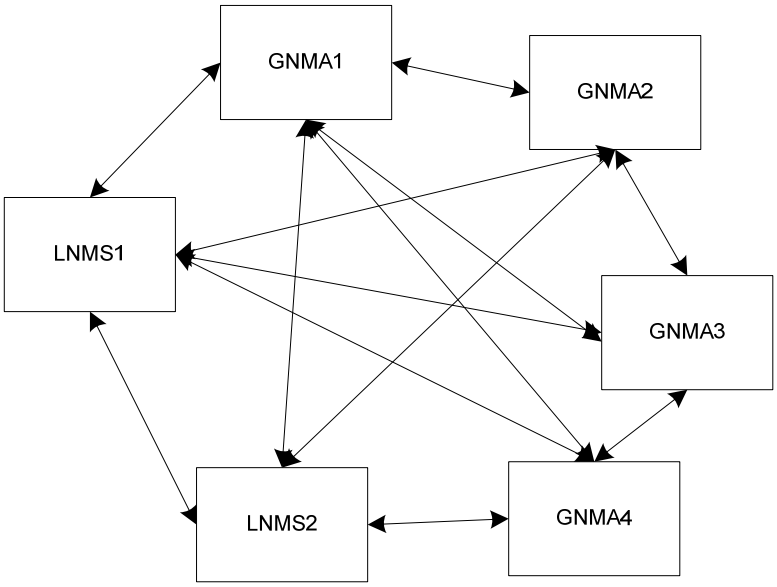


Figure 5.13: Number of tightly-coupled interfaces between network management remote systems

With the Routing Service provided by the Core NMS Service Bus, the number of interfaces is equal to exactly the number of remote services. This approach adds more flexibility to the infrastructure and makes the architecture more extensible for future needs.

Figure 5.14 shows a comparison of the interfaces that need to be created for an architecture that follows a tightly-coupled approach and an architecture that follows a loosely-coupled approach.

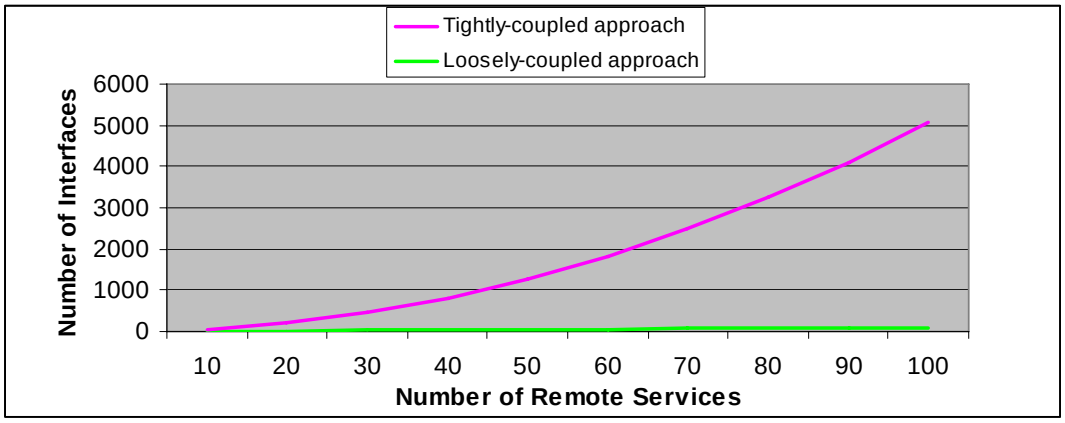


Figure 5.14: Number of interfaces for tightly-coupled and loosely-coupled remote services

5.3.4.2 Routing Functions and Routing Rules

The Routing Service is responsible for performing routing functions and this is achieved by applying routing rules based on the content of each message. Moreover, the Routing Service provides intelligent routing rules for routing messages to the appropriate destination. Motivated by the Enterprise Application Integration Patterns (EAI) that introduce solutions for integrating applications, the Routing Service implements three functions based on EAI [HOHP04]:

1. Content-based routing functions,
2. content-enrichment functions,
3. Content splitting functions.

Figure 5.15 demonstrates the Routing Service performing content-enrichment functions, splitting functions and content-based routing.

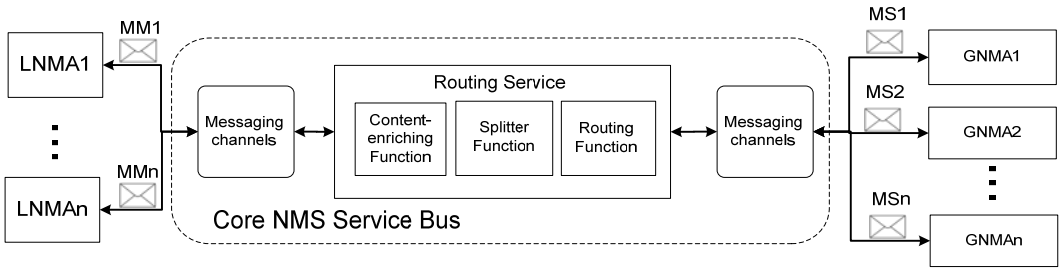


Figure 5.15: Routing Service performing routing functions in the Middleware Layer

When a message is sent from a LNMA to the Core NMS Service Bus via the incoming message channel, the Routing Service is activated. The content-enrichment function will inject additional information on each message indicating its origin. Each message is a large XML-based message, thus the splitting function is applied in order to split the message into smaller messages, where each message will contain one event of an individual network node. These event messages are processed by the routing function. Routing function routes the event messages based on the actual content of the message, rather than by the destination specified in the message header. The Routing Service parses the Event Message (EM) and applies a set of rules to its content to determine the event message's destination. As a result, the Routing Service provides a high degree of flexibility and adaptability to change. These are essential factors that should be taken into account when designing an SOA framework.

5.3.5 Persistent Storage Service

One of the middleware services offered by the Core NMS Service Bus is the provision of a persistence store that is designed for message

persistence. The persistent store is a relational database (MySQL) that stores all incoming and outgoing messages to and from the Core NMS Service Bus and to examine messages by querying the database [MySQL]. It is used in order to recover the data in case of a Middleware failure or failures of the LNMSs or the GNMS in order to increase reliability. The persistence store uses a schema consisting of three tables. Two of the tables are used in order to hold messages and the third table is used as a lock table in order to ensure that only the middleware can access the persistence store. The use of persistence store makes the NMP more reliable and fault tolerant.

5.3.6 Message Archiving Service

The messages that are passed through the Middleware Layer are stored into folders for inventory purposes. The Message Archive Service is created to accommodate management information in XML-based form messages. This service creates folders according to the message destination and stores all the messages that have been transmitted from different LNMSs. This function allows external access from service providers to request information regarding the health of the managed networks for inventory purposes. Six different folders have been created by the Archive Service in the vicinity of Middleware Layer. These folders are described in Table 5-4.

Table 5-4: Folders storing messages

Folder name	Storing name
NMS1_F	Contains all the messages (MM1) transmitted by NMS1.
NMS2_F	Contains all the messages (MM2) transmitted by NMS2.
MService1_F	All messages (MS1) received by Management Service 1.
MService2_F	Contains all messages (MS2) received by Management Service 2.
MService3_F	Contains all messages (MS2) received by Management Service 2.
MService4_F	contains all messages (MS4) received by Management Service 4
Topic1_F	All messages from Topic1
Topic2_F	Contains all messages from Topic2
Topic3_F	Contains all messages from Topic3
Topic44_F	contains all messages from Topic4

5.4 Conclusion

NGN is a very dynamic environment. Services will continuously need to be activated and deactivated in the Service Stratum. Devices will be added, removed and change configuration in the Transport Stratum; therefore, managing NGN will be a challenging task. NGN might be considered as one network, but it is by far the most complex of all. Its management has to deal with multiple vendors, multiple applications, multiple physical devices from data and voice networks, multiple databases, and multiple service layers (infrastructure plane, control plane, service plane). Any management solution for NGN must be architected in a way that it can scale to manage the current and future NGNs. This scalability challenge is a requirement for flexibility so that the solution can be rapidly adapted to support new services and technologies in the future without the need for long term and complex upgrades. SOA-based architecture facilitates loose coupling and “plugability” of new interfaces. As a result, it provides

extensibility and flexibility. The Middleware Layer, which is based on message-oriented technology, is the most important layer in the creation of an SOA-based framework that simplifies the task of bridging the distributed systems.

The benefits of using messaging technology for the development of the NMP include asynchronous communication, platform and language integration, throttling, variable timing and reliable communication.

- **Asynchronous Communication**: For the NMP, remote communication is a vital requirement due to the fact that the architectural approach follows a distributed pattern. The NMP is based on asynchronous communication. Messaging service supports this communication pattern by enabling the 'send-and-forget' approach. In this approach the sender which in this case is the LNMA does not have to wait for the GNMA to receive and process the message. The sender only needs to wait for the message to be sent and successfully stored in the messaging channel. Once the message is stored in the Middleware Layer, the LNMS can perform other tasks while the message is transmitted to the GNMA. The messaging service component acts as a universal communication point that allows the communication among remote systems that reside on different operating platforms and are written in different programming languages.

Synchronous communication on the other hand, can cause performance degradation and even cause the receiver to crash if too many calls are received on a single receiver. In contrast to synchronous communication where the caller must wait for the receiver to finish processing the call before the caller can receive the result in order to be able to continue, asynchronous communication has variable timing. The variable timing gives the ability to the LNMAs to submit requests to the GNMA in order for the messages to be processed at their own rate. This allows NMSs to run at maximum throughput and not having delays on waiting the messages to be processed by the GNMA.

- **Platform and Language Integration:** Communication in the NMP is based on XML messages and not on exchanging object data structures. The functionality of each service is abstracted and defined in an interface form, where other systems can use it and bind with it. The method calls are based on messages that are abstracted from the programming language that is used. As a result, management systems programmed in Java language can communicate with other management systems implemented in other languages.

- **Throttling**: Messaging services provide throttling. This is an important requirement in the design of the NMP due to the fact that the messaging service queues up requests until the receiver is ready to process them. The consumer is able to

control the rate at which it consumes requests so as not to become overloaded by too many simultaneous requests.

- **Reliability**: Furthermore, messaging service provides reliable delivery through the store-and-forward approach for transmitting messages that the messaging service supports. Management data is packaged as messages, which are atomic and independent units. When a LNMA sends a message, the messaging service in the Middleware Layer stores this message and it then delivers it by forwarding it to the GNMA. In addition, the provision of Persistent Messaging and Message Archiving Service also increase the reliability of message delivery.

This chapter has presented the design and the development of the Network Management Middleware Layer. The service components that have been created in order to provide middleware functions have been analyzed. The Middleware Layer of the NMP enables communication and transfer of management information between heterogeneous NMSs. The main contribution in this chapter is the design of a proposed Network Management Middleware Layer, which is the basis for developing the SOA-based NMP. The contribution includes

- the design of a messaging service that is a component, which allows the communication and data transfer from one management system to another.

- a persistence store that is designed for message persistence and which stores all messages. This service is used for recovering management data in case of a middleware failure.
- a validation service that is created for the purpose of validating messages received from heterogeneous NMSs. Validating messages eliminates the creation of unnecessary faults and errors by invalid messages.
- a transformation mechanism that will be responsible for dealing with different data formats is described. Taking into account the problem that arose from legacy systems, the NMP needs the transformation mechanism in order to be able to accommodate heterogeneous systems.
- a Routing Service has been created in order to minimize the interfaces and the dependencies among remote services as well as to provide intelligent routing rules for delivering the messages to the appropriate destination.
- finally, a Message Archive Service was designed in order for the messages that passes through the Middleware Layer to be stored into folders for reliability and inventory purposes.

Chapter 6 : IMPLEMENTATION, TESTING AND EVALUATION

6.1 Introduction

This chapter focuses on the implementation and evaluation of a global network management prototype based on the proposed SOA-based NGN management framework presented and designed in previous chapters.

The prototype development is divided into two phases:

- Software module development for the Core NMS Service Bus.
- Prototype development including the development of a Trouble Ticketing System (TTS) as an application.

Individual software modules for the Core NMS Service Bus are tested before integrating into the testbed. The global network management prototype is developed by integrating the Core NMS Service Bus with the TTS. The prototype as a whole is then tested and evaluated, subject to a set of test scenarios and evaluation criteria.

This chapter is organized as follows: section 6.2 presents the implementation architecture of the proposed Core NMS Service Bus. The Validation Service, Transformation Service and the Routing Service are explained. Next, a Trouble Ticketing System that is a part of the proposed Network Management Platform is presented. Finally, the testing environments as well as testing scenarios are illustrated, followed by an analysis and discussion.

6.2 Service Implementation in the Core NMS Service Bus

The Core NMS Service Bus is implemented using the open-source ESB. The Core NMS Service Bus that provides the proposed service components (Messaging Service, Message Validation Service, Message Transformation Service, Routing Service and Archive Service) has been implemented on the ServiceMix ESB platform [SERVICEMIX]. The overall Core NMS Service Bus architecture is depicted in the following figure (figure 6.1)

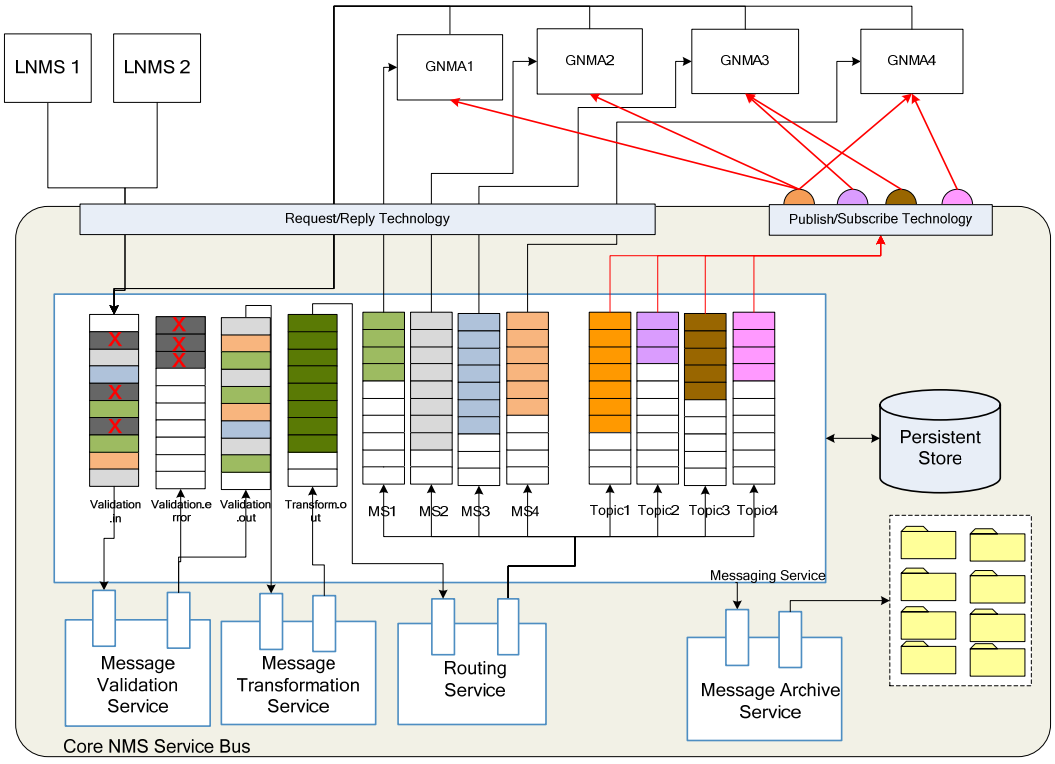


Figure 6.1: Developed Core NMS Service Bus

In figure 6.1, LNMS1 and LNMS2 are sending management information to the Core NMS Service Bus. Their management information is

encapsulated in XML-based messages that are processed by services in the Core NMS Service Bus. The destinations of the management information are four GNMAs each of which consumes a particular type of messages. The process order in the Core NMS Service Bus is as follows: first, messages are being validated, second, messages are being transformed, and lastly, messages are routed to queues and topics. The following subsection presents the proposed service components used for providing a dedicated middleware function.

6.2.1 Message Validation Service

6.2.1.1 Implementation Architecture

To demonstrate the validation process of the management messages, a Message Validation Application has been developed for such a purpose. This application is written in Java language and is based on the Swing framework [JSR296] Swing technology is used in order to provide a sophisticated GUI that is lightweight and independent of software platforms [JSR296]. Figure 6.2 illustrates the proposed implementation architecture for validating management messages. The communication between LNMSs and Core NMS Service Bus is based on queues where all NMS systems will make use of only one connection point in order to send management messages to the Core NMS Service Bus. Three message queues are created to provide asynchronous communication with the Message Validation Service:

- The *Validation.in* stores management messages created by NMSs

- The *Validation.out* stores the successfully validated management messages processed by the Message Validation Service
- The *Validation.error* message queue stores management messages that do not comply with the validation schema (Validation.xsd).

The Message Validation Application is connected to the Validation.in message queue and listens to the Validation.out and Validation.error messages queues.

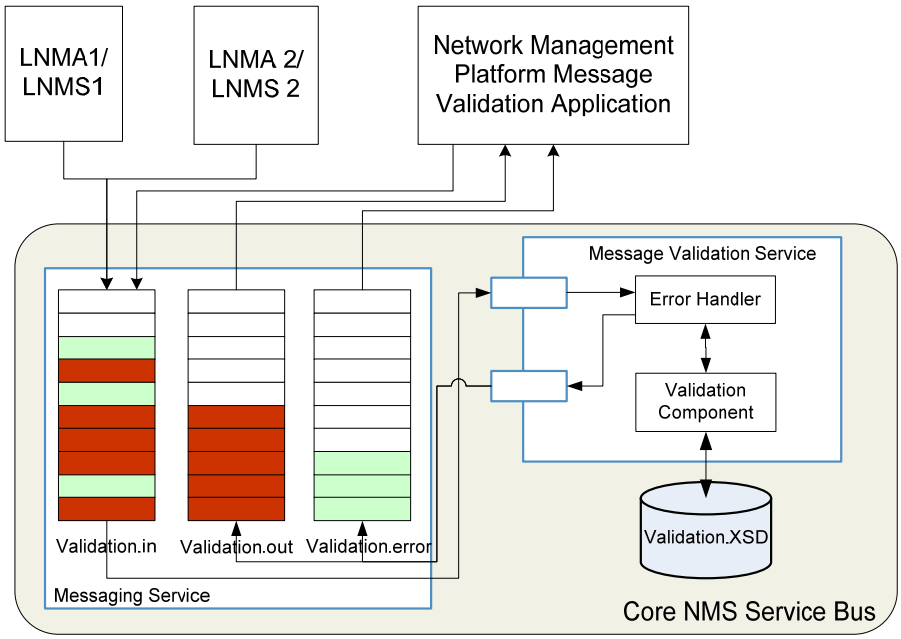


Figure 6.2: Implementation of the Message Validation Service

The development of the messaging queues is based on the JMS technology (JMS API) [SUNJMS]. In this scenario, LNMS1 and LNMS2 store management messages from LNMA1 and LNMA2 respectively into the Validation.in message queue. Additionally, the Message Validation Application can create XML-based messages and store them into the Validation.in message queue. This function has been developed

specifically for the testing scenario where each management message can be customized according to specific testing rules. For instance, instead of waiting for the LNMA1 and LNMA2 to create a valid or invalid management message, the application can produce valid and invalid management messages and inject them directly into the Validation.in message queue. Moreover, management messages can be viewed through the application's GUI.

6.2.1.2 Algorithmic Process for the Message Validation Service

Figure 6.3 illustrates the process that the Validation Service performs in order to distinguish valid and invalid management messages. A valid management message is the message that complies with the specified schema and an invalid management message is the message that does not comply with the schema.

Two functional components outline the Validation Service:

- Error Handler
- Validation Component.

The Error Handler is responsible for message exchange between the Validation Component and the message queues (Validation.in, Validation.out and Validation.error). The Validation Component decides whether a management message is valid or invalid. The decision is made by comparing the XML structure of the management message to the structure that has been defined by the XML schema. Java API for XML Processing offers an API for validating XML documents [JAXP]. JAXP Validation API [JAXP] (javax.xml.validation.*) that has been used for

developing the validation component, instantiates an object representation of a schema and uses it to validate one or more XML documents.

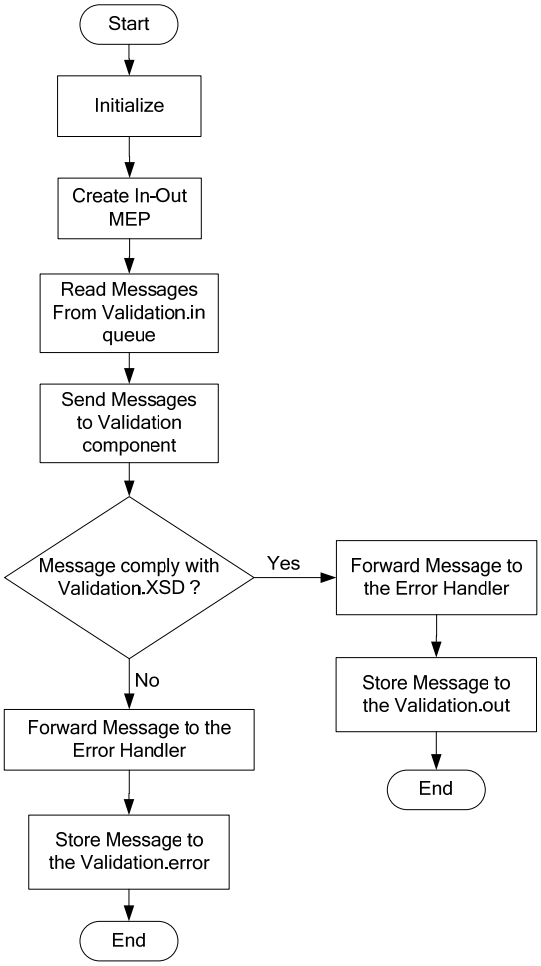


Figure 6.3: Process for validating management messages

Figure 6.4 demonstrates the initialization process of the Message Validation Service. First, the Message Validation Service is initialized. The Core NMS Service Bus command line shell indicates that the Message Validation Service has started and the Error Handler is active. Furthermore, the command line shell indicates the IP address of the Core NMS Service Bus where the Message Validation Service resides.

After the initialization process, the Error Handler creates an In-Out Message Exchange Pattern (MEP). The In-Out MEP is used to receive management messages from the Error Handler and to respond back. Next, the Error Handler copies the management messages from the Validation.in message queue and sends them to the Validation Component. The Validation Component parses the content of the management message.

If the content and syntax of the XML-based management message complies with the Validation.xsd schema then it forwards the message to the Error Handler and the Error Handler stores the management message to the Validation.out message queue. If the content and the syntax of the management message do not comply with the Validation.xsd schema then the Validation Component creates an error message that contains the parsing errors and forwards the message to the Error Handler. The Error Handler, which has an established In-Out MEP, stores the error message to the Validation.error message queue.

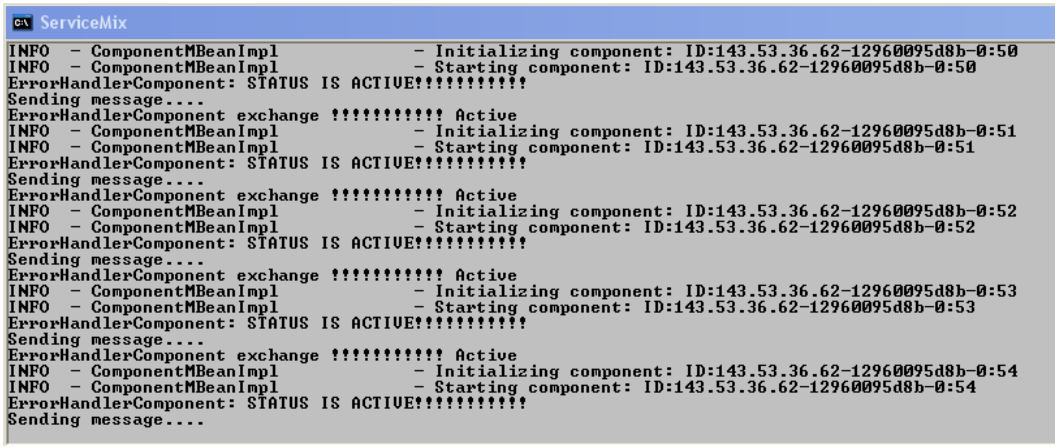


Figure 6.4: Message Validation Service, initialization process

6.2.2 Message Transformation Service

6.2.2.1 Implementation Architecture

Figure 6.5 illustrates the implementation of the Message Transformation Service. After management messages have been validated, the Validation.out messaging queue containing the successfully validated management messages becomes the input queue for the Message Transformation Service.

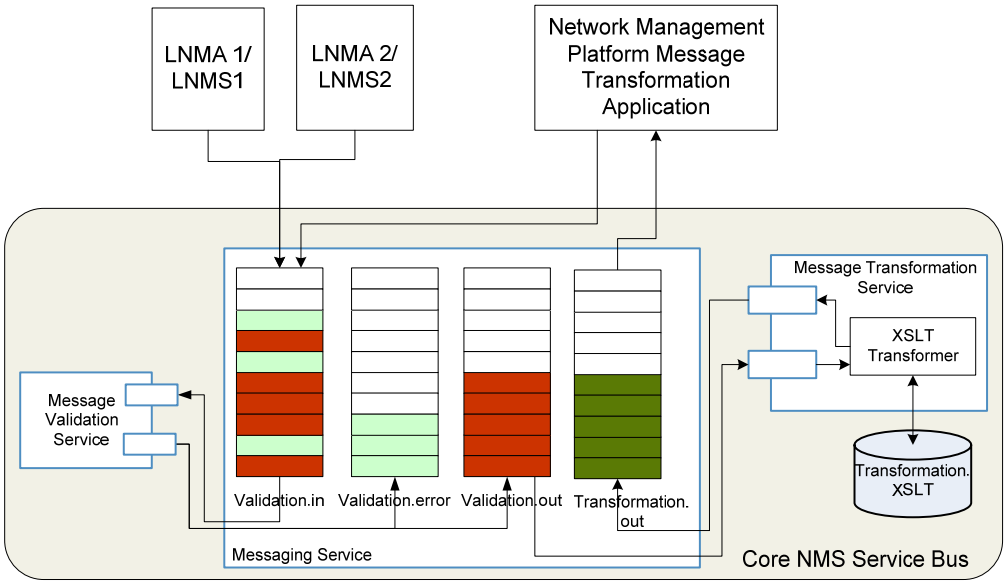


Figure 6.5: Implementation of the Message Transformation Service

The messaging service has created the Transformation.out messaging queue in order to store the transformed management messages. Moreover, an application has been developed in order to demonstrate the transformation process. This application is based on the Message Validation Application but has been modified to enable injection of management messages to the Validation.out messaging queue and read messages that have been stored in the Transformation.out messaging queue.

An XSLT Transformer component has been developed in order to perform the transformation function. The transformation function is performed by using the Saxon API [SAXON]. The latter is able to transform an incoming message based on XSLT stylesheet. Saxon provides an XSLT processor that takes as an input an XML document and stylesheet to convert the XML document to other formats. In the transformation function, the processor reads through the XML document tree, looking at each node in turn, and compares it with the pattern of each template rule in the stylesheet as described in chapter 5, section 5.3.3.1. When the processor finds a node that matches a template rule's pattern, it outputs the rule's template. After the transformation process, the management message is stored into the Transformation.out messaging queue.

6.2.2.2 Implementation Process

The management messages that have been stored in the Validation.out queue have been created by two LNMAs of different LNMSs with different data representation. This means that the messages sent to the Core NMS Service Bus are not homogeneous. Figure 6.6 illustrates the management information of an event that has been generated by LNMS 1 and an event generated by LNMS2.

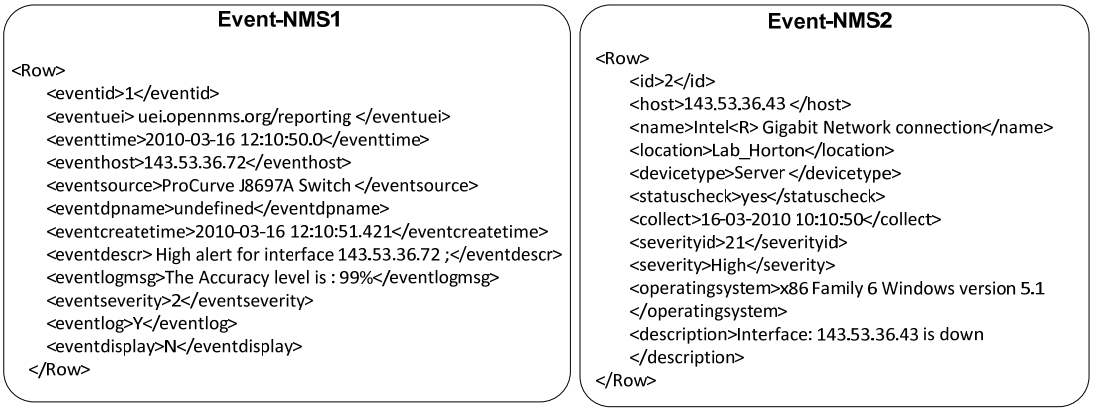


Figure 6.6: Events occurred in two different NMSs

The Validation.out messaging queue contains two types of management messages where each event is expressed by using different element names. For instance, in LNMA1 the IP address of the occurred event is encapsulated in an element with name <eventsource>, whereas in NMS2 the element that encapsulates the IP address of an event is <Host>. In order to have a common information model that can be understood by other management applications, such as Trouble Ticketing Systems, both events need to be translated into a common message format. The transformation rules that are used by the Message Transformation Service are contained into the transformation.xslt stylesheet.

The common Information model used in the Core NMS Service Bus is illustrated in figure 6.7. It represents only a subset of a standardized information model and it is used as a guideline for legacy information to be able to be expressed into a common model.

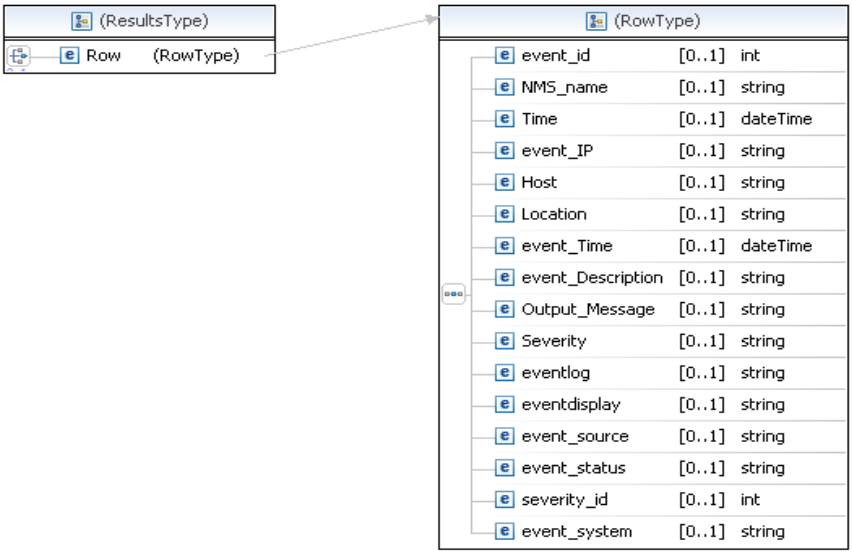


Figure 6.7: Common information model used for event mapping

SID [M.3190] can be used in order to provide the common information model in the Network Management Platform so that 'legacy' information be transformed into a standardized format. In this way, applications based on the NGOSS framework [NGOSS04] could be integrated in the Network Management Platform. SID specification, even if it is an open industry standard, it is not publicly available without a membership license fee [M.3190].

6.2.3 Message Routing Service

6.2.3.1 Implementation Architecture

The implemented Routing Service is based on the JAXP API [JAXP]. JAXP API deals with XML payload and provides XPath routing functions based on the content of an XML document [JAXP]. Figure 6.8 illustrate the components of the Core MS Service Bus and their relationships.

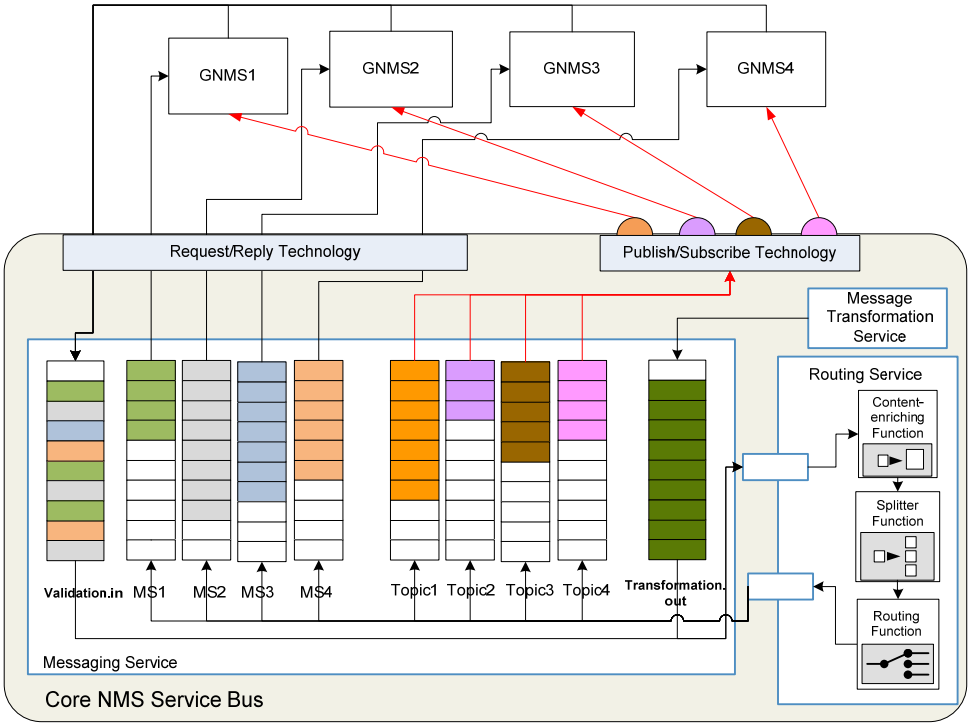


Figure 6.8: Implementation of the Routing Service

Both point-to-point and publish/subscribe communication patterns are used for communication between the LNMSs and the GNMA through the Core NMS Service Bus. With the publish/subscribe paradigm, four different topics (Topic1, Topic2, Topic3, and Topic4) have been defined for publishing different management information acquired from the local NMSs.

- Topic1 publishes critical event messages and configuration messages acquired from different NMSs. Critical events are related to fault management information (i.e. an interface is down). Moreover, the Topic1 topic publishes events related to configuration management. For instance, an interface is up, a server has been restarted, etc.

- Topic2 publishes events related to performance measurements acquired from NMS systems. These measurements are classified as minor events. Furthermore, this topic publishes events related to configuration management. For the implementation, two NMS systems are used (NMS1 and NMS2).
- Topic3 publishes information related to fault management, performance management and configuration management acquired from NMS1.
- Topic4 publishes event messages containing faults, performance and configuration measurements acquired from NMS2.

The publish/subscribe pattern based on topics has been proposed in order to completely decouple the GNMA from the LNMSs. The management information in each topic can be consumed by GNMAs. Each GNMA can now specify the type of management information that is required for them to process in order to perform their own functions. For instance, one or many customer care services from different service providers can connect to Topic2 in order to monitor and improve the QoS of their customers.

Furthermore, GNMAs are required to have a communication channel that will allow them to communicate with each other. The communication pattern used for inter-GNMA intercommunications is based on the point-to-point approach. The reason for choosing this approach is that the each GNMA should have a dedicated messaging queue in order to receive information from other GNMA. For this reason, four messaging queues have been created (MS1, MS2, MS3, and MS4).

6.2.3.2 Routing and Publishing Management Information

The content of the management message defines an element tag <severity></severity> that determines the severity level of the management information. Three types of severity levels have been defined: critical level, minor level and notifications. The critical events are dedicated to fault management indicating faults occurred in the network. The minor level events are concerned with performance measurements. Notifications are events depicting configuration parameters of the network.

For NMS1 the severity levels are:

- <Severity> 1 </Severity> for critical events
- <Severity> 2 </Severity> for minor events
- <Severity> 3 </Severity> for notifications

For NMS2 the severity levels are:

- <Severity> High </Severity> for critical events
- <Severity> Low</Severity> for minor events
- <Severity> information <Severity> for notifications

As described in section 6.2.2, the Transformation Service transforms the management messages into a common information model and stores them into the Transformation.out messaging queue. Routing Service consumes management messages from the Transformation.out queue and processes them. First, the enriching function adds content to the messages. This is performed due to the fact that in some messages the payload may not contain any information concerning the identity of the

NMS that they were extracted from. A solution is proposed to add a dedicated element tag in every management message payload indicating the origin of the message.

Management messages contain an XML namespace (xmlns) header indicating the origin of the message (figure 6.9). For instance, messages published by LNMA1 have namespace attribute http://esb.nms1.org and messages published from LNMA2 have namespace attribute http://esb.nms2.org.

```
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns="http://esb.nms1.org"></Results>
```

Figure 6.9: LNMS1 namespace

The content-enriching function parses the management message and if the namespace of the management message is http://esb.nms1.org then it inserts an element tag <NMS> with value 1. If the namespace is http://esb.nms2.org then the element tag will have value 2. This function is implemented in the processor interface of the JAXP API. The psudo code for this interface is illustrated below (figure 6.10):

```
Get namespaceURI

If (namespace.equals (“http://esb.nms1.org”)){
Normalized Message nms createMessage();
nms.setContent(new StringSource("<NMS>1</NMS>");
}

else if(namespace.equals (“http://esb.nms2.org”)){
Normalized Message nms createMessage();
nms setContent(new StringSource("<NMS>2</NMS>");
}

else{
 throw exception
}
```

Figure 6.10: enriching algorithm

The content-enriching function (figure 6.11) uses the JAXP Spring technology [JAXP] in order to call the processor interface. The following XML parameters have been implemented in order to define the processor's interface class (esb:DecisionPoint) and the service from which the content-enriching function receives the management messages (esb:TransformationService).

```
<content-enricher service="esb:ContentEnrichingFunction"
endpoint="EnrichingEndpoint">
  <enricherTarget>
    <exchange-target service="esb:DecisionPoint" />
  </enricherTarget>
  <target>
    <exchange-target service="esb:TransformationService" />
  </target>
</content-enricher>
```

Figure 6.11: content-enriching function

Each management message consists of multiple events. The splitting function splits the management message into messages that contain an individual event. Each <Result> parent element in the management message contains multiple <Row> elements and each <Row> element encapsulates an individual event. The following XPath expression is used for splitting the management message into multiple event messages.

```
<xpath-splitter service="esb:RRouter" endpoint="RRouterEndpoint"
xpath="/Results/Row" namespaceContext="#nsContext">
<target>
  <exchange-target service="esb:AppInput" />
</target>

<namespace-context id="nsContext">
  <namespaces>
    <namespace prefix="nms1">http://esb.nms1.org
    </namespace>
    <namespace prefix="nms2">http://esb.nms2.org
    </namespace>
  </namespaces>
</namespace-context>
```

Figure 6.12: splitting function

In figure 6.12, the expression /Results/Row splits all Row elements that are children of Results and forwards them to the content-routing function (target-service=”esb:AppInput).

The namespace context is used as an identifier in the management message and it is defined in the header of the management message, and since there are more than one LNMA sending messages to the Core NMS Service Bus, the namespace context indicates in which messages the functions will be performed. For instance, the splitting function is performed in management messages transmitted by both LNMS1 and LNMS2.

The routing function uses XPath expressions in order to route each message to the appropriate destination. Figure 6.13 shows the XPath code which illustrates the routing function’s decision part for routing event messages to Topic2. As stated earlier, Topic2 publishes events related to performance management for both LNMS1 and LNMS2. The predicate XPath expression denotes the rule to be applied while JAXP is parsing the

message. If the condition is true, then the output will be forwarded to the exchange-target service (esb:duplicate1).

```
<content-based-router service="esb:AppInput"
      endpoint="AppInputEndpoint">
<rules>
 <routing-rule>
  <predicate>
      <xpath-predicate
xpath="//nms1:severity='2'|xpath="//nms1:NMS='1'"
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:Duplicate1"></exchange-
target>
      </target>
 </routing-rule>
 <routing-rule>
  <predicate>
      <xpath-predicate xpath="//nms2:severity='Low'|
xpath="//nms2:NMS='2'"
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:Duplicate2"></exchange-
target>
      </target>
 </routing-rule>
<rules>
</content-based-router>
```

Figure 6.13: XPath Routing Rule

While Topic2 publishes events related to performance measurements acquired from NMS1 and NMS2, Topic3 publishes the same events but only from NMS1. Thus, two topics (Topic2 and Topic3) require the same message to be published. The event message needs to be duplicated; as a result, the XPath code as shown in figure 6.14 has been used in order to create a copy of the original event message and send it to two destinations (esb:Topic2 and esb:Topic3).

```
<wire-tap service="test:Duplicate1" endpoint="Dupendpoint1">
       <target>
             <exchange-target service="esb:Topic2" />
       </target>
      <inListener>
            <exchange-target service="esb:Topic3" />
      </inListener>
</wire-tap>
```

Figure 6.14: Duplicating messages

6.2.3.3 Process for Routing Management Message to Topics

Figure 6.15 shows the algorithmic process of the Routing Service that decides the destination of the management messages.

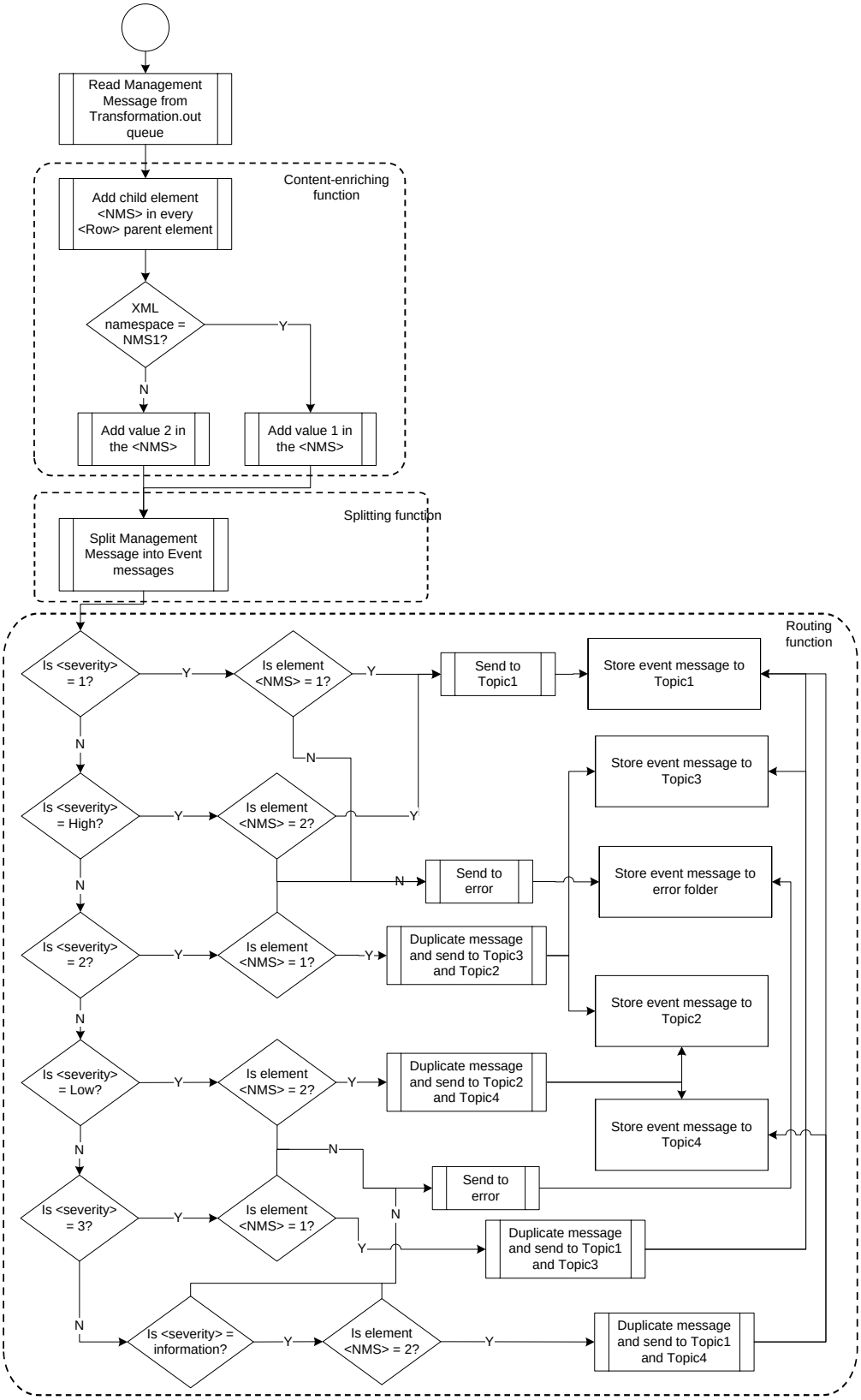


Figure 6.15: Process for routing management messages to Topics

1. Routing Service consumes management messages from the Transformation.out messaging queue.
2. The content-enriching function parses the management message and adds the element <NMS> </NMS> in every <Row> parent element. If the xmlns is http://esb.nms1.org the element <NMS> will have value of 1, if not, the value of <NMS> will be 2.
3. Next, management messages are split into event messages. Each event message contains information regarding an individual event occurred in the managed network. Each event contains an element that indicates the severity of the event.
4. After management messages have been split, routing function parses each event message and:
    - If the EM message has an attribute value of 1 in the severity element and if the value of the NMS element is set to 1, then the EM message is routed to Topic1.
    - If the EM message has an attribute value of High in the severity element and if the value of the NMS element is set to 2, the destination of the EM message is again Topic1.
    - If the severity element has an attribute value of 2 and the NMS element has value 1, then the EM message is duplicated and routed to Topic1 and Topic3.

- If the EM message has an attribute value of Low in the severity element and if the value of the NMS element is set to 2, then the EM message is duplicated and routed to Topic2 and Topic4.
- If the attribute value of severity element is 3 and if the value of the NMS element is set to 1, then the EM message is routed to Topic1 and Topic3.
- If the attribute value of severity element is information and if the value of the NMS element is set to 2, then the EM message is routed to Topic1 and Topic4.
- If the predefined severity values or the NMS values do not exist in the EM then the message is stored into an error folder in the Core NMS Service Bus.

6.2.3.4 Process for Management Service Inter-communication

Even though management services subscribe to one or all topics for acquiring management information, it is necessary for them to be able to communicate with each other. Topics are publishing messages to the subscribed applications, this means that the same message is multiplied and 'pushed' to the registered destinations.

To establish communication among GNMAs, four queues have been developed one for each GNMA. These queues are uni-directional, meaning that they can only receive information. For sending messages, the services use the Validation.in messaging queue as depicted in figure 6.8. In every request/reply interaction, the application components of GNMA are required to declare a unique namespace that will be included in

each message request. For example, xmlns=”http://esb.management.service1.org is included in every message request by GNMA1. The XML namespace can be used to identify the origin of the message so that each service component in the Core NMS Service Bus is able to differentiate the GNMA components. Message requests and replies do not need to be validated or transformed as they would have already conformed to a common information model. The xmlns is used as a rule for excluding the messages from being processed by the validation service and the transformation service. Hence, messages created by Management Services can bypass the validation and transformation processes. To route the messages to the appropriate queue, GNMAs need to indicate in the message, the recipient’s intended destination. This will allow the Routing Service component to process each message and route it to a queue.

An element tag needs to be defined in each message (<Destination></Destination>) indicating the recipient. Four values are specified for the destination element:

- <Destination>MS1</Destination> for GNMA1 destination
- <Destination>MS2</Destination> for GNMA2 destination
- <Destination>MS3</Destination> for GNMA3 destination
- <Destination>MS4</Destination> for GNMA4 destination

The Routing Service component consists of three functions as stated before. Content-enriching and splitting functions are not required to be implemented for messages exchanged among GNMAs, Thus, they need

to be bypassed. The prefix values (i.e. Service1, Service2 etc.) bind a particular rule function to a message that has the appropriate namespace URI. In other words, it instructs the routing function to apply specific XPath rules only to messages that have the approved xmlns URI. The XML code in figure 6.16 illustrates the prefixes as well as the xmlns URIs used in the routing function for the purpose of routing messages to MS1, MS2, MS3 and MS4 queues.

```
<namespace-context id="nsContext">
  <namespaces>
    <namespace prefix="Service1">http://esb.managementservice1.org
    </namespace>
    <namespace prefix="Service2">http://esb.managementservice2.org
    </namespace>
    <namespace prefix="Service3">http://esb.managementservice3.org
    </namespace>
    <namespace prefix="Service3">http://esb.managementservice4.org
    </namespace>
  </namespaces>
</namespace-context>
```

Figure 6.16: Namespace prefixes for the GNMAs

The XML code in figure 6.17 illustrates the XPath rules applied for routing messages to the queues.

```
<rules>
 <routing-rule>
  <predicate>
      <xpath-predicate xpath="//Service1:Destination='MS1'
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:MS1"></exchange-target>
      </target>
 </routing-rule>
 <routing-rule>
  <predicate>
      <xpath-predicate xpath="// Service2:Destination='MS2'
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:MS2"></exchange-target>
      </target>
 </routing-rule>
```

```
<routing-rule>
  <predicate>
      <xpath-predicate xpath="//Service3:Destination='MS3'
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:MS3"></exchange-target>
      </target>
 </routing-rule>
<routing-rule>
  <predicate>
      <xpath-predicate xpath="//Service4:Destination='MS4'
      namespaceContext="#nsContext" ></xpath-predicate>
  </predicate>
      <target>
            <exchange-target service="esb:MS4"></exchange-target>
      </target>
 </routing-rule>
<rules>
```

Figure 6.17: Routing rules for GNMA intercommunication

The Routing Service differs from classical routing methods in the sense that management messages are addressed based on their content instead of their destination. In conventional systems [BALD05], the sender explicitly specifies the intended message recipients using either a unicast address in the case when the recipient is one or a multicast address in the case when there are many recipients. Instead, in the Network Management Platform, the sender simply injects the management messages in the network, and the Routing Service determines how to route the management messages according to the recipient's (Management Service) interests. Therefore, the Middleware Layer determines the message delivery and not the senders. Routing Service results in a more optimal solution than conventional routing in the sense that routing is dynamically reconfigured in one location and not in every LNMS. LNMSs are focused on providing management functionality and not performing routing algorithms that will couple them to specific clients.

Furthermore, interfaces among application clients are less when Routing Service is used. Finally, routing algorithms could be easily updated and adjusted according to the destination's needs. Routing Service component allows the Network Management Platform to be scalable for future needs. For instance, when a new NMS needs to be added into the platform, the Routing Service is in charge of providing the routing rules for deciding the message destination and not the services.

For storing the messages into folders, the Message Archive Service creates folder destinations where each message can be stored. The folders are located in the Core NMS Service Bus and via FTP, remote access can be achieved. A duplicate method has been defined in the routing rule in order to duplicate each message before it is sent to the topic or queue. The following XML code (figure 6.18) illustrates the wire tap method used for duplicating messages. The figure shows only one of the ten folders created in the Core NMS Service Bus. The messages sent to Topic1 are also sent to Folder 1. A destination directory has been created (NMS1_F) for storing the each message in an XML-based file.

```
<rule:wire-tap service="esb:wireTap6" endpoint="wireTapendpoint6">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder1" />
  </rule:inListener>

<file:sender service="esb:Folder1" endpoint="folder1Endpoint"
            directory="file:NMS1_F"></file:sender>
```

Figure 6.18: Archive message duplication and destination of the message

Appendix E contains the Core NMS Service Bus, the routing service's routing rules.

6.3 Implementation of the Global Trouble Ticketing System (TTS)

6.3.1 Implementation Architecture

TTS is a sophisticated application providing a number of tools related to processing, categorizing and presenting the tickets. Tickets are incidents that have been created in a network [GREE01]. TTS is used for network management purposes, as well as for other business-related functions; for example in many organizations in order to resolve reported customer issues or even issues reported by the organization's employees [GREE01]. TTS acts like a hospital chart, coordinates the work of multiple people who may need to work on the problem and aids the Network Operations efficiency [JOHN92]. The functions that TTS performs are as follows:

- It acts as a short-term memory about the specific problems for the Network Operation Center (NOC) as a whole.
- It provides real time lists of open problems, sorted by priority and allows network operators to keep track of the current NOC workload.
- It is useful for statistically analyzing equipment and NOC performance.

TTS is effective and efficient if it is integrated with network monitoring systems for alert, with electronic mails for notification, and with other TTSs for a global view of network status.

To evaluate the performance of the Core NMS Service Bus, four different TTSs have been developed, each acting as the message consumer in the MOM technology context. Using a loosely coupled SOA implementation, the systems have been integrated with the Core NMS Service Bus that connects with different LNMS. Each TTS handles a specific event and sends out e-mail notifications to network operators and high-end customers who are concerned with the performance of the network service. The TTS's architectural design consists of three main parts:

- Connectivity and Message Consumption part that connects the Core NMS Service Bus with the TTS and how to consume the messages.
- Presentation part that presents the application's information through a web-browser
- Application logic for creating and sending e-mails.

TTSs use the publish/subscribe paradigm to subscribe to any topics related to management information (i.e. performance and/or fault management etc.) that they are interested in. Moreover, TTSs use the point-to-point paradigm and message queues for information exchange as depicted in figure 6.19.

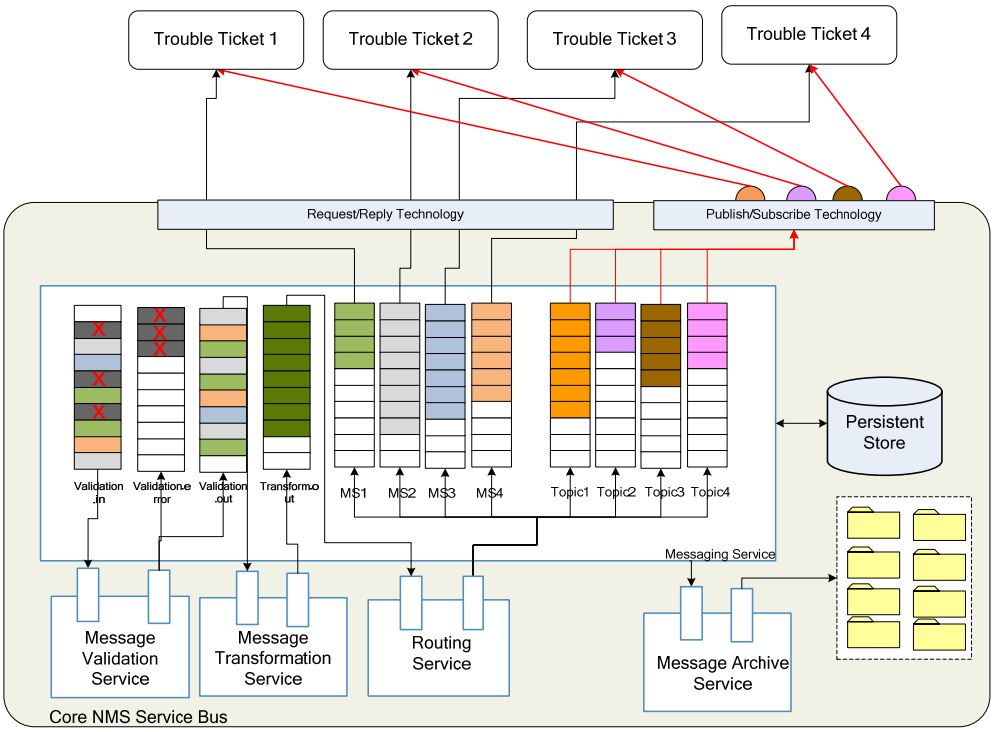


Figure 6.19: Trouble Ticket System integrated with Core NMS Service Bus

In the basic model that was designed in chapter 5, four topics are created:

- Topic1 contains fault- and configuration- related management messages required for the GNMAs (i.e. TTSs)
- Topic2 contains performance management messages required for the GNMAs (i.e. TTSs).
- Topic3 contains messages required by LNMS1 (i.e. Local Network Planning, Local Provisioning, etc.).
- Topic4 contains messages for LNMS2 (i.e. Local TTSs, Statistics, etc.).

These four different topics provide critical information about the network status, i.e. fault and configuration both local and global. Although the topics can also be classified according to Fault, Configuration, Accounting, Performance and Security (FCAPS) functions. Furthermore, in a large-

scale information processing environment, content-based messaging system can provide more choices. Content-based router examines the message content and routes the message onto a different channel based on data contained in the message. Routing can be based on a number of criteria such as existence of fields, specific field values. However, implementation will be much more complex when compared to topic-based messaging. Thus, topic-based messaging is used for the implementation of the TTS.

6.3.2 Implementation of TTS with J2EE

J2EE is a development for Java enterprise applications, which provides a powerful set of APIs in order to reduce the development time and the application complexity and to improve the performance of the applications. Java EE platform uses a distributed multi-tier application model for enterprise applications. That is, in a J2EE multi-tier environment, each part of the application can run on a different platform or node [J2EE]. The following figure (figure 6.20) illustrates the Java Enterprise Edition multi-tier architecture.

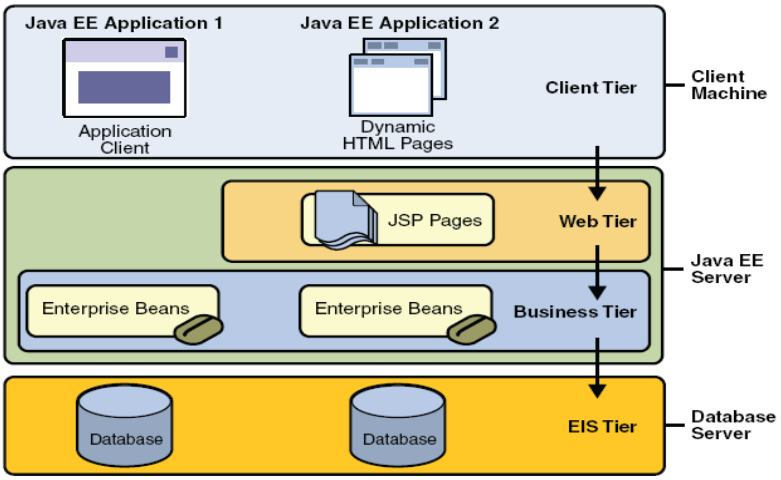


Figure 6.20: J2EE multi-tier architecture [J2EE]

The Java EE platform consists of four tiers:

- Client-tier: Provides components that run on the client machine. Examples of a client-tier are the web clients (dynamic web pages containing various markup languages such as HTML and XML), applets, or application clients.
- Web-tier: Runs on the Java EE server. The components are servlets and JavaServer Pages (JSP). Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that are executed as servlets but allow a more natural approach for creating static content.
- Business-tier: This tier provides the business code, which is the logic that solves a particular business domain. It receives data from client programs, processes it and sends it to the Enterprise Information System-tier for storage and vice versa. The business-tier resides on the server side.

- Enterprise Information System-tier (EIS): EIS includes database systems and other legacy information systems. The application components might need access to enterprise information systems for database connectivity.

For the development of the TTS, the following J2EE technologies have been deployed: Enterprise Java Beans (EJB), JMS, JSP with Servlet, JavaMail 1.3 and MySQL, as can be seen in table 6.1.

Table 6-1: Technologies for Trouble Ticketing System

Enterprise Java API	Applications in J2EE for GTTS
Enterprise Java Beans (EJB)	A server-side model that encapsulates the business logic of an application. Provide an infrastructure for creating, hosting and accessing server-based, distributed business components.
JMS 1.1 API	Provide reliable point-to-point and publish/subscribe messaging. Communicate with the Core NMS Service Bus.
JavaServer Pages(JSP)	enables Java inside HTML pages i.e. adding dynamic content. [JSP].
Servlet	A protocol for a java class to respond to HTTP requests. Provides a concise mechanism for creating and accessing web-based applications that are server and platform independent [SERV].
JavaMail 1.3	Provide complete support for accessing; creating and sending e-mail messages using IMAP, POP and SMTP protocols [JMAIL]
MySQL 5.1	Storing and persisting management messages.

In particular, Enterprise Java Bean (EJB) [EJB] was chosen for the implementation of the TTS. EJB was chosen since it uses JMS API for establishing communication with queues and topics in the Core NMS Service Bus [SUNJMS]. Two Java-bean classes (ticket.class and email.class) form the business-tier of the TTS. Figure 6.21 illustrates the relationship of the implemented classes.

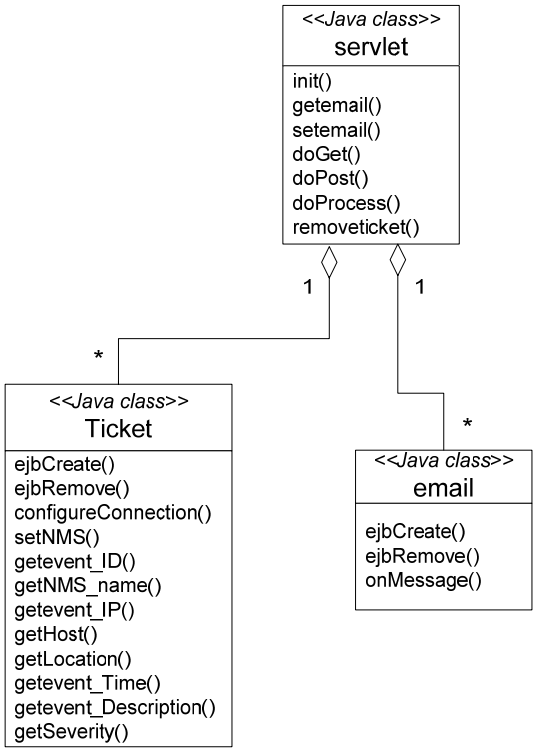


Figure 6.21: Application's classes and relationships

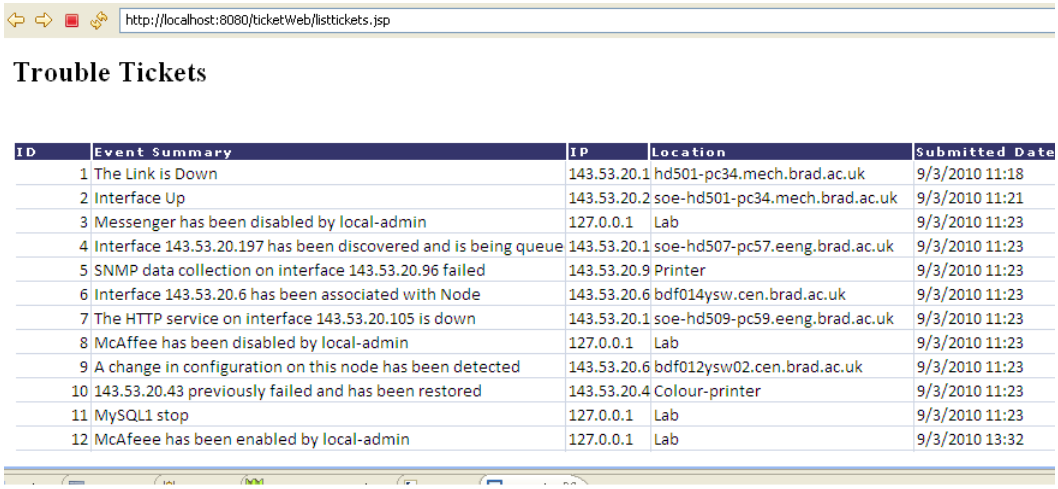
The ticket.class reads the management information from the Core NMS Service Bus by subscribing to a topic through the use of Java 'getters' [VALE99]. By adding setter and getter methods, the state of the managed bean is made accessible. The configureConnection method contains the connection details of the subscribed topic and queue.

The email.class provides the onMessage method, which contains the business logic for creating emails. JavaMail API classes have been used in order to establish connection with an e-mail server, create the email and transport it to the specified address. ejbCreate and ejbRemove methods have been used by both classes in order to insert and delete data from the database. These methods use SQL expressions for inserting and deleting objects from the database.

The servlet class performs doGet and doPost methods that are called in response to an HTTP GET and an HTTP POST respectively, which are submission methods used in HTML form. They have been used in combination with the doProcess method in order to send and receive requests from the trouble ticketing system's web page. The user interface of the TTS is implemented on a JSP web page as it is illustrated in figure 6.22. Moreover, the 'get' and 'set' methods are exposed via a web service in order to be able to exchange data with other applications and not only with the Core NMS Service Bus.

The WSDL file implemented as a service contract can be seen in Appendix E. The service contract is used in order to expose the binding parameters required to be known by other applications in order to communicate with the TTS.

Figure 6.22 illustrates the TTS's main user interface. It consists of two html web pages. The main page presents the event messages received from the Core NMS Service Bus and the second page creates tickets and assigns the person responsible to resolve the event (figure 6.23). In figure 6.22 the list of events is presented on the JBoss application server. These events have been captured by both LNMS1 and LNMS2.



ID	Event Summary	IP	Location	Submitted Date
1	The Link is Down	143.53.20.1	hd501-pc34.mech.brad.ac.uk	9/3/2010 11:18
2	Interface Up	143.53.20.2	soe-hd501-pc34.mech.brad.ac.uk	9/3/2010 11:21
3	Messenger has been disabled by local-admin	127.0.0.1	Lab	9/3/2010 11:23
4	Interface 143.53.20.197 has been discovered and is being queue	143.53.20.1	soe-hd507-pc57.eeng.brad.ac.uk	9/3/2010 11:23
5	SNMP data collection on interface 143.53.20.96 failed	143.53.20.9	Printer	9/3/2010 11:23
6	Interface 143.53.20.6 has been associated with Node	143.53.20.6	bdf014ysw.cen.brad.ac.uk	9/3/2010 11:23
7	The HTTP service on interface 143.53.20.105 is down	143.53.20.1	soe-hd509-pc59.eeng.brad.ac.uk	9/3/2010 11:23
8	McAffee has been disabled by local-admin	127.0.0.1	Lab	9/3/2010 11:23
9	A change in configuration on this node has been detected	143.53.20.6	bdf012ysw02.cen.brad.ac.uk	9/3/2010 11:23
10	143.53.20.43 previously failed and has been restored	143.53.20.4	Colour-printer	9/3/2010 11:23
11	MySQL1 stop	127.0.0.1	Lab	9/3/2010 11:23
12	McAfeee has been enabled by local-admin	127.0.0.1	Lab	9/3/2010 13:32

Figure 6.22: Global TTS subscribed to Topic 1

Each of these events can be sent to an email in order to inform the person who is responsible to fix the particular fault.

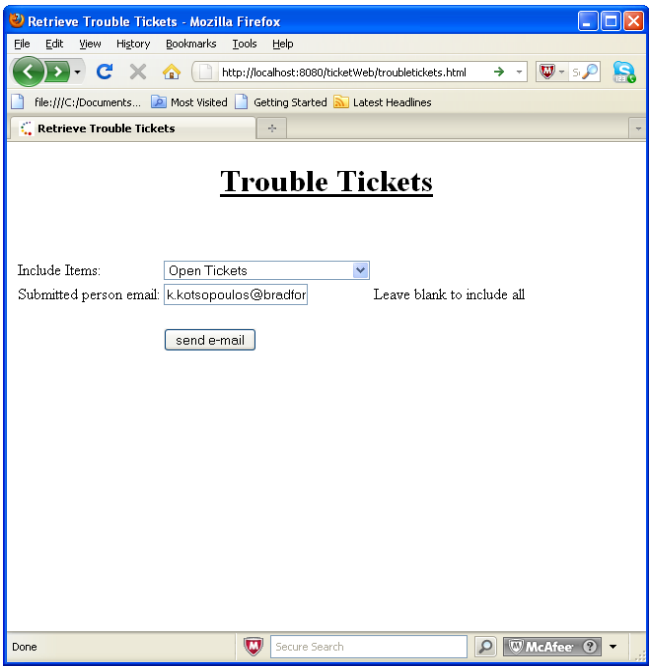


Figure 6.23: User Interface of the Trouble ticketing system

The TTS has been used as a Management Service. Four TTSs are hosted in separate PCs, each of which has a dedicated subscription to a particular type of management information. Thus, four instances of the

same TTS have been implemented, each one residing on different host. The above figure illustrates only one of the three instances.

6.4 Test Procedure

6.4.1 Testing Environment

Testing is performed on different platforms. Since Java is a cross-platform technology [VALE99], it can be embedded on different operating systems such as Windows platforms, Solaris, Linux etc. therefore, Core NMS Service Bus can run on every platform.

The test environment involves the following software:

- Eclipse Integrated Development Environment (IDE) tool has been used for compiling and executing the Core NMS Service Bus [ECLIPSE]: Eclipse is an open source multi-language software development tools with extensible plug-in system.
- Application Server: JBoss application server 4.2 functions as a Java EE application server and deploys EJB requiring JDK [JBOSS].
- Java Development Kit (JDK): JDK 1.6 [JDK].
- Jconsole is used to observe information about an application running on the Java platform [JCON].
- MC4J console to create a visual management application for Java servers. It supports connections to all major J2EE application servers with the feature of register and track notifications [MC4J].

6.4.2 Software Module Tests

6.4.2.1 Tests for Message Validation Service

Tests have been carried out through the Message Validation Service application. The application's Graphical User Interface is illustrated in figure 6.24. The user interface is split into two areas. The GUI's area in the red premises is the Network Management Platform Input area and the blue area is the Network Management Platform Output area.

The input area has two tabs: Msg-1 and Msg-2. Msg-1 tab is developed for creating and sending valid management messages to the Validation.in message queue and the Msg-2 tab is developed for creating and sending errored management messages to the Validation.in message queue. Similarly, the output area has two tabs: Rec-1 and Rec-2. Rec-1 tab reads the management messages that have been successfully processed by the Message Validation Service and have been stored into the Validation.out message queue. Rec-2 tab reads the errored management messages stored in the Validation.error message queue.

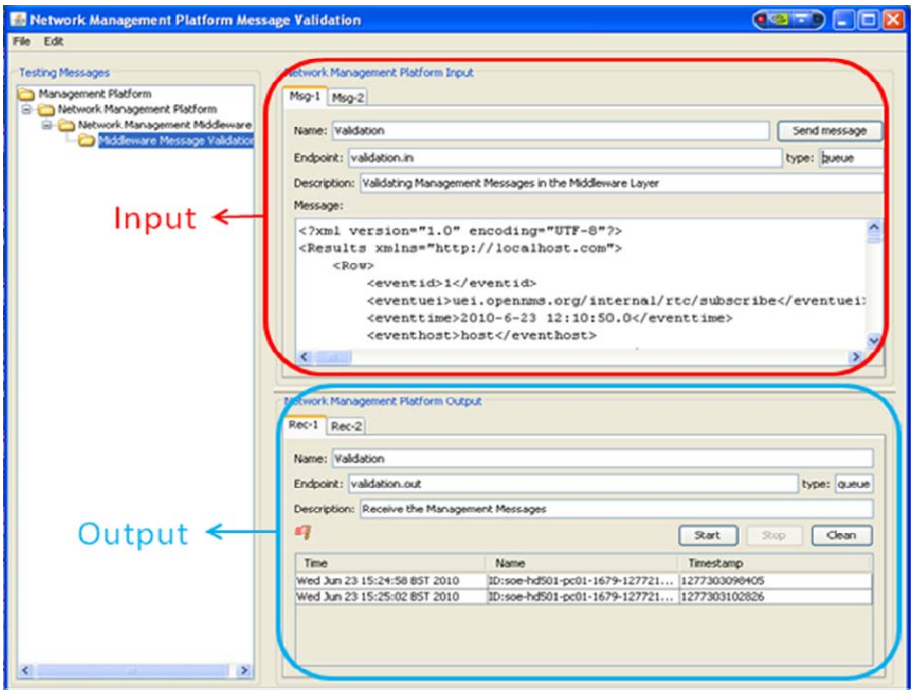


Figure 6.24: Message Validation GUI, Valid Message Console

The valid management messages are stored in sequence in the Validation.out message queue. The messages have a timestamp indication to indicate the time that the message was processed by the Validation Service. Furthermore, a unique name id has been injected to each management message serving as a message identifier.

6.4.2.1.1 Test Scenario 1: Validation of a Valid Management Message

Figure 6.25 shows a valid management message that can be viewed by using the Message Validation Application. The validated management message is extracted from the Validation.out messaging queue. It can be seen that the valid management message agrees upon the Validation.xsd schema and is placed in the appropriate messaging queue.

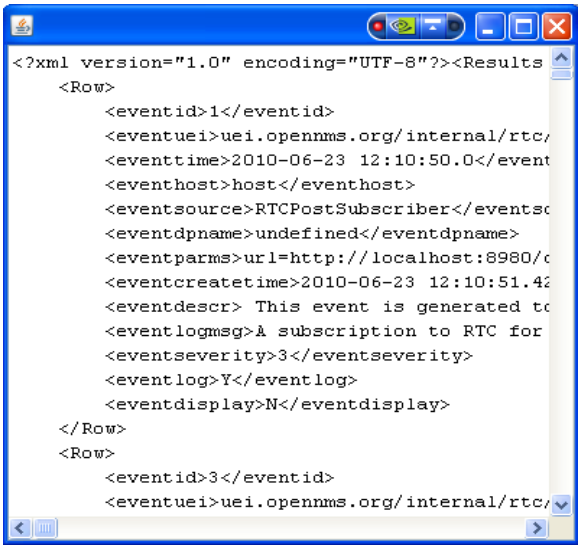


Figure 6.25: Valid management message

6.4.2.1.2 Test Scenario 2: Validation of an Errored Management Message

Figure 6.26 shows the Invalid Message Console of the Message Validation Application. In the input area, a management message with errors is injected in order to test whether the Validation Service can process the error management message correctly.

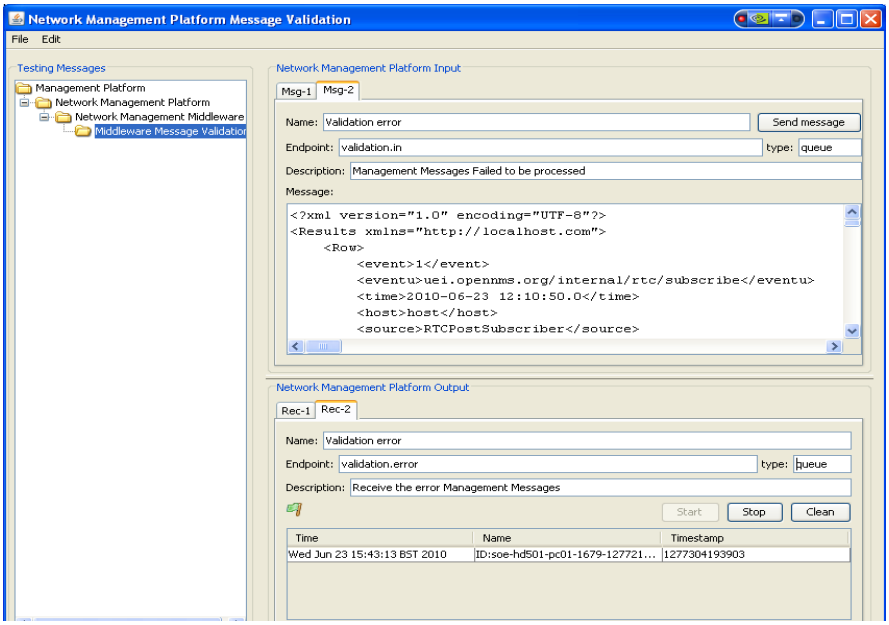


Figure 6.26: Message Validation Application GUI, Invalid Message Console

The errored management message does not have errors concerning the syntax of the document. Since this message contains error, it will not be understood by the GNMAs.

Figure 6.27 illustrates the invalid management message that is stored into the Validation.error messaging queue. The invalid message indicates that the content of the message has errors, which are also presented in that message.

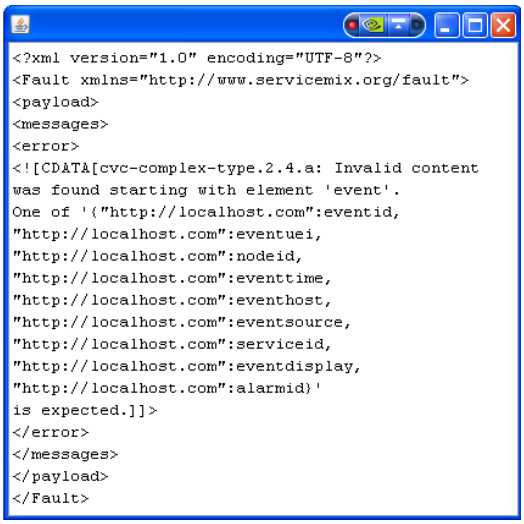


Figure 6.27: Invalid management message

6.4.2.2 Tests for Message Transformation Service

The initialization of the Message Transformation Service is depicted in figure 6.28. The Core NMS Service Bus command line shell indicates that the Message Transformation Service has started and the XSLT processor (XSLT Transformer) is waiting to consume messages.

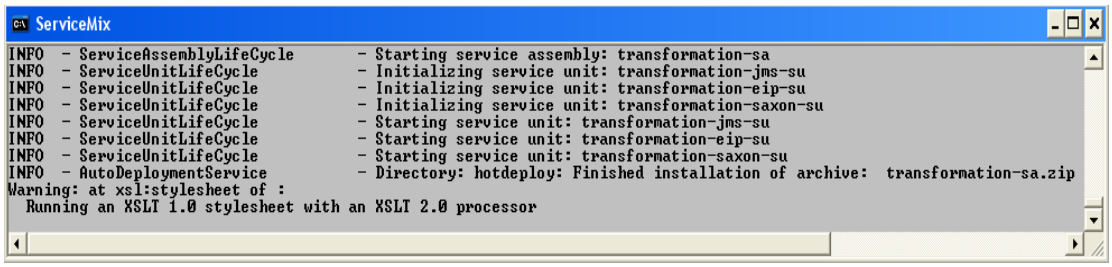


Figure 6.28: Message Transformation Service, initialization process

To test the Transformation Service component, a Message Transformation application has been developed in order to inject management messages to the Validation.out messaging queue and to read the Transformation.out queue. Figure 6.29 illustrates the Message Transformation application.

A management message is sent to the Validation.out message queue. The message contains multiple events concerning faults and performance measurements generated from the LNMSs. The events described in this management message follow the representation presented in figure 6.29. Each Row element (<Row> </Row>) contains information regarding a certain event that occurred in the network. The management message can be seen in the Input area of the transformation application.

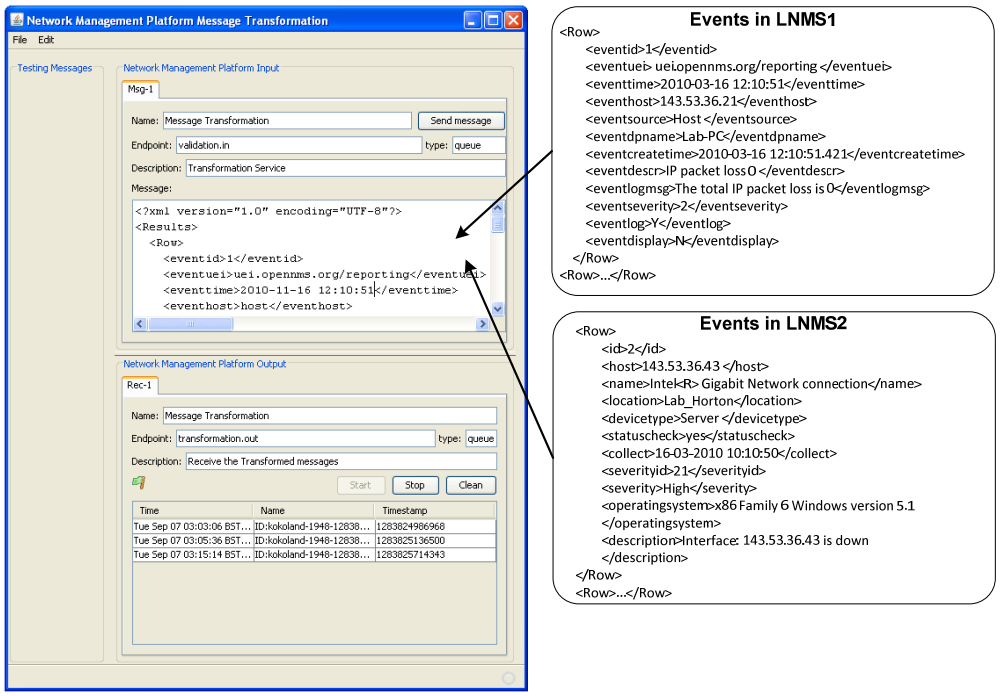


Figure 6.29: Message Transformation GUI, LNMS1 and LNMS2

The transformed management message is stored in the Transformation.out messaging queue and is read by the Message Transformation application. The transformed management message is shown in the figures 6.30.

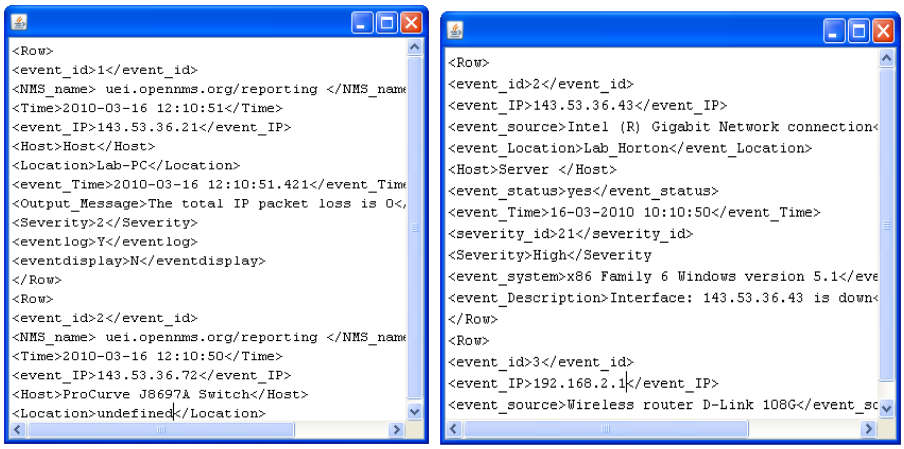


Figure 6.30: Transformed management messages from LNMS1 and LNMS2

The information of the management message generated by LNMA1 and LNMA2 is transformed into a common information model as illustrated in the figures above, demonstrating the capability of the Message Transformation Service for solving the heterogeneity of the management information by transforming the information into a common data format. A user application in the GNMA can use this common information model provided by the Core NMS Service Bus to process the management information obtained from LNMA1 and LNMA2. With the Transformation Service implemented in the Core NMS Service Bus adopting a common information model can alleviate the processing load of transformation in the GNMA. The benefit of this approach is even more noticeable when the Core NMS Service Bus needs to accommodate numerous heterogeneous LNMSs that have different data representation. In this way, GNMA can be kept lightweight and can concentrate on carrying out the network management functions that it is supposed to perform.

6.4.2.3 Tests for Message Routing Service

The final stage of the message process is the Routing Service. Each event message needs to be sent to the appropriate topic. The routing algorithm defined in section 6.2.3.2 is tested in this section. LNMS1 and LNMS2 are sending messages to the Core NMS Service Bus, the messages are first enriched, split into individual messages each of which contains one event and finally routed to the appropriate topic according to the rules defined previously. In order to demonstrate the process, Hermes tool [HERMES] has been used in order to capture the messages contained into the topics.

As explained earlier, four topics have been developed in the Core NMS Service Bus. Each topic accepts event messages for a particular event. Figure 6.31, illustrates the Hermes GUI, capturing the messages from the topics. In this figure, each column contains the event messages from each topic (Topic1, Topic2, Topic3, and Topic4). Moreover, in each column, the body of the event message can be seen. As can be seen, event messages are stored in the topics as defined in the routing algorithm.

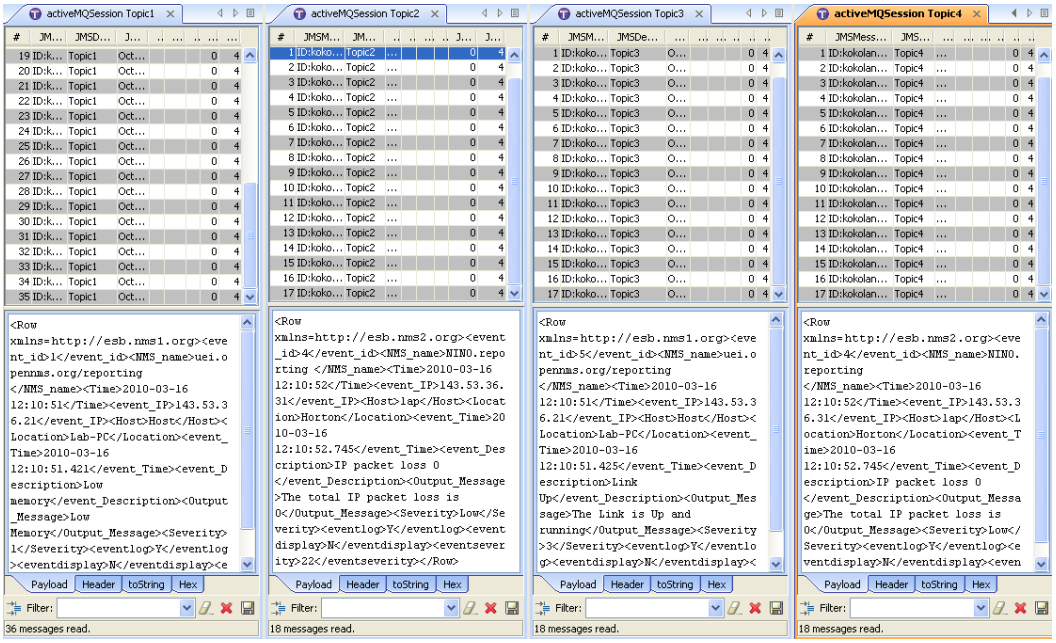
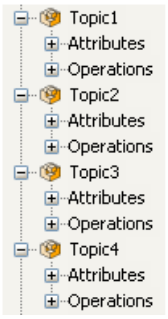


Figure 6.31: events captured by Hermes Software

In addition, Jconsole [JCON] has been used in order to demonstrate that topics are filled with messages. Figure 6.32 depicts the number of messages stored in each topic. In more detail, the AverageEnqueueTime shows the average time that the messages are stored in the topic before they sent to the destination. ConsumerCount represents the number of subscribers in the specific topic. DequeueCount represents the number of messages removed from the topic. DispacheCount shows the number of

messages sent to the subscriber correctly. The EnqueueCount represents the messages stored in the topic. As can be seen in the figure, Topic 1 received 113 messages. The 113 messages are left the topic and 113 messages are received by the subscriber.



Topic1	AverageEnqueueTime	2.495575221238938
	ConsumerCount	1
	DequeueCount	113
	DispatchCount	113
	EnqueueCount	113
Topic2	AverageEnqueueTime	0.8727272727272727
	ConsumerCount	1
	DequeueCount	55
	DispatchCount	55
	EnqueueCount	55
Topic3	AverageEnqueueTime	1.6785714285714286
	ConsumerCount	1
	DequeueCount	14
	DispatchCount	14
	EnqueueCount	14
Topic4	AverageEnqueueTime	4.836363636363636
	ConsumerCount	1
	DequeueCount	63
	DispatchCount	63
	EnqueueCount	63

Figure 6.32: Topic detailed measurements

6.4.3 Testbed for the NGN Management Prototype platform

6.4.3.1 Testbed Set up and Objectives

Figure 6.33 illustrates the testbed architecture used for testing the Network Management Platform. The Core NMS Service Bus has a dedicated server that runs on Windows XP Professional. The server is equipped with 3 GHz CPU processor and 3 GB system memory. The two LNMSs as well as the TTSs run on PCs with 2 GHz processors and 1 GB memory each. Each TTS represents an application on a GNMA and each is connected to the Core NMS Service Bus's 61616 port in order to send and receive messages to/from the queues and topics.

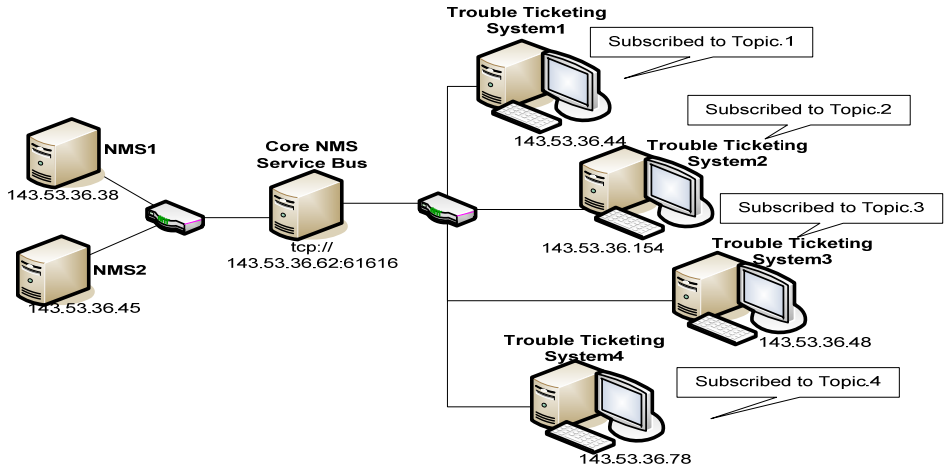


Figure 6.33: Testbed architecture

The objectives of the testbed are:

- To validate the correct operation of the Core NMS Service Bus
- To measure the performance and examine the behaviour of the Core NMS Service Bus

Pre-integration tests on individual software modules for the message validation service, message transformation service and message routing service were carried out as described in the previous sections. Connections between each equipment were also tested during the integration process. Tests, each lasting for 100 seconds, were conducted in several stages as follows:

1. *The First Stage*: A large amount of management information was generated by the LNMSs and was stored in their databases respectively. This allows the construction of different XML-based message sizes to be sent to the Core NMS Service Bus.
2. *The Second Stage*: The LNMSs ran at full capacity, i.e., they sent management messages as fast as possible to the Core NMS Service

Bus. Once the Core NMS Service Bus became overloaded, the messages would be stored in queues.

3. *The Third stage*: The messages that go through the Core NMS Service Bus were captured and calculated. MC4J console and Jconsole have been used in order to capture the messages as well as to calculate the throughput of the Core NMS Service Bus under different scenarios. For verification purposes, the measurements are repeated several times.

6.4.3.2 Validation of Core NMS Service Bus Functions

Figure 6.34 illustrates a typical message transaction between a LNMS, the Core NMS Service Bus and the TTS, which is used in carrying out the tests.

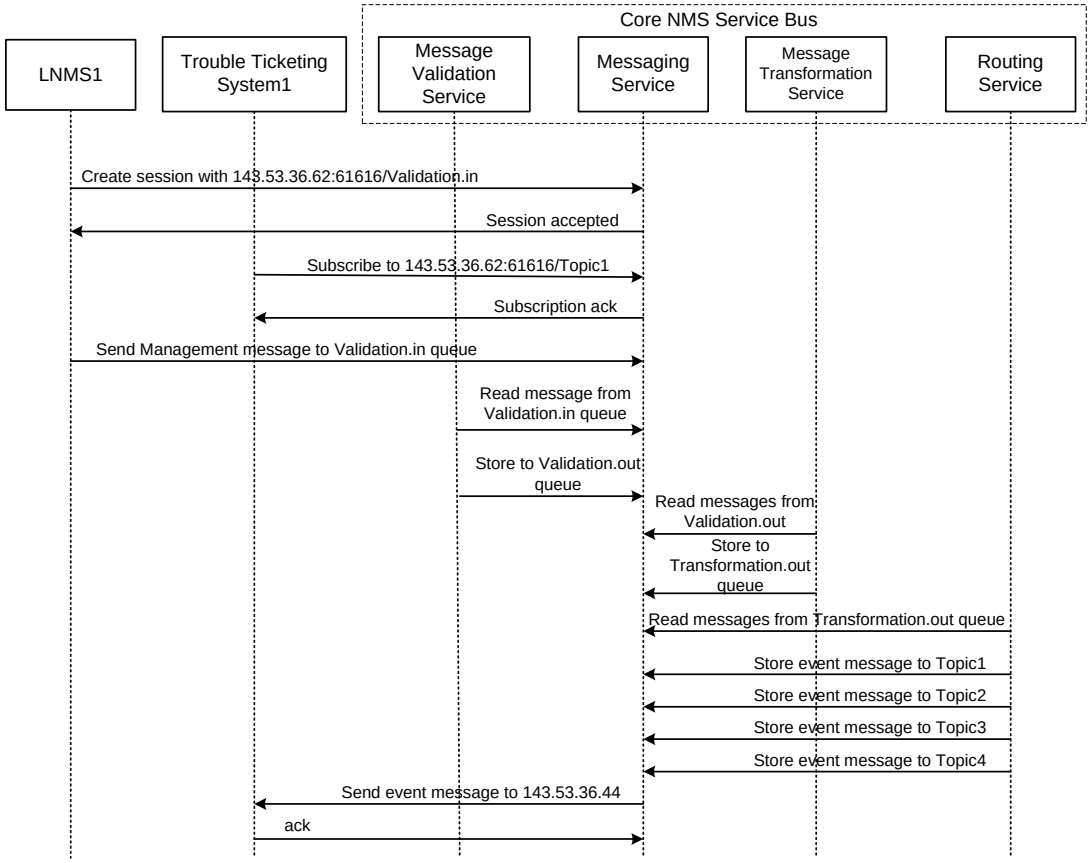


Figure 6.34: Interactions between remote services and the Core NMS Service Bus

In figure 6.34, the LNMS creates a session with the Core NMS Service Bus as described in chapter 5. The Core NMS Service Bus sends an acknowledgement back to the LNMS. The Messaging Service contains all the queues and topics as explained in the previous chapter. The LNMS sends its management information in the form of management messages to the validation.in queue. As messages are stored into the queue, the Message Validation Service is initialized and reads the messages from the queue. It processes each management message and sends the validated management message back to the Messaging Service to store it in the validation.out queue, if it complies with the XSD schema, or else it sends an error message and stores it in the validation.error queue.

The Message Transformation Service reads the messages from the validation.out queue and transforms them. After the transformation process the messages are sent back to the Messaging Service and they are stored into the transformation.out queue. Next, the Routing Service reads the messages from the transformation.out queue and processes them. Each management message completes a series of processes as explained in the previous section by the Routing Service and the output of each management message is stored to the different topics. The final step is the transmission of the event messages to the subscribed TTS. Each message that is sent to the subscriber is acknowledged.

The message that is being sent to the Core NMS Service Bus is depicted in figure 6.35. As can be seen this message consists of three events that have been originated from the LNMS1 (http://esb.nms1). The message is consumed by the Messaging Service and is being validated. The validation output is successful since the management message complies with the validation.xsd schema. This stage is not depicted in a figure since the Transformation Service is being activated. In case of a message error in the validation process, the message would not be transformed since it would have been placed to the validation.error queue. The message, after validation is sent to the validation.outqueue.

```
<Results xmlns=" http://esb.nms1">
<ROW>
        <eventid>1</eventid>
        <eventuei> uei.opennms.org/reporting </eventuei>
        <eventtime>2010-03-16 12:10:50.0</eventtime>
        <eventhost>143.53.36.72</eventhost>
        <eventsource>ProCurve J8697A Switch </eventsource>
        <eventdpname>undefined</eventdpname>
        <eventcreatetime>2010-03-16 12:10:51.421</eventcreatetime>
        <eventdescr> High alert for interface 143.53.36.72 ;</eventdescr>
        <eventlogmsg>The Accuracy level is : 99%</eventlogmsg>
        <eventseverity>1</eventseverity>
        <eventlog>Y</eventlog>
        <eventdisplay>N</eventdisplay>
    </ROW>
<ROW>
        <eventid>2</eventid>
        <eventuei> uei.opennms.org/reporting </eventuei>
        <eventtime>2010-03-16 12:10:51.0</eventtime>
        <eventhost>127.0.0.1</eventhost>
        <eventsource>Host </eventsource>
        <eventdpname>Lab-PC</eventdpname>
        <eventcreatetime>2010-03-16 12:10:51.463</eventcreatetime>
        <eventdescr> IP packet loss 0</eventdescr>
        <eventlogmsg>The Total IP packet loss is 0</eventlogmsg>
        <eventseverity>2</eventseverity>
        <eventlog>Y</eventlog>
        <eventdisplay>N</eventdisplay>
    </ROW>
<ROW>
        <eventid>7</eventid>
        <eventuei> uei.opennms.org/reporting </eventuei>
        <eventtime>2010-03-16 15:09:23.53</eventtime>
        <eventhost>127.0.0.1</eventhost>
        <eventsource>Host </eventsource>
        <eventdpname>Lab-PC</eventdpname>
        <eventcreatetime>2010-03-16 15:09:23.124</eventcreatetime>
        <eventdescr> hrSWRStatus.200 = INTEGER: running(1)</eventdescr>
        <eventlogmsg>Application 200 is running</eventlogmsg>
        <eventseverity>3</eventseverity>
        <eventlog>Y</eventlog>
        <eventdisplay>N</eventdisplay>
    </ROW>
</Results>
```

Figure 6.35: Input management message consist of 3 events

The Transformation Service reads the message from the validation.out queue and the output of the process is shown in figure 6.36.

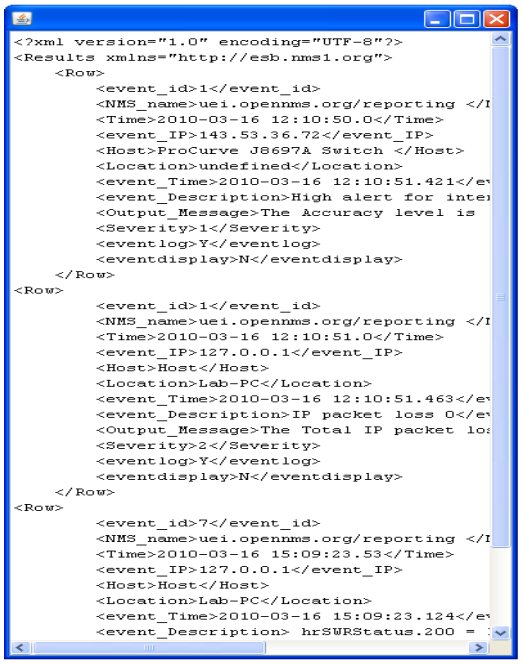


Figure 6.36: output of the Message Validation Service

From the figure above, it can be seen that the representation of the elements have been transformed according to transformation.xslt file. The message is now stored in the transformation.out queue. The Routing Service retrieves the message from the queue and processes it. The output of the process can be seen in the figure 6.37.

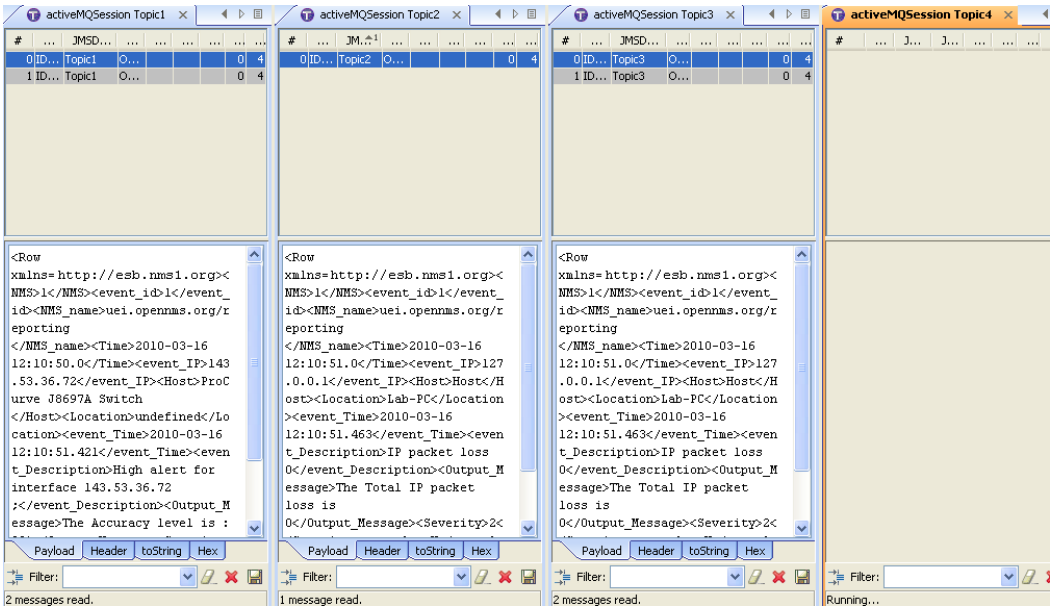


Figure 6.37: Event messages in the 4 Topics

Hermes software has been used in order to capture the event messages. The message is split into thee individual event messages that each had been enriched with an element <NMS>1</NMS>. As can be seen in figure 6.37, there are five event messages in total. This is because two of the tree events in the management message are being duplicated. The results show the correct functioning of the Core NMS Service Bus as an integral part of the network management platform.

6.4.3.3 Performance Behaviour of the Core NMS Service Bus

This section presents the tests being carried out in order to evaluate the performance behaviour of the Core NMS Service Bus subject to the following performance parameters:

- Message throughput
- Total events per message

6.4.3.3.1 Message Throughput

The test involved two LNMSs sending high volumes of management information related to faults to the Core NMS Service Bus during the event reporting period. These faults are collected and processed by both the LNMS itself and the GNMA in order that they can be shared by other LNMSs. For this experiment, each management message has been predefined to contain 120 events and its message size to be approximately 120Kbytes. Four TTSs were connected to the Core NMS Service Bus and act as consumers. Each TTS subscribed to one of the topics that the Core NMS Service Bus provided in order to consume specific management information. Thus, one consumer per topic was used for this experiment. The test ran for 100 seconds in order to evaluate the throughput of the Core NMS Service Bus in order to examine the efficiency of the Core NMS Service Bus and to test whether it is capable of handling large amounts of information within a specific time period. Figure 6.38 illustrates the performance of the Core NMS Service Bus.

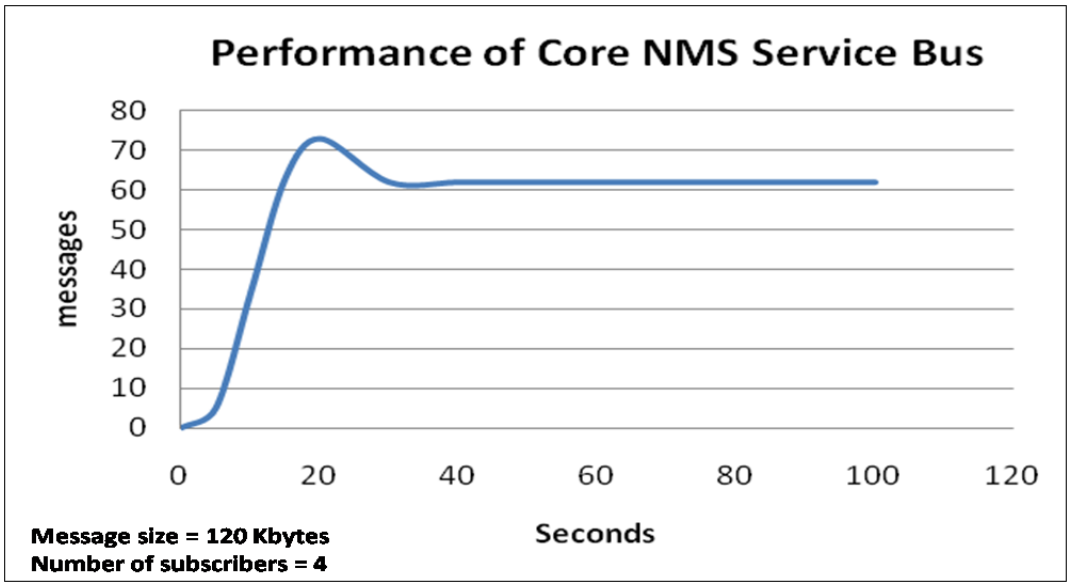


Figure 6.38: Throughput of the Core NMS Service Bus

Results and Analysis

Figure 6.38 illustrates the message throughput at the Core NMS Service Bus. The throughput measurements have been conducted by measuring the time needed for each management message to be validated, transformed and routed to the appropriate topic and dispatched to the TTSs. The experiments do not focus on the actual message size of each management message but on the number of events that are contained in it. This has been decided due to the fact that the size of each message could vary according to the information that they carry. For instance, an event could have a long descriptive text explaining the particular event. As can be seen in figure 6.38, there is a warm-up period that the Core NMS Service Bus needs in order to reach the maximum rate. This is due to hardware and software requirements since there is a high volume of

information that has to be processed and the appropriate resources have to be allocated to the Java Virtual Machine. The recourses are given to the software gradually and not immediately. The overall message throughput reaches its average rate at 62 msgs/sec for four subscribers and stay at this rate almost constantly. The throughput indicates the number of event messages sent to the TSSs. Since the average message size of each event is approximately 1 Kbyte, the amount of information that is processed per second is approximately 62 Kbytes. The events are shown in the each TTS's web page as illustrated in figure 6.22.

6.4.3.3.2 Event Processing Capability

The previous scenario presented the performance of the Core NMS Service Bus when the number of events is fixed to 120, each event requires a message size of 1 Kbytes to carry. Due to the event-driven nature of the Core NMS service Bus, it will be of interests to examine the number of events that the Core NMS Service Bus can handle. A change in the state of the network will trigger an event being captured by the LNMSs and being forwarded to the Core NMS Service Bus. The test that follows, examines the impact on the performance of the Core NMS Service Bus when the LNMSs send management messages that contain a variable number of events per management message.

The tests were performed as follows: management messages were sent constantly to the validation.in queue. From the validation.in queue, each message is validated, then transformed and split into several one-event messages. The throughput of the Core NMS Service Bus was measured

by increasing the number of events per management message each time. The results are depicted in figure 6.39.

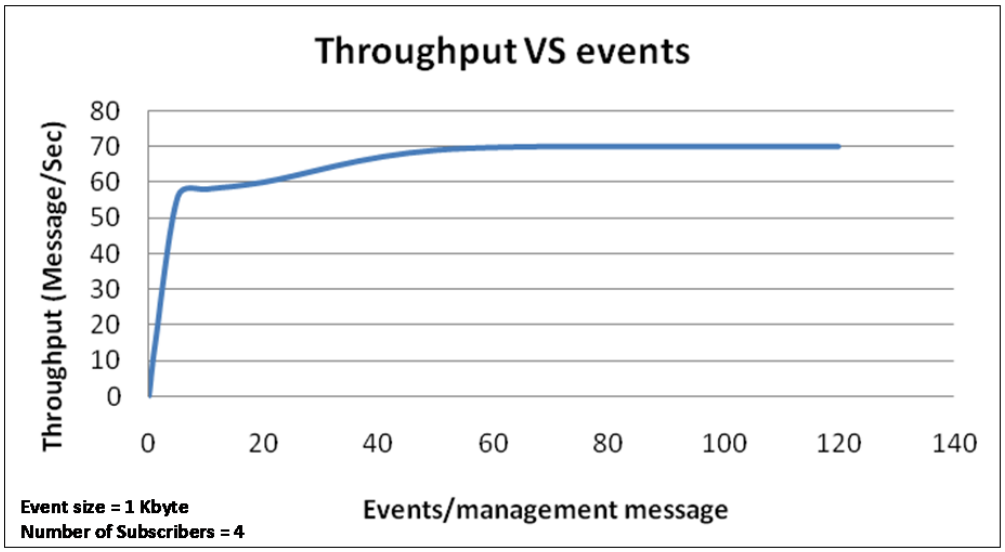


Figure 6.39: Throughput of the Core NMS Service Bus in relation to events per message

Results and Analysis

As seen in figure 6.39, the Core NMS Service Bus performs better when there are many events in each management message. The reason is that each management message is considered as a packet that has to be processed in several steps. First, the message is stored in the incoming queue (validation.in) and processed by the Validation Service. In this step, the body of the message is processed by the validation component where it is compared against a predefined XML. As explained in Chapter 5, JAXP validation API has been used in order to process the XML-based body of the message. This means that the body is parsed and compared against the XSD schema. After the validation process the message is stored in the validation.out queue. Next, the Message Transformation Service reads the message from the queue and processes it again. This time, the body of the message is parsed by the XSLT transformer and transformed into a

format that has been defined in the XSLT stylesheet file. The third step is the routing of the message where, as explained in the previous chapter, the body of the message is enriched, split and routed to different queues and topics according to the content of the message. Messages originated from the LNMS are routed to the topics and messages originated from the trouble ticketing systems are routed to queues.

Thus, in the Core NMS Service Bus each management message is parsed and processed in three separate instances. In the case of small messages, such as management messages with one event per message, the parsing and processing of the XML-based body takes less time when compared to larger management messages, because the content that has to be processed is less. However, the process of reading and storing each message from/to the queue is more frequent with messages being passed from one pluggable component to another (i.e. from Message Validation Service to Message Transformation Service etc.). This results in many interactions (message exchange) taking place in the service bus and consequently leads to a reduced performance. As the number of events increase in every message exchange, there is a trade-off in performance. As depicted in figure 6.39, with over 50 events per management message the throughput rate reaches its maximum.

6.4.3.4 Number of Subscribers

In this test, the impact of the number of subscribers on the message throughput is examined. For this experiment Hermes software [HERMES]

has been used in order to act as message subscriber. Hermes provides access to JMS queues and topics. Multiple instances of this software have been used in four different hosts in order to increase the number of subscribers. Each of these instances is listening on every topic of the Core NMS Service Bus. In total, the number of subscribers used for this experiment is 160. The following figure illustrates the results of this experiment.

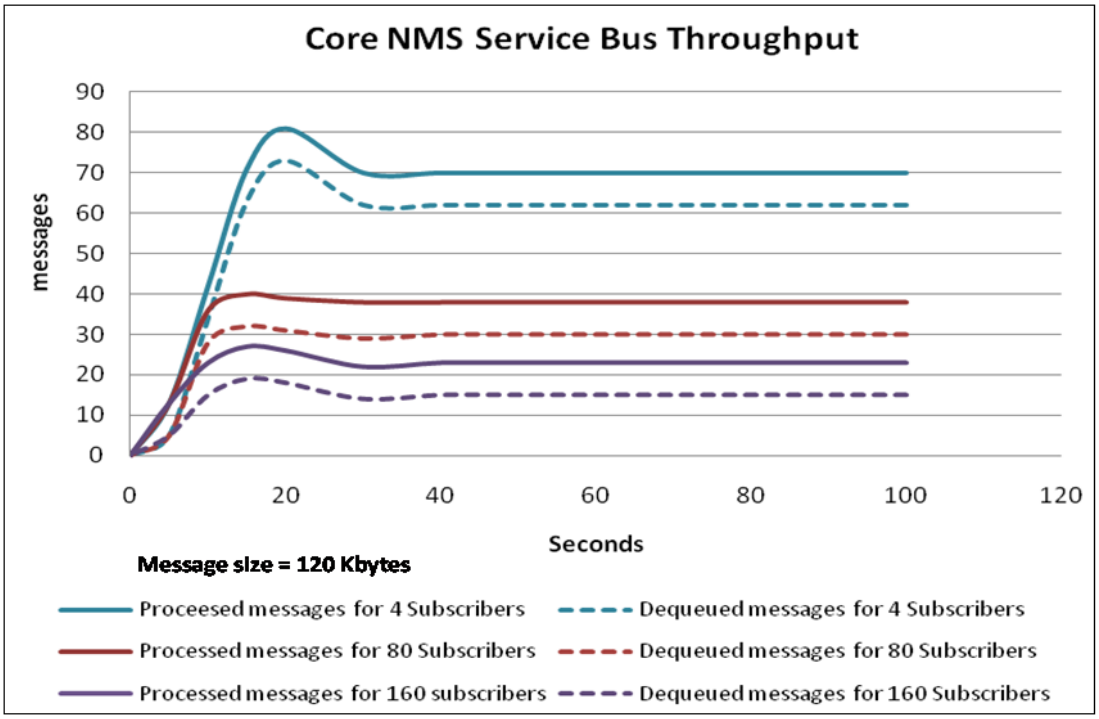


Figure 6.40: Throughput of the processed and dispatched messages

Result and Analysis

Figure 6.40 shows the throughput of the processed and dispatched (dequeued) messages. Dequeued are the messages that have been successfully read off the queue (i.e., they have been acknowledged by the consumer) [SUNJMS]. The tests have been performed for a wide range of subscribers but this figure illustrates three cases that show the major

performance degradation due to the increasing numbers of subscribers. First observation from the figure 6.40 is the difference between the processed messages per second by the Core NMS Service Bus and the dispatched messages per second. This difference is due to the extra time that each event message needs in order to be sent to the destination and to be processed by the subscriber. In case of a slow connection or an application that processes slowly each message, the actual throughput could be lower. On the other hand, if the subscriber resides in a system that is powerful and the connection is fast (i.e. intranets) then the throughput could be closer to the processed throughput.

The first test has been performed by using four subscribers. For this test the implemented TTS has been used, as explained in the previous section. Running multiple instances of the trouble ticketing application on each PC can result in high memory consumption. An alternative software (Hermes) has been used in order to consume event messages from the topics. It is lightweight and it can be subscribed more than once to each topic; as a result, it is possible to increase the number of subscribers by using this software. As seen in figure 6.40, as the number of subscribers increases the throughput drops. The received message rate decreases significantly with an increasing number of subscribers. This can be explained as follows: all event messages are delivered to many subscribers, therefore each message is replicated according to the number of subscribers. This requires more CPU processing power for dispatching messages and increases the overall processing time of a single message in the Core NMS Service Bus.

6.5 Conclusion

This chapter first presented the development of the Core NMS Service Bus according to the theoretical design of the previous chapter. Furthermore, the development and testing of Message Validation Service, Message Transformation Service and Message Routing Service have been presented. Moreover, a TTS that has been developed as a part of the overall proposed architecture has been presented. TTS has been used as a global management service in order to consume management information provided by LNMSs through the Core NMS Service Bus. This chapter also included tests that have been performed in order to test the performance of the Core NMS Service Bus. Several experiments in the form of scenarios have been conducted in order to evaluate the behaviour of the proposed Network Management Platform. Furthermore, an analysis of the results is included in each test case. It is shown how the Core NMS Service Bus behaves under different conditions.

Chapter 7 : CONCLUSIONS AND FUTURE DEVELOPMENTS

This final chapter summarises the research work in this thesis. Section 7.1 summarises work being carried out in this thesis and draws the conclusions that the proposed network management framework can fulfil the SOA design principle. Section 7.2 states the significance of this thesis in terms of contributions and achievements and section 7.3 indentifies areas in which this work can be developed further.

7.1 Summary

The key contribution of this thesis is the design of a Next Generation Network Management framework based on the SOA concept. International Telecommunication Union (ITU) has foreseen the benefits of loosely-coupled architectures and has adopted the SOA concept in many areas in the NGN architecture such as IMS (IP Multimedia Subsystem) and SDP **(**Service Delivery Platform) for delivering Internet based services such as VoIP, email and social network to mobile telecommunication network users [OHNI07]. Furthermore, ITU proposes the use of SOA for designing the management plane of the NGN architecture; however a complete SOA-based model has not yet been proposed [M.3060].

The focal attention regarding the use of SOA principles and methodologies in management frameworks has been made in the Business Management Layer due to the fact that TM Forum follows a top-down approach [TMF053]. eTOM and SID frameworks provided by the TM Forum are the

leading industry models for developing business process functions aimed at simplifying interoperability and promoting process and service reuse between IT systems and business partners [TMF]. Moving towards the lower layers of the management framework (from Service Management Layer to Element Management Layer), there is a mixture of legacy management applications, monolithic OSS systems and software systems that are usually operate in isolation. The emergence of the NGN will require the collaboration and the convergence of those monolithic management systems running in distributed and heterogeneous environments to be operated as one agile Next Generation Network Management framework supporting the business needs of the service and network providers. Although solutions have been proposed over the years [PAV00], [HASS09], [LI05], the outcome (of these solutions) is a management infrastructure that is still tightly coupled, not flexible and not scalable enough to support the NGN as having discussed in chapter 2.

Given the fact that there is not yet a complete solution for an open standard management framework for integrating heterogeneous management systems into a loose coupling way, this research is focused on designing and developing an SOA-based management framework, which could be used as a backbone infrastructure for managing NGNs.

For achieving this goal, a comprehensive understanding of the evolution of telecommunication management frameworks has been studied at Chapter 2. Additionally, a thorough examination of software architecture and the integration technologies has been conducted and the concept of the SOA philosophy has been explored at Chapter 3.

This research spans across all the layers of the TMN model starting from the lowest layers to the highest layers. More specifically, the author first presented how management information is being collected from different network elements through the use of agents and then how this information is being processed by the NMSs. Next, a Network Management Middleware Layer that allows heterogeneous management systems to communicate with each other, regardless of the implementation technology has been proposed. Experiments were carried out throughout the development phase and verified that the components comprising the Network Management Middleware Layer were functioning as it was anticipated. Finally, a testbed consisting of NMSs, the Core NMS Service Bus and a developed trouble ticketing system has been developed in order to conduct experiments and test the behaviour of the proposed Network Management Platform.

7.2 Fulfilling SOA Design Principles

In relation to fulfilling the SOA design objectives, the agile and scalable Network Management Framework developed in this thesis provides a dynamic integration of heterogeneous network systems that support a wide range of management protocols with different information models. The integration adopts a loosely coupled approach that allows the critical network management information, such as fault data, to be exposed as a service that can be subscribed by customers or network operators. In this case, a TTS is designed for notifying network operators about the health of

the network infrastructure. The framework can also be linked up with a SDP in order to manage the service delivery. Therefore, the management of the NGN based on the SOA philosophy is achieved (figure 7.1).

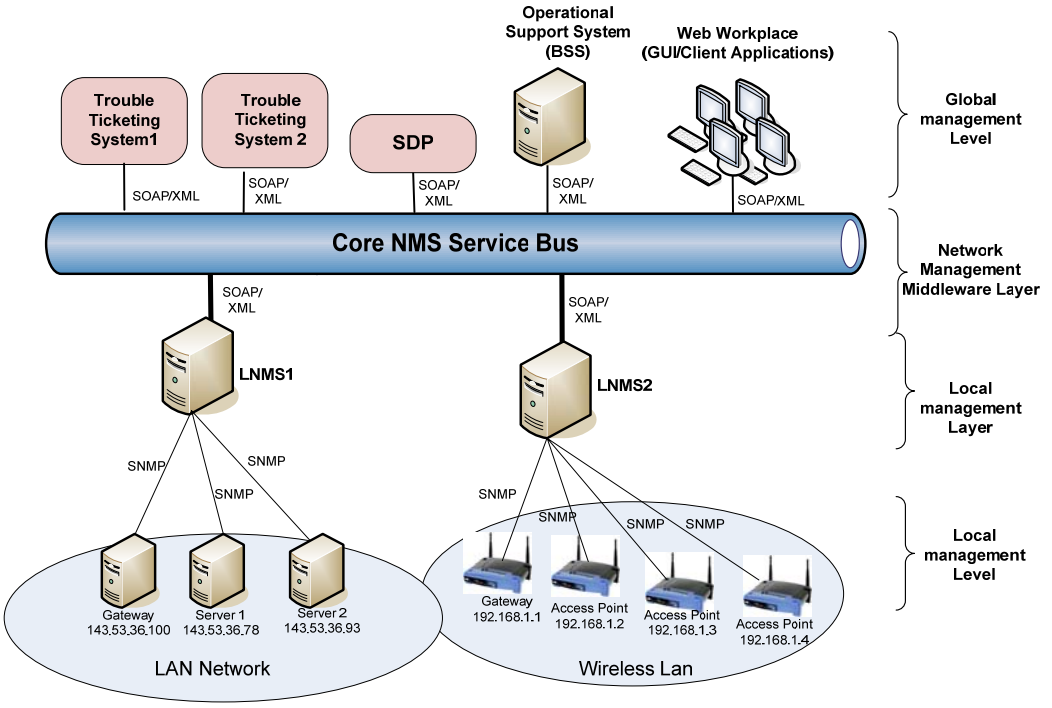


Figure 7.1: SOA-based Network Management Platform

The following summarises how the proposed network management framework fulfils the SOA design principle.

7.2.1 Service Reusability

As stated in chapter 3, service-orientation encourages reuse in all services, even if there is no immediate requirement for reuse [ERL05]. The communication between management applications in the local and global level in the proposed Network Management Platform is based on messaging and is achieved using queues and topics residing on the Network Management Middleware Layer, providing asynchronous

communications. Furthermore, the management information is not bound to particular management services and applications; however, it is categorized and made available for consumption by any remote service interested in it.

In addition, both local and global management services can be easily repurposed and reused in other parts of the architecture. For example, TTSs can subscribe to other queues and topics as defined in Chapter 6 in order to consume management information that could be provided by external providers or it can consume other types of management information in the same infrastructure. For instance, a TTS can register to topics and consume management information related to faults, performance, configuration, accounting, and security. Furthermore, the Routing Service implemented in the Core NMS Service Bus is capable of accommodating other rules related to other types of information. For instance, new rules can be included in order to relate the network usage of a particular service to the subscribed customers. A Customer Relationship Management (CRM) system can acquire this type of information from the Core NMS Service Bus in order to charge those customers.

7.2.2 Services Discoverability

Services need to express their functionality via service contracts. This means that a service contract should express the functionality, the data types and data models. Each remote service has its own service contract. The service contracts are WSDL [W3C01] files that expose the available operations, the structure of the request parameters, and the response

generated by the service. The types of binding supported by the service are also included in this file as well as the location of the service known as endpoint. The content of the WSDL file is expressed in XML-based format. Appendix E contains the WSDL files of the local management services. The remote service's WSDL files can be uploaded in a public UDDI service repository such as IBM UDDI [IBMU] for public exposure and consumption or can be stored internally in an application server for local exposure and consumption by other services. These services allow their underlying logics to be discovered, accessed and understood by new potential service requestors. Thus, services are naturally discoverable.

7.2.3 Service Loosely Coupling

Loose coupling is a fundamental concept of SOA aimed at reducing dependencies between different systems [ERL05]. Asynchronous communication reduces the dependencies among systems and as a result it promotes loose coupling. The LNMAs in the Network Management Platform are not connected directly with each other; however; they are connected to the Core NMS Service Bus. They can be added or removed to/from the Network Management Platform without affecting the overall function of the architecture. This forms a high degree of loose coupling and new applications can be added or removed from the architecture, can listen to more than one type of information without affecting any other system.

7.2.4 Service Composability

On a fundamental level, this principle emphasizes the need to hide the underlying details of a service. This directly enables and preserves the loosely coupled principle and allows services to act as black boxes, hiding the underlying logic. Data abstraction is the approach followed in this thesis in order to meet this principle. This approach hides the complexity of the data shared among services by defining a new, better organized structure, which is handled by the Core NMS Service Bus. The result is that a remote service can access the data in a well-organized, logical format, without knowing the actual physical layout of the data that it receives. For data abstraction to be provided for LNMAs, the data model used for exchanging information among them is based on the XML. XML provides mechanisms that define and describe the structure, content and schematics of the data and can support the creation of coarse-grained services. Through XML, new coarse-grained functionality, which is derived from other remote services, has been created in the Core NMS Service Bus.

In the proposed Network Management Platform the new coarse-grained functionality is exposed in the form of topics. This means that each topic exposes a particular type of information, for instance, performance management information, fault management information, etc. LNMAs offer basic services that provide the information in order to create this new coarse-grained functionality, which abstracts the underlying logic, data structures and data format of the LNMSs. Furthermore, the new coarse-grained functionality can be viewed as a basis of creating new

composable services because it exposes information that cannot be provided by other individual LNMAs. This composable service can be regarded as a new service that exposes its own types of management information by using its own communication patterns (Publish/Subscribe), having its own data representation and schematics.

7.2.5 Service Autonomy

Autonomy gives the service control over the logic that they encapsulate [ERL05]. In the proposed Network Management Platform, each NMS encapsulates its own logic (i.e. agents, control unit, polling unit etc.) and as a result it can operate independently as a standalone application. This means that they can collaborate to provide shared functionality but at the same time they remain independent and no other system can affect their internal operations (i.e. data polling intervals, agent configuration, etc.). In the same way, the GNMAs (i.e. TTSs) control their own logic.

7.2.6 Service Statefulness

Depending on the scale of a service landscape, state management can become one of the central problems in the efficient service design [ERL05]. Stateful services are based on the assumption that a service keeps the state of an ongoing interaction, leading to problems for services with many clients, high throughput and long-running transactions. The alternative is to design a stateless architecture where each service does not keep a state with other cooperating services.

In the proposed Network management Platform the remote services are stateless because they are based on asynchronous (fire and forget) communication. For instance, the Core NMS Service Bus does not reply to requests originated from the TTSs in the testbed. This means that it does requests (i.e. keep a state), but stores the management information to topics and the TTSs that are interested in a particular type of information (i.e. faults, performance, configuration etc.) can perform a subscription request to the interested topics and then the information is forwarded to them. In the same way, LNMSs are not processing requests from the Core NMS Service Bus to acquire management information; it rather sends (push) the collected information to the Core NMS Service Bus. As a result, the components comprising the Network Management Platform are stateless. Stateless services can achieve low bandwidth consumption compared to stateful services due to the fact that interactions between stateless services are fewer compared to stateful services.

7.3 Achievements Derived from the Thesis

The following subsections elaborate the major contributions and achievements of this thesis.

7.3.1 Design and Development of an Agent

An agent has been designed based on the SNMP framework. This agent resides in a network element and collect performance, faults and configuration management information from network elements. This agent

was then to pass the collected management information to a proposed network management system when requested.

7.3.2 Design and Development of an Event-driven Network Management System

Following the principles of SOA, a network management system has been developed. The author designs FCAPS functionalities to be exposed as Web service. NMS can share its functionality with other systems such as customer relationship management system and consequently network management information can be accessed by network operators as well as users at the NGN.

7.3.3 Design and Development of an XML-based Gateway Component

An XML gateway component has been developed in order to expose the management information in a common message format, XML-based.

Other approaches have been proposed to express SNMP-based management information into XML [YOON06], [MART02], [STRA99]; however, these approaches cannot be used for managing large, heterogeneous networks as discussed in Chapter 4. The proposed XML gateway converts management information residing in the NMS's database, into XML-based messages. This allows the NMS to use its own information model to retrieve management information from the agents and store it to a database without the XML gateway interfering with this

process. The benefit is that the XML gateway can be used to retrieve management information from any NMS system.

7.3.4 Design and Development of a Network Management Middleware Layer

The main contribution of this research is the design and development of a Network Management Middleware Layer. The proposed layer is based on messaging and asynchronous communication that removes the integration complexity from the management systems. Moreover, it handles the heterogeneity on the information expressed by different systems.

The contribution includes the design and development of a messaging service that allows communication and data transfer among management systems. A persistent store has been proposed in order to recover management data in case of a middleware failure. Moreover, a Validation Service that has been created for the purpose of validating management messages received from heterogeneous NMSs has been developed. The Validation Service eliminates the creation of unnecessary faults and errors by invalid messages. Furthermore, a transformation mechanism that is responsible for dealing with different data formats has been developed. Taking into account the problem that arose from legacy systems, the SOA-based management platform uses this transformation mechanism in order to accommodate heterogeneous systems. In addition, a Routing Service has been designed and developed in order to minimize the interfaces and the dependencies among remote services as well as to provide routing rules for delivering management messages to other management

systems. Finally, a message archive service has been developed in order the management messages that are passed through the Middleware Layer to be stored into folders for inventory purposes.

7.3.5 Testbed Development – Applications and Evaluation

A trouble ticketing system has been developed as a part of the overall proposed architecture. It has been used as a Management Service in order to consume management information provided by the Network Management Middleware Layer. A testbed has been developed in order to test the performance and behavior of the Network Management Middleware Layer. Several experiments in the form of scenarios have been conducted in order to evaluate the behavior of the proposed SOA-based management platform.

7.4 Future Work

The research in this thesis can be progressed further in several areas. The following subsections illustrate some of the areas that this research could be extended into.

7.4.1 Alternative Mechanisms for Message Routing

Management information is routed to the appropriate destination via rules. These rules are based on XPath expressions. Other techniques can be incorporated in order to provide more advanced rules for making decisions according to the content of the message. For example, content-based routing algorithms based on the Ant Colony Optimization could be applied

in the Core NMS Service Bus [ABBA02]. Furthermore, advertisement-based routing techniques could be used as filters by the Core NMS Service Bus in order to indicate its intention to publish notifications to the subscribers. Advertisements can be used as an additional mechanism to further optimize content-based routing [BALD05].

Additionally, rule engines such as Drools [DROOL] could be used in order to dynamically reconfigure the routing rules when new management systems are connected to the Core NMS Service Bus.

7.4.2 Scheduling of Message Queues

In the proposed Core NMS Service Bus, four different queues (MS1, MS2, MS3, and MS4) have been developed. As explained in Chapter 5, queues have been used in order to make possible the communication among Management Services connected to the Core NMS Service Bus. A major drawback of dedicating one queue for each Management Service is that if an additional Management Service would like to connect and communicate with an existing Management Service via the Core NMS Service Bus, a new queue has to be created manually. A proposed solution is to use one queue for all Management Services. A scheduling algorithm can be applied in the Core NMS Service Bus in order to distribute the messages to the connected Management Services. Hence, new Management Services can be connected to the Core NMS Service Bus without the need for manually creating new queues.

7.4.3 Security, Policy and Co-ordination

In this research, the proposed Network Management Middleware Layer is being built on a single Enterprise Service Bus. However, from a business perspective, some questions need to be answered: If there are competing service and network providers that would like to share management information among them via the proposed middleware layer then who will have the ownership of the middleware infrastructure? How secure is the management information exchange?

In an actual business environment, enterprise organizations need to consider ownership as well as intellectual property rights, which need to be protected from other competitors. This could require preservation of their technological innovations and trade secrets. As a result, one common Core NMS Service Bus may not be an attractive solution for the enterprises. However, a solution would be for each enterprise to develop its own Core NMS Service Bus and expose the information that they would like to share with their partner's Core NMS Service Bus. This will allow each of them to control, categorize and prioritize their own management information that they will share internally and with other partners. In this case, multiple Core NMS Service Buses will be required to communicate with each other and be able to exchange management information among them. This means that they need to regulate and secure their management information. To regulate management information, WS-Coordination, WS-policy and WS-security [W3C07d], [W3C06c] [OASIS07a], could be used in the Core NMS Service Bus. WS-Coordination is a Web Service specification that enables an application to

create context needed to propagate an activity to other services and to register for coordination. It could be used in order to define the structure of context and requirements for propagating context between cooperating remote services. In other words, it can specify which applications are allowed to exchange information under certain circumstances. The WS-Policy specification can also be used in order to advertise the policies (i.e. certain circumstances) to other Core NMS Service Buses. The WS-security could also be used in order to add security features to messages that will be exchanged among Core NMS Service Buses.

7.4.4 SID Information Model

The information framework, also known as the SID, is one of the TM Forum's foundational frameworks [TMF]. As it is described in Chapter 2, SID addresses the needs of the industry where shared information and data model is required. SID can be used as a way to map application programme interfaces that are exposed by different applications and to express the information in the standard structure and terminology. In the proposed Core NMS Service Bus, the Message Transformation Service transforms the management information into a common message format based on an information model that does not follow a standardized specification. SID model can be applied in the Core NMS Service Bus in order to be able to standardize the management information that is transformed by the Message Transformation Service. As a result, it will allow the Core NMS Service Bus to operate as a backbone messaging infrastructure for NGOSS enabled applications standardized by TM Forum.

NGOSS applications can be integrated into the Core NMS Service Bus in order to expand the proposed SOA-based Network Management Platform to become a complete OSS/BSS solution. NGOSS applications are developed by companies such as Ericsson and IBM by being exposed as Web Services and by following the SID information model [TMF053]. These NGOSS solutions are envisioned to be a part of the eTOM framework.

There are many emerging research topics that can be further studied in relation to SOA, Next Generation Network Management, which are not limited to the above mentioned directions.

REFERENCES

[ABBA02] H. A. Abbass, R. A. Saeker and C. S. Newton, Data mining: A Heuristic Approach

[ABER06] Aberdeen Group, “Enterprise Service Bus and SOA Middleware,” June 2006. [Online] Available at http://www.fiorano.com/docs/aberdeen.pdf

[ADAM98] D. X. Adamopoulos and C. A. Papandreou, “Distributed Processing Support for New Telecommunications Services,” in proceedings of IEEE/IEE ICT ’98, Greece, Vol. III, pp. 306-310, 1998.

[AMIR95] K. Amirthalingam and R. J. Moorhead, “SNMP-an overview of its merits and demerits,”in proceedings of the 27th Southeastern Symposium on System Theory (SSST'95), 1995.

[AXIS] Apache Axis, http://ws.apache.org/axis, March 2008.

[BALD05] R. Baldoni, L. Querzoni and A. Virgillito, “Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey,” 2005. [Online]. Available at citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.1108

[BEN90] A. Ben-Artzi, A. Chandna and U. Warrier, “Network management of TCP/IP networks: present and future”, IEEE Network Magazine, Vol. 4, pp. 35-43, 1990.

[BEHA10] G. K. Behara, P. Mahajani and P. Palli, “Telecom Reference Architecture, Part 1.” 2010. [Online]. Available at http://www.bptrends.com/publicationfiles/FOUR%2007-10-ART-Telecom-Reference%20Arch-Gopala%20et%20al.pdf

[BERNE05] T. Berners-Lee, R. T. Fielding and L. Masinter, “Uniform Resource Identifier (URI): generic syntax,” IETF RFC 3986, January 2005.

[BIH05] J. Bih, “Deploy XML-Based Network Management Approach,” IEEE Potentials, Vol. 24, Iss. 4, pp. 26-31, 2005.

[BHOL02] S. Bohla, R. Strom, R. Bagchi, S. Zhao and Y. Auerbach, “Exactly-once Delivery in a Content-based Publish-Subscribe System, in proceedings of the International Conference on Dependable Systems and Networks, 2002.

[BLUM99] U. Blumenthal and B. Wijnen, “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3),” IETF RFC 2574, April 1999.

[BOHO02] C. Bohoris, G. Pavlou, A. Liotta, “A Hybrid Approach to Network Performance Monitoring Based on Mobile Agents and CORBA,” in proceedings of the IEEE/ACM International Workshop on Mobile Agents for Telecommunication Applications (MATA'02), Barcelona, Spain, pp. 151-162, Springer, October 2002.

[BORL] Borland, A Micro focus Company, http://www.borland.com/us/products/visibroker/index.html.

[CAET07] J. Caetano, et al, “Introducing the user to the service creation world: concepts for user centric creation, personalization and notification,” in proceedings of the International Workshop on User centricity – state of the art, Budapest (Hungary), 2007.

[CAPO02] M. Caporuscio, A. Carzaniga and A. Wolf, “An Experience in Evaluating Publish/Subscribe Services in a Wireless Network,” in proceedings of the 3rd International Workshop on Software and Performance, Rome, Italy, pp. 128-133, 2002

[CAST02] M. Castro, P. Druschel, A. Kermarrec and A, Rowston, “Scribe: A large-scale and decentralized application-level multicast infrastructure,” IEEE Journal on Selected Areas in Communications, Vol. 20, 2002.

[CARE02a] P. Carey, *New Perspectives on XML*, Course Technology, 2nd edition, 2002.

[CARE02b] K. Carey and F. O’Reilly, “Heterogeneous Network Management using WBEM,” Distributed Management Task Force (DMTF) Conference, San Jose USA, 2002.

[CARL08] M. L. Carlander and J. Olander, “Service delivery platforms for the multimedia marketplace,” Erickson Review, No. 2, 2008.

[CASE96] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2),” IETF RFC 1905, January 1996.

[CASE99] J. Case, R. Mundy, D. Partain and B. Stewart, “Introduction to Version 3 of the Internet-standard Network Management Framework,” IETF RFC 2570, April 1999.

[CASE02] J. Case, R. Mundy, D. Partain and B. Stewart, “Introduction and Applicability Statements for Internet Standard Management Framework,” IETF RFC 3410, December 2002.

[CHAE05] L. Chae-Sub and D. Knight, “Realization of The Next-Generation Network,” IEEE Communications Magazine, Vol. 43, Iss. 10, pp. 34-41, 2005.

[CHAP04] D. A. Chappell, Enterprise Service Bus, O’Reilly Media, 2004.

[CHUN98] P. E. Chuang, Y. Huang, S. Yajnik, D. Liang, J. C. Smith and C. Y.Wang, “DCOM and CORBA Side by Side, Step by Step, and Layer by Layer,” C++ Report, January 1998

[CISC07] CISCO, “Performance Management: Best Practices White Paper,” 2007.

[CLEM07] A. Clemm, Network Management Fundamentals, Cisco press, 2007.

[COM] Microsoft, Distributed Component Object Model technologies, http://www.microsoft.com/com/default.mspx

[CORBA] CORBA, http://www.corba.org/

[DATE06] C. J. Date, The relational database dictionary, O’Reilly Media, 2006.

[DDWRT] DD-WRT, http://www.dd-wrt.com/site/index

[DAVI99] R. Davison and T. Turner, “A Practical Perspective on TMN Evolution,” in proceedings of the 6th International Conference on Intelligence and Services in Networks: Paving the Way for an Open Service Market, Barcelona, Spain, April 1999, pp. 3-12, 1999.

[DROOL] Drools, http://www.jboss.org/drools

[ECLIPSE] Eclipse IDE http://www.eclipse.org/

[EGGE03] R. Egger and S. Sunku, “Efficiency of SOAP Versus JMS,” in proceedings of the International Conference on Internet Computing (IC’2003), June 2003.

[EJB] Enterprise JavaBeans Technology, http://www.oracle.com/technetwork/java/index-jsp-140203.html

[EMME00] W. Emmerich, “Software Engineering and Middleware: ARoadmap,” in proceedings of the International Conference on the Future of Software Engineering, Limerick, Ireland, pp. 117-129, 2000.

[ERL04] T. Erl, Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, New York: Prentice Hall, 2004.

[ERL05] T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design, New York: Prentice Hall, 2005.

[ERL09] T. Erl, SOA Design Patterns, New York: Prentice-Hall, 2009.

[ERL10] T. Erl, C. Utschig, B. Maier, H. Normann, B. Trops, T. Winterberg and P. Cheliah, Next Generation SOA: A Real-World Guide to Modern Service-Oriented Computing, New York: Prentice-Hall, 2010.

[ETSI] European Telecommunications Standards Institute (ETSI), http://www.etsi.org/deliver/etsi_ts/101400_101499/101441/06.11.00_60/ts_101441v061100p.pdf

[EURE99] Eurecom, “Project P812-GI: TMN Evolution – Service Providers’ Needs for the Next Millennium,” 1999.

[EURO04] T. Unterschütz, “Operations Support Systems for NGN: Co-ordinating Actions for Telecoms,” 2004.

[EURO06] European Communications, “Data Management/NGOSS – SID to the rescue,” 2006. [Online]. Available at http://www.eurocomms.com/features/111083/Data_management%252FNGOSS_-_SID_to_the_rescue.html

[FIEL00] R. T. Fielding, “Architectural Styles and The Design of Network-based Software Architectures,” PhD thesis, University of California, Irvine, 2000. [Online] Available at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[FIEL07] R. T. Fielding, “A little REST and Relaxation,“ in proceedings of the International Conference on Java Technology (JAZOON07), Zurich, Switzerland, June 2007.

[FOWL95] J. Fowler, "TMN-based Broadband ATM Network Management," IEEE Communications Magazine, Vol. 33, Iss. 3, pp. 74-79, March 1995.

[FUEN95] L. A. de la Fuente, M. Kawanishi, M. Wakano, T. Walles and C. Aurrecoechea, “Application of the TINA-C Management Architecture,” in proceedings of the 4rth International Symposium on Integrated Network Management, pp. 424-435, 1995.

[G.774.05] ITU-T Recommendation G.774.05, “General aspects of digital transmission systems,” July 1995.

[GALL05] C. R. Gallen and J. S. Reeve, “Investigating the Feasibility of Open Development of Operations Support Solutions,” in proceedings of the 9th IFIP/IEEE International Conference on Integrated Network Management, 15-19 May 2005, Nice, France, 2005.

[GOLD93] G. Goldszmidt, “On Distributed System Management,” in proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Distributed Computing, Toronto, Ontario, Canada, Vol. 2, pp. 637-647, 1993

[GOTT02] K. Gottschalk, S. Graham, H. Kreger and J. Snell, “Introduction to Web Services Architecture,” IBM Systems Journal, Vol. 41, Iss. 2, 2002.

[GREE01] J. H. Green, The Irwin handbook of telecommunications management, 3rd edition, McGraw-Hill, 2001

[HARO04] E. R. Harold and W. S. Means, XML in a Nutshell, O'Reilly Media, 3rd edition, 2004.

[HARR99] D. Harrington, R. Presuhn and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks," IETF RFC 2571, April, 1999.

[HARR02] D. Harrington, R. Presuhn and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," IETF RFC 3411, December 2002.

[HASS09] R. Hassan, R. Razali, S. Mohseni, O. Mohamad and Z. Ismail, "Architecture of Network Management Tools for Heterogeneous System," International Journal of Computer Science and Information Security, Vol. 6, No. 3, pp. 31-40, 2009.

[HENN06] M. Henning, "The Rise and Fall of CORBA," Association for Computing Machinery, 2006.

[HERMES] Hermes JMS http://www.hermesjms.com/confluence/display/HJMS/Home

[HP07] Hewlett-Packard White Paper, HP Service Delivery Platform, Oct 2007.

[HOHP04] G. Hohpe and B. Woolf, Enterprise Integration Patterns, Pearson education, 2004.

[HUBA98] J. P. Hubaux, C. Gbaguidi, S. Kopperhoefer and J.Y LeBoudec,"The impact of the Internet on Telecommunication Architectures," Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol. 31, Iss. 3, February 1999.

[IBM] International Business Machines, http://www.ibm.com/uk/en/

[IBMU] http://www-01.ibm.com/software/solutions/webservices/uddi/

[IEC02] Operations Support Systems 2002: Enabling the Next Generation Network By International Engineering Consortium]

[INTGR] IntelliGrid Electric Power Research Institute http://intelligrid.epri.com/

[IETF] Internet Engineering Technology Taskforce, http://www.ietf.org/

[ISO93] ISO/IEC10165-1, “Information Processing Systems - Open Systems Interconnection – Structure of Management Information - Part 1: Management Information Model”, Geneva, 1993.

[ITU-T] International Telecommunication Union - Telecommunications, http://www.itu.int/ITU-T/.

[J2EE] Java 2 Platform, Enterprise Edition (J2EE) Overview http://java.sun.com/j2ee/overview.html

[JAXP] Java API for XML Processing JAXP, https://jaxp.dev.java.net/

[JBOSS] JBoss Community, http://www.jboss.org/

[JCON] The Java Monitoring and Management Console JConsole http://openjdk.java.net/tools/svc/jconsole/

[JCP] Java Community Process, “JSR 21: JAINTM JCC Specification,” http://jcp.org/en/jsr/detail?id=21

[JENK06] I. Jenkins, “NGN Control Plane overland and its management, MSF technical report, MSF-TR_ARCH-007-Final,” 2006.

[JDK] JDK, http://www.oracle.com/technetwork/java/javase/downloads/index.html

[JMAIL] JavaMail http://www.oracle.com/technetwork/java/index-jsp-139225.html

[JOHN92] D. Johnson,”NOC Internal Integrated Trouble Ticket System Functional Specification Wishlist,” IETF RFC 1297, January 1992.

[JOSU07] N. M. Josuttis, SOA in Practice: The Art of Distributed System Design, O'Reilly Media, 2007.

[JRMI] Remote Method Invocation http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[JSON] JavaScript Object Notation, http://www.json.org/

[JSP] JavaServer Pages Technology, http://java.sun.com/products/jsp/

[JSR296] JSR 296, http://java.sun.com/developer/technicalArticles/javase/swingappfr/

[JING09] L. Jing-min and L. Zi-hui, "Design and Implementation of an Embedded Devices Monitoring System Based on SNMP," 2009. [Online]. Available at http://en.cnki.com.cn/Article_en/CJFDTOTAL-XDJS200906054.htm.

[KAST91] F. Kastenholz, "SNMP Communications Services," IETF RFC 1270, October 1991.

[KAVA00] R. Kavasseri and B. Stewart, "Distributed Managament Expression MIB," IETF RFC 2982, October 2000.

[KNUT05] K. Knüttel, T. Magedanz, and D. Witszek, "The IMS Playground @ FOKUS: An Open Testbed for Next-Generation Network Multimedia Services," in proceedings of the 1st International Conference on Testbeds and Research Infrastuctures for the Development of Networks and Communities, Tridentcom 2005, IEEE CS Press, pp. 2-11, 2005.

[KOTS08] K. Kotsopoulos, P. Lei and Y. F. Hu, "A SOA-based Information Management Model for Next-Generation Network," in proceedings of the International Conference on Computer and Communication Engineering, ICCCE 2008, 13-15 May 2008, Kuala Lumpur, Malaysia, pp. 1057-1062, 2008.

[KREG01] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," IBM Software Group, May 2001.

[KREG05] H. Kreger, „Management Using Web Services: A Proposed Architecture and Roadmap, tech report,

IBM, Hewlett-Packard, and Computers Assoc., June 2005.

[LAGH09] K. Laghari, I. Griba ben Yahia amd N. Crespi, “Analysis of Telecommunication Management Technologies,” International Journal of Computer Science and Information Technology, Vol. 2, Iss. 2, pp. 152-166, 2009.

[LEW99] D. Lewis, “TeleManagement Forum Business Process to TINA-C Business Role Mapping.” FlowThru Consortium, 1999. [Online]. Available at http://www.cs.ucl.ac.uk/research/flowthru/content/bmp-role-map/bmp-role-map.pdf.

[LI05] M. Li and K. Sandrasegaran, “Network management challenges for next generation networks,” The IEEE Conference on Local Computer networks, 2005, 30th Anniversary, pp. 592-598, 2005.

[LOPE00] R. P. Lopes and J. L. Oliviera, “Managing Mobile Agents with SNMP,” 2000. [Online]. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.3028

[M.2301] ITU-T Recommendation M.2301, “Performance objectives and procedures for provisioning and maintenance of IP-based networks,” July 2002.

[M.3010] ITU-T Recommendation M.3010, “Principles for a telecommunications management network”, November 2005.

[M.3016] ITU-T Recommendation M.3016, “TMN Security Overview,“ June, 1998.

[M.3020] ITU-T Recommendation M.3020, “Management interface specification methodology,” July 2007.

[M.3050.1] ITU-T Recommendation M.3050.1, “Enhanced Telecom Operations Map: The Business Process Framework (eTOM), “March 2007.

[M.3060] ITU-T Recommendation M.3060/Y.2401, “Principles for the Management of Next Generation Networks,“ March, 2006.

[M.3100] ITU-T Recommendation M.3100, “Generic network information model,” April 2005.

[M.3120] ITU-T Recommendation M.3120, “CORBA generic network and network element level information model,” January 2001.

[M.3190] ITU-T Recommendation M.3190, “Shared information and data model (SID),” July 2008.

[MacV06] L. MacVittie, “Taking a REST from SOAP,” Network Computing, Vol. 17, No. 20, pp. 25-26, October, 2006.

[MAGE03] T. Magedanz, A. Hafezi, and R. Wechselberger, “Practical Experiences in Deploying OSA/Parlay on Top of UMTS and 3G Beyond Networks—The 1st Project Opium and the FhG FOKUS 3Gb Center,” in proceedings of the International Conference on Intelligence in Networks (ICIN 2003), Adera, 2003, pp. 65-70.

[MAGE06] T. Magedanz, M. Sher: “IT-based Open Service Delivery Platforms for Mobile Networks: From CAMEL to the IP Multimedia System”, in Mobile Middleware, P. Bellavista, A. Corradi (Eds), Chapman & Hall/CRC Press, pp. 999 - 1026, January 2006.

[MAGE07] T. Magedanz, N. Blum and S. Dutkowski, “Evolution of SOA Concepts in Telecommunications,” Computer, Vol. 40, Iss. 11, pp. 46-50, November 2007.

[MART00] J. P. Martin-Flatin, “Web-Based Management of IP Networks and Systems,”Ph.D. thesis, Swiss Fed. Inst. Of Technology, Lausanne (EPFL), October 2000. [Online]. Available at http://library.epfl.ch/en/theses/?nr=2256.

[MART02] J.P. Martin-Flatin, Web Based Management of IP Networks & Systems, John Wiley & Sons Ltd, 2002.

[MAUR01] D. Mauro and K. J. Schmidt, Essential SNMP, O’Reilly Media, 2001.

[MC4J] MC4J Console, mc4j.org/

[McCL91] K. McCloghrie and M Rose, “Management Information Base for Network Management of TCP/IP-based Internets: MIB-II,” IETF RFC 1213, March 1991.

[McCL99] K. McCloghrie, D. Perkins and J. Schoenwaelder, “Structure of Management Information Version 2 (SMIv2),” IETF RFC 2578, April1999.

[MIDD97] Middleware white paper, 1997. [Online]. Available at http://web.cefriel.it/»alfonso/WebBook/Documents/isg midware.pdf.

[MORA02] L. Morand and S. Tessier, “Global mobility approach with Mobile IP in “All IP” networks”, in Proceedings of the IEEE International Conference on Communications (ICC), 28 April – 2 May, 2002.

[MORI08] The Moriana Group, “SDP 2.0: Service Delivery Platforms in the Web 2.0 Era,” Sepember 2008. [Online]. Available at available http://www.morianagroup.com/.

[MySQL] MySQL http://www.mysql.com/

[NAKA95] N. Nakamura, N. Kashimura and K. Motomura, “CMIP to SNMP Translation Technique Based On Rule Description,” in proceedings of the 4th International Conference on Computer Communications and Networks (ICCCN '95), 20-23 September, Las Vegas, USA, pp. 266-271, 1995.

[NARA00] N. Narang and R. Mittal, “Network Management for Next Generation Networks,” in Proceedings of the 8th International Conference on Advanced Computing and Communications, Cochin, India, 14-16 December, 2000.

[NGOSS04] J. Strasser, J. Fleck, J. Huang, C. Fauer and T. Richardson, “TMF White Paper on NGOSS and MDA,” version 1.0, BPTrends April, 2004

[NINO] NINO http://nino.sourceforge.net/nino/index.html

[OASIS06] OASIS, “Reference Model for Service Oriented Architecture,” February 2006.

[OASIS07] OASIS, “WS-SecurityPolicy 1.2,” July 2007.

[OASIS07] OASIS, “Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1,”June 2007.

[OASIS08a] OASIS, “UDDI Version 3.0.2,” July 2008

[OASIS08b] OASIS, OASIS Telecom: Adopting SOA for Telecom Workshop, 2008.

[OECD07] Organisation for Economic Co-operation and Development, “Participative Web: User-Created Content,” OECD DSTI/ICCP/IE(2006)7/FINAL, Apr 2007.

[OHNI07] H. Ohnishi, Y. Yamato, M. Kaneko, T. Moriya, M. Hirano and H. Sunaga, “Service Delivery Platform for Telecom-Enterprise-Internet Combined Services,” in proceeding of the IEEE Global Tellecommunications Conference, GLOBECOM ’07, pp. 108-112, 2007.

[OMA] Open Mobile Alliance, http://www.openmobilealliance.org/

[OPENNMS] The Open NMS Project, http://www.opennms.org/

[OPENVIEW] HP OpenView, http://www.managementsoftware.hp.com/

[ORACLE] JSLEE and the JAIN Initiative http://java.sun.com/products/jain/

[OSA/PARLAY] Open API Solutions http://www.openapisolutions.com/brochures/OSAParlayOverview.pdf

[PANT08] H. Pant, C. Kelvin-Chu, S. H. Richman, A. Jrad and G. P. O’Relly, “Reliability of next-generation networks with a focus on IMS architecture,” Bell Lab Technical Journal , Vol. 12, Iss. 4, pp. 109-125, 2008. [Online]. Available at http://onlinelibrary.wiley.com/doi/10.1002/bltj.20270/pdf.

[PARLAY4] The Parlay Group, Inc., “Parlay 4.0: Parlay X We Services Specification,” version 1.0.1, June 2004.

[PAUT08] C. Pautasso, O. Zimmermann and F. Leymann, “RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision,” in proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 805-814, 21-25 April, Beijing, China, 2008.

[PAV97] G. Pavlou and D. Griffin, “Realizing TMN-like Management Services in TINA,” Journal of Network and Systems Management, Vol 5, No.4, pp. 437-457, 1997.

[PAVL98] G. Pavlou, T. Mota, F. Steegmans and J. Pavón, “Issues in Realising the TINA Network Resource Architecture,” 1998. [Online]. Available at citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.8217

.

[PAV00] G. Pavlou,”Using Distributed Object Technologies in Telecommunications Network Management,” IEEE Journal on Selected Areas in Communications, Vol. 18, Iss. 5, 2000.

[PEL03] A. Peltz, “Web Services Orchestration and Choreography,” Computer, Vol. 36, Iss. 10, pp 46-52, Oct 2003.

[PINU04] H. Pinus, “Middleware: Past, present a Comparison,” June 2004.

[PRAS04] A. Pras, T. Drevers, R. van de Meent and Quartel, D.A.C. “Comparing the Performance of SNMP and Web Services-Based Management”, IEEE Transactions on Network and Service Management, Vol.1 (2), pp. 72-82, 2004.

[PRES02] R. Presuhn, J. Case, K. McCloghrie, M. Rose and S. Waldbusser, “Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP),” IETF RFC 3416, December 2002.

[POSTGRE] PostgreSQL, http://www.postgresql.org/

[RAMA97] S. Ramaswamy, “TMN Applied to an Integrated management system for dod communications networks using military and commercial satellites,” in MILCOM 97 Proceedings, Monterey, CA, USA, Vol.2, 2-5 November., pp. 922-928, 1997.

[REDL98] J. P . Redlich, M. Suzuki and S. Weinstein, S., “Distributed Object Technology for Networking,” IEEE Communications Magazine, Vol. 36, No. 10, , pp. 100-111, October 1998.

[ROSE90] M.T. Rose and K. McCloghrie, “Structure and Identification of Management Information for TCP/IP-based Internets,” IETF RFC1155, May 1990.

[SART95] S. Sartzetakis, C. Stathopoulos, V. Kalogeraki and D. Griffin, “Managing the TMN,” in proceedings of the 3rd International Conference on Intelligence in Broadband Services and Networks: Bringing Telecommunication Services to the People, Heraklion, Crete, Greece, October 1995.

[SASA09] K. Sasaki, “Standardization Trends in the TeleManagement Forum (TM Forum),” NTT Technical Review, Vol. 7, No. 10, 2009.

[SAXON] Saxon: the Java API http://saxon.sourceforge.net/saxon7.0/api-guide.html

[SERV] Java Servlet Technology, http://www.oracle.com/technetwork/java/index-jsp-135475.html

[SERVICEMIX] Apache Service Mix, http://servicemix.apache.org/home.html

[SATM09] SatMagazine, “Insight: The Evolution of Network Mangement,” March 2009. [Online]. Available at http://www.satmagazine.com/cgi-bin/display_article.cgi?number=982110919

[SCHO02] J. Schonwaelder, “Simple Network Management Protocol (SNMP) over Transmission Control Protocol (TCP) Transport Mapping,” IETF RFC 3430, December 2002.

[SIEG02] J. Siegel, “A preview of CORBA 3,” Computer, Vol. 32, Iss. 5, pp. 114-116, 1999.

[SIDH00] S. S. Sidhu and M. K. Sidhu, “Enhanced Services With Intelligent Networks,” Electronics For you, Communications, 2000.

[SNMP4J] The SNMP API for Java, http://www.snmp4j.org/

[STAL98] W. Stalling, “Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards,” The Internet Protocol Journal, Vol. 1, No. 3, 1998.

[STAL99] W. Stalling, SNMP, SNMPv2, SNMPv3 and RMON 1 and 2, third edition, Addison-Wesley, 1999.

[STRA99] F. Strauss, “A Library to Access SMI MIB Information,” [Online]. Available at http://www.ibr.cs.tu-bs.de/projects/libsmi/

[SUBR00] M. Subramanian, Network Management Principles and Practice, Addison- Wesley, 2000.

[SUN96] Sun Microsystems, “Solstice™ CMIP 8.2 Administrator’s Guide,” 1996. [Online]. Available at http://dlc.sun.com/pdf/802-5282/802-5282.pdf

[SUNJMS] Sun Microsystems, “Java Message Service,” version 1.1, April 2002.

[SYBA] Sybase, http://www.sybase.co.uk/

[SWIM] SWIM-SUIT, “Identification of Technology and Services
Options,” D.2.2.1, 2008.

[TARK09] S. Tarkoma and J. Kangasharju, Mobile Middleware: Architecture, Patterns and Practice, John Wiley & Sons Ltd, 2009.

[TINA] Telecommunication Information Networking Architecture, http://www.tinac.com/about/about.htm

[TMF053] TMF, “The NGOSS Technology-Neutral Architecture,” TMF 053, Public Evaluation, Version 3.0, April, 2003.

[TMF] Telemanagement Forum, http://www.tmforum.org/browse.aspx

[TOMCAT] Apache TomCat, http://tomcat.apache.org/

[TRIM01] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T’Joens, L. Georgiadis, C. Jacquenet and R. Egan, “A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks,” IEEE Communications Magazine, Vol. 39, Iss. 5, pp. 80-88, 2001.

[VALE99] T. Valeski, Enterprise JavaBeans(TM): Developing Component-Based Distributed Applications, Addison-Wesley Professional, 1999.

[VALL99] A. Vallecillo, “RM-ODP: The ISO Reference Model for Open Distributed Processing,” 1999. [Online]. Available at http://www.enterprise-architecture.info/Images/Documents/RM-ODP.pdf.

[VENI00] I. Venieris, F. Zizza, and T. Magedanz, eds., Object-Oriented Software Technologies in Telecommunications: From Theory to Practice, John Wiley & Sons, 2000.

[VINO97] S. Vinoski, “CORBA: integrating diverse applications within distributed heterogeneous environments,” IEEE Communications Magazine, Vol. 35, Iss. 2, pp. 46-55, 1997.

[W3C98] W3 Consortium, “Document Object Model (DOM) Level 1 Specification,” W3 Consortium Recommendation, October 1998.

[W3C99a] W3 Consortium, “XML Language (XPath) Version 1.0,” W3 Consortium Recommendation, November 1999.

[W3C99b] W3 Consortium, “XSL Tranformations (XSLT) Version 1.0,” W3 Consortium Recommendation, November 1999.

[W3C01] W3 Consortium, “Web Services Description Language (WSDL) 1.1,” W3 Consortium Recommendation, March 2001.

[W3C04] W3 Consortium, “XML Schema Part 1: Structures Second Edition,” W3 Consortium Recommendation, October 2004.

[W3C06a] W3 Consortium, “Web Services Addressing 1.0 – SOAP Binding,”W3 Consortium Recommendation, May 2006.

[W3C06b] W3 Consortium, “Extensible Markeup Language (XML) 1.0,” W3 Consortium Recommendation, August 2006.

[W3C06c] W3 Consortium, Web Services Policy 1.2 - Framework (WS-Policy),” W3 Consortium Recommendation, April 2006.

[W3C07a] W3 Consortium, “SOAP Version 1.2 Part1: Messaging Framework,” W3Consortium Recommendation, April 2007.

[W3C07c] W3 Consortium, “XQuery 1.0: An XML Query Language,” W3 Consortium Recommendation, January 2007.

[W3C07d] W3 Consortium, Web Services Coordination (WS-Coordination) Version 1.1,” W3 Consortium Recommendation, July 2007

[W3C09] W3 Consortium, “XML Schema Definition Language (XSD) 1.1 Part 1: Structures,” W3 Consortium Recommendation, December 2009.

[WALD95] S. Waldbusser, “RemoteNeteork Monitoring Management Information Base,” IETF RFC 1757, February 1995

[WARR89] U. Warrier and L. Besaw, “The Common Management Information Services and Protocol over TCP/IP (CMOT),” IETF RFC 1095, April 1989.

[WARR90] U. Warrier, L. Besaw, L LaBarre and B. Handspicker, “The Common Management Information Services and Protocols for the Internet (CMOT and CMIP),” IETF RFC 1189, October 1990.

[WEIS07] A. J. Weissberger, “In Search for the Next Generation Network,” HPC in the Cloud, 2007. http://www.hpcinthecloud.com/

[WEST00] A. Westerinen and J. Strassner, “Common Information Model (CIM) Core Model Version 2.4,” DMTF, 2000.

[X.690] ITU-T Recommendation X.690, “Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),”ITU, July 2002.

[X.805] ITU-T Recommendation X.805, “Security architecture for systems providing end-to-end communications,”October 2003.

[Y.2011] ITU-T Recommendation Y.2011, “General principles and reference model for Next Generation Networks,” October 2004.

[YATE97] M. Yates, W. Takita, L. Demoudem, R. Jansson and H. Mulder, “TINA Business Model and Reference Points Version: 4.0”, May 1997. [Online]. Available at http://www.tinac.com/specifications/documents/bm_rp.pdf.

[YOON06] J. H. Yoon, H. T. Ju and J. W. Hong, “Development of SNMP-XML translator and gateway for XML-based Integrated Network Management,” International Journal of Network Management, Vol. 13, Iss. 4, pp. 259-276, 2003.

[ZHAN06] Y. Zhang, X. Qui and l. Meng, L., “A Web services-based dynamically cooperative network management architecture,” IEEE Conference in Communications and Networking.2006.

Appendix A : Simple Network Management Protocol Limitations

A.1 SNMPv1 LIMITATIONS

SNMP v1 has many limitations. Extensive research has been done over the past years on the SNMP framework [BEN90], [MART00], [PRAS04]. These studies have exposed the weaknesses and limitations of the SNMP v1 as presented below:

- The security mechanism of SNMPv1 is community based, which is known as trivial authentication. The community name is not encrypted; as a result, it can be easily discovered by an unauthorized person. When the correct community name has been revealed, the unauthorized person could execute the protocol operations. For that reason some SNMPv1 vendors do not want to implement a Set-Request operation. Consequently, SNMPv1 is more appropriate for monitoring NEs rather than controlling them.
- For retrieving large amount of data, such as an entire routing table, SNMPv1 is not a good choice because the manager has to send many requests to the agent in order to acquire these data. This can lead to bandwidth overhead.
- The SNMPv1 traps, which are the notifications that agents send to the manager, are unacknowledged due to the use of UDP protocol.

A critical message from the agent is not ensured that it will be received by the manager.

- SNMPv1 does not provide manager-to-manager communication. There is no mechanism that allows a NMS to be aware of the networks and devices managed by another NMS.
- SNMPv1 does not define enough error codes. The manager could fail to recognise the cause of an error. In many cases, the manager has to successively apply for parts of the original request, in order to find the problem.
- SNMPv1 is not suitable for managing really large networks due to the performance limitations of polling. Based on the polling mechanism, one packet must be sent in order to get one packet of information back. This type of polling results in large volumes of routine messages and generates problematic response times that may not be acceptable.
- SNMPv1 uses a minimal set of protocol operations, and follows a simplified way of managing the network.
- SNMPv1 has insufficient functions for retrieving bulk information, which gives performance problems. Accessing MIB tables containing repeating variables requires successive Get-Next-Request operations to an agent. If the MIB tables are very large, it takes lot of time to complete all the necessary transactions. This is resource-intensive in real time, network bandwidth, and the agent's CPU time.

SNMPv1 is a lightweight protocol that provides management capabilities and does not have any impact on the operation of the device or its performance. In addition, the message size of SNMPv1 is small, which allows for low network overhead to be achieved. The above limitations are the primary reason for implementing the successors of SNMPv1.

A.2 SNMPv2

SNMPv2 is a revised protocol, which includes some enhancements to SNMPv1, but still uses the existing community-based security and the same message format of SNMPv1 [CASE99]. The following section describes the enhancements provided by SNMPv2 protocol.

SNMPv2 provides several improvements to SNMPv1, as stated in the RFC 2570 [CASE99]. These improvements are separated into three basic categories: improvement to SMI, improvement to manager-to-manager capability and improvement to protocol operations. The SNMPv2 SMI extends the SNMPv1 SMI into macros that define object types to include new data types [McCL99]. Another improvement in this category is the new convention that has been provided in order to create and delete conceptual rows in a table. The improvements to protocol operations can be seen in the following bullet points [PRES02]:

- SNMPv2 supports improved efficiency and performance by introducing a Get-Bulk-Request operation to allow the manager to retrieve a large amount of data. Especially, it is well suited for retrieving multiple rows in an MIB table.

- SNMPv2 has expanded the data types to be up to 64 bits compare to 32 bits that SNMPv1 provides [CASE02].
- SNMPv1 does not provide manager-to-manager communication. SNMPv2 provides manager-to-manager communication by introducing an Inform-Request operation that is an acknowledged trap type, in order to facilitate a hierarchical network management system. The Inform-Request operation enables the manager to send a trap type of information to another manager.
- SNMPv2 changes the atomic Get-Response operation to a non-atomic one (Request-PDU), to permit partial responses to a request. For instance, in SNMPv1 Get-Request operation carries more than one variable binding; if an error occurs in one of the variable, then none is returned. Nevertheless, in SNMPv2, the valid ones are returned and the error index is set to the location of the invalid one in the variable bindings. The non-atomic Get-Response reduces the overall management traffic.
- SNMPv2 offers better error handling by defining twelve new error codes and introducing the concept of exceptions in order the users to be informed about the cause of a failed operation [CASE96]. Consequently, this improved error handling results in fewer message exchanges, which are needed to resolve a problem, between the manager and the agent.

SNMPv2 uses the simple and unsecured password-based authentication, known as the community feature, provided in SNMPv1. To fix the security problem, a number of independent groups began to work on a security

improvement [HARR99]. In April 1999, IETF produced a set of proposed standards for SNMPv3. In December 2002, these SNMPv3 specifications and documentation were standardized.

A.3 SNMPv3

SNMPv3 is the newest version of SNMP. It is actually the SNMPv2 plus security [HARR02]. This means that it maintains the same management operations as SNMPv2, but it introduces alignments to SNMP messages to carry proper security parameters that finally make SNMP a secure protocol. This allows encryption of management messages and strong authentication of the senders. The security features added to the third version of SNMP are based on the Used-based Security Model (USM) that provides Confidentiality, Data Integrity, and Authentication functions to the message [BLUM99].

- Confidentiality: Encryption of packets in order to prevent snooping by an unauthorized source. For encryption, SNMPv3 uses the Data Encryption Standard (DES) protocol in order to provide encryption to the encapsulated SNMP packets.
- Data Integrity: Message integrity to ensure that a packet has not been tampered. SNMPv3 uses message digest algorithm (MD5) in order to verify the user on whose behalf the SNMPv3 message was generated and to verify the integrity of the received SNMPv3 message.

- Authentication: To verify that the message is from a valid source. MD5 and Secure Hash Algorithm (SHA) are supported by the SNMPv3.

SNMPv3 is less vulnerable to security attacks [STAL98]. When the agent receives an SNMP request, it can determine that an authorized manager issued the request and that the message was not corrupted by an unauthorized person. SNMPv3 includes a standardized and modularized architecture for SNMP agent implementations. SNMPv3 does not introduce a new specification language. So with SNMPv3, it becomes feasible to use SNMP for applications that have greater security needs than monitoring, such as provisioning applications. SNMPv3 has become much more powerful yet more complex than the original SNMP specification that appeared almost a decade earlier. This reflects greater maturity and also increased agent processing capabilities and availability of more powerful implementation tools. SNMP is the most successful protocol for network management and is implemented based on the principles of simplicity, in order to enable widespread adoption.

A.4 SNMP PRIMITIVES (PDU)

The following table (table A-1) illustrates the SNMP primitives that are implemented by the different versions of the SNMP protocol.

Table A-1: SNMP primitives

SNMP Primitives	Description	SNMP		
		v1	v2	v3
GetRequest	SNMP manager requests information from the SNMP agent (polls the agent)	Yes	Yes	Yes
SetRequest	SNMP manager sends a command to the SNMP agent for reconfiguration of the associated network element	Yes	Yes	Yes
GetNextRequest	SNMP manager requests that the SNMP agent send the next value in a table or matrix.	Yes	Yes	Yes
GetBulkRequest	Manager sends a single request to generate a response from the agent containing a large amount of data	No	Yes	Yes
GetResponse	SNMP agent response to a GetRequest PDU	Yes	No	No
Response	SNMP agent response to a Get type message, confirmation of a Set message or a response to an InformRequest	No	Yes	Yes
Trap	Message sent by the SNMPv1 agent to concerning the occurrence of a given alarm or other predetermined event	Yes	No	No
SNMPv2-Trap	Message sent by the SNMPv1 agent to concerning the occurrence of a given alarm or other predetermined event	No	Yes	Yes
Report	SNMP message containing message in the form of a report	No	Yes	Yes
InformRequest	Trap with an acknowledgement. The SNMP agent can resend the trap message if no response is received in a predetermined time	No	Yes	Yes

Appendix B : Evolution of Middleware Technologies

B.1 DISTRIBUTED OBJECT TECHNOLOGY (DOT)

Middleware is a software component that resides between the applications and the underlying operating systems, network protocol stacks, and hardware. It can be embedded in the application or can be standalone software [SCHA01]. Its primary role is:

- Functionally bridge the gap between application programs and the lower-level hardware and software infrastructure in order to coordinate how parts of applications are connected and how they interoperate and
- Enable and simplify the integration of components developed by multiple technology suppliers

Middleware can help to shield software developers from low-level, tedious, and error-prone platform details, such as socket-level network programming. It also provides reusable functions and a consistent set of higher-level network-oriented abstractions that are much closer to application requirements in order to simplify the development.

A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture. As the size of software system increases, the overall system structure becomes more complex in the

issues: communication protocols, synchronization, data access, scalability, performance, security and etc. Figure B.1 presents the evolution of software architecture. Before 1980 software architecture was mainly monolithic mainframe systems that empowered organizations with appropriate computational resources. These environments had bulky mainframe back-ends served by dumb terminals at front-end.

In the mid 90's, distributed object computing transformed the way in which system is organized. Clients and servers are distributed over computer network on separate hardware but they both reside in the same system. This two-tier client-server architecture introduced fat clients, a personal computer (PC), with intelligence. This allowed the logic and the processing duties to be performed on separated PC and greatly reduced the cost of computing. Later the multi-tier client-server architecture is introduced. This network centric architecture broke the monolithic client executable into components. The application logic distributed among multiple components (some residing on clients, others on servers) reduced the deployment problems by centralizing a greater amount of the logic on servers. Additionally, the Remote Procedure Call (RPC) technology was developed, such as Common Object Request Broker Architecture (CORBA) and Distributed Computing Object Model (DCOM) which allowed remote communication between components residing separately on the client workstations and servers. At the same time, the Internet became the platform for computing due to the introduction of Web browser and the Web. Web Services are emerged as the new important type of distributed systems that is based on service-oriented concept. SOA is not a new

concept [ERL04] but its popularity has increased over the past few years due to the wide adoption of Web Services for SOA implementation. It takes all the best practices from previous architectures and is the next evolutionary step to the realization of dynamically configurable architecture.

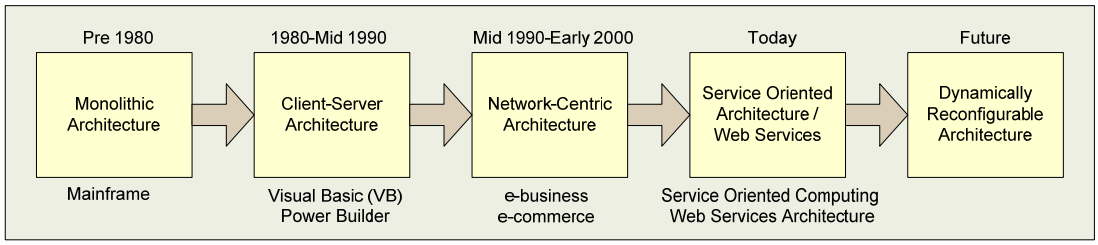


Figure B.1 The evolution of systems architectures

Distributed Object Technology (DOT) introduced the middleware layer concept in order to integrate heterogeneous systems. DOT is the merger of object technology and distributed system technology [PAV00]. This technology reduces the development time and has modular architecture. The distributed system technology is based on the idea that systems are not only networked together as isolated components but they are also coordinated together in a heterogeneous network environment in order to carry out small unit of related tasks [SIEG02].

Three established DOT paradigms exist today: CORBA by Object Management Group (OMG), DCOM by Microsoft and Remote Method Invocation (Java/RMI) created by Java Soft [CORBA], [COM], [JRMI]. These technologies are used as a middleware layer within the management architecture in order to provide integration between different

systems. This section highlights these middleware technologies that are available for integrating telecommunication management systems.

B.2 COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

CORBA, the most well known integration framework adopted by the telecommunication industry, was proposed by The OMG. OMG is an international consortium created in 1989 with the goal to provide solutions for implementing portable software components and platforms that could operate under multiple environments [CORBA]. The architectural approach should interoperate irrespective of the hardware, operating system and programming languages. The outcome of the OMG consortia was the CORBA framework that was standardized in 1993. CORBA implements the concept of interfaces where CORBA objects are encapsulated and are accessible through interfaces. Figure B.2 illustrates the CORBA architecture.

Any relationship between distributed objects has two sides: the client and the server. The server in the CORBA architecture provides a remote interface called Dynamic Skeleton Interface (DSI), and the client calls the remote interface. On the client, the client application includes a reference called Dynamic Invocation Interface (DII) for the remote object. The object reference has a stub method for remote call. The stub is connected to the Object Request Broker (ORB). When the stub calls the method it invokes the ORB's connection capabilities, which forwards the invocation to the server. The most central component of CORBA is the ORB that provides

the common ground for all object interaction within the architecture. ORB supports interactions between services, locating objects, and communication between clients and servers. It is the facilitator for sending and receiving messages between different objects and components in a location-independent and platform-neutral manner. On the server side, the server ORB uses skeleton code to translate the remote invocation into method call on the local object. The skeleton translates the call as well as the parameters to their implementation specific format through the DSI and calls the method that is being invoked. When the method returns, the skeleton code either transforms the information to results or gives error, and sends the results back to the client via the server ORB. DSI is used for dynamically invoking CORBA objects that does not have compile-time knowledge of the type of object it is implementing. DSI interface resides on the server side. On the client side, DII is used to allow dynamic creation and invocation of object requests on the client side.

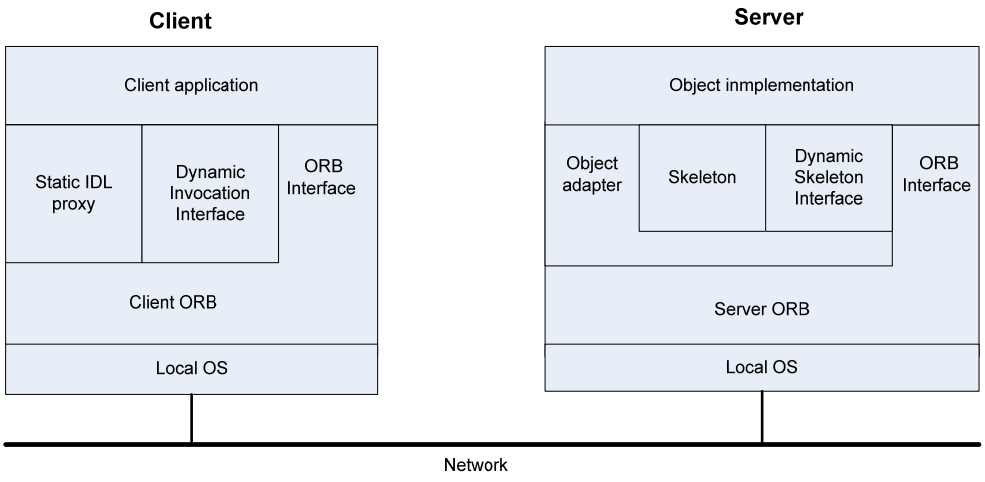


Figure B.2: CORBA Architecture

The ORB objects are accessed through the use of ORB interfaces. ORB interface contains functionality that is required by the clients and the servers. These interfaces are defined by the Interface Definition Language (IDL). This language defines the services offered by objects in a uniform manner, Client applications can use the IDL as the basis for their object invocations. Object implementations need to comply with the definitions of the IDL by implementing the methods defined in the interfaces. In the CORBA architecture the interoperability between ORBs is critical. The Internet Inter-ORB Protocol (IIOP) is used as the interoperability protocol for ORB communication. IIOP uses the TCP/IP protocol to ensure reliable connection, to maintain message ordering, and to provide delivery acknowledgment and connection-loss notification. CORBA uses the Basic Object Adaptor (BOA) API that allows the servers to register their object implementations. The role of the IDL is twofold: Firstly, IDL allows the creation of a definition of the interface of the remote system, independent of any particular implementation and programming language. Secondly, IDL ‘forces’ the developer to define the system in terms of portable data types and operations available in the restricted language of IDL. This guarantees portability because interfaces do not need to be defined through the system’s implementation language. The drawback of using the IDL is that there is no guarantee that the service interface will remain unchanged throughout the lifecycle of the service. Every redeployment of the service means that the contract (interface) needs to change.

CORBA has been adopted extensively by the telecommunication industry [M.3120]. This architectural approach is implemented within telecom products and used as an architectural backbone for integration. The use of CORBA in the TMN environment is studied in various papers [BOHO02], [VINO97], [ADAM98], [REDL98], [TRIM01] over the last decade. CORBA is now a mature technology that has a wide range of tools and support that can deal with heterogeneous systems and integrate legacy systems. CORBA allows interoperability between objects, between programming languages, and between ORBs.

CORBA has limited capabilities as it requires that a system communicating over an ORB must be tight coupled. There is tight coupling between the client and the server. Both must share the same interface, with a stub on the client-side and the corresponding skeleton on the server-side. The management architecture for Next Generation Networks, on the other hand, must be built up as decoupled distributed systems in order to be able to provide flexibility and scalability for the demands of NGN's heterogeneous environment. From this perspective, service providers and Independent Software Vendors (ISVs) have recognized that CORBA cannot be relied upon as the integration backbone for the NGN management [SIEG02]. Moreover, the IIOP protocol, which is the heart of CORBA objects communication, does not offer the characteristics required to access the Internet. A solution to overcome this difficulty is to use HTTP tunnelling, which encapsulates IIOP messages in HTTP frames. This technique has been developed by certain companies such as Sybase [SYBA] and Borland [BORL] but is still immature and has not been

standardized by the OMG. In addition, this solution forces the communicating parties (other organizations) to use CORBA ORB in their infrastructure in order to make possible the communication.

B.3 DISTRIBUTED COMPONENT OBJECT MODEL (DCOM)

Component Object Model (COM) technology is the foundation of Microsoft's attempt to enable communication between reusable software components. DCOM is the distributed version of COM that extends the component over a network environment. Due to COM binary specifications, DCOM components can be written in various programmable languages such as Java, C++. DCOM uses the Object-oriented Remote Procedure Call (ORPC) as its application level protocol for supporting remote objects. Microsoft Interface Definition Language (MIDL) is used for defining interfaces and Service Control Manager (SCM) is used for the location and the activation of an object in the DCOM architecture. DCOM is a language independent platform but available only on windows operating platforms. This limitation makes DCOM unsuitable for cross-platform environments; as a result it is not considered for managing telecommunication networks [CHUN98].

B.4 REMOTE METHOD INVOCATION (RMI)

RMI is standardized by Java Soft [JRMI] and relies upon the Java paradigm; it means that both client and server must be implemented in Java in order to communicate. RMI applications consist of two separate programs: a server and a client. RMI provides the mechanisms by which

the server and the client communicate and pass information back and forth. Java RMI establishes inter-object communication. If a particular method is performed on a remote machine, Java provides the capability through the RMI to make the method appear as it is performed on the local machine. This technology uses the JRMP (Java Remote Method Protocol) for remote object communication. Java/RMI is based upon the concept of Java object serialization that is used to marshal and demarshal objects as streams, while the Java Virtual Machine (JVM) enables the object location and activation. Furthermore, RMI can support diverse platforms and operating systems. By using Java/RMI, the development of distributed applications is fast and simple. Due to its dependence over the Java paradigm, it is not suitable for integrating heterogeneous environments.

B.5 LIMITATIONS OF THE DISTRIBUTED OBJECT TECHNOLOGY (DOT)

The use of distributed object technologies in Telecommunication management has been the subject of intensive research over the last years [TRIM01], [ADAM98], [REDL98], [M.3120]. Middleware technologies such as CORBA, DCOM, and RMI are paradigms for integrating data and services. The drawback for those technologies is the interoperability among different system components residing on different platforms that is weak and difficult to achieve. CORBA and DCOM for example, cannot communicate unless there is a bridge between them and this is due to the different communication protocols that they are using (CORBA uses IIOP and DCOM utilizes ORPC).

The following table (Table B-1) shows the differences between RMI, CORBA and DCOM.

Table B-2: Characteristic of the Distributed Object Technologies

	RMI	DCOM	CORBA
Programming language	Operate only with Java systems. No support for systems implemented on legacy or future languages.	Support multiple languages	Support multiple languages
Interface definition	No specific language for interface description	Microsoft Interface definition Language (MIDL)	Interface Definition Language (IDL)
Communication protocol	Object Remote Procedure Call	Java Remote method Protocol	Internet Inter-ORB Protocol
Object location and activation	Service Control Manager (SCM)	Java Virtual Machine (JVM)	Object Request Broker for Location and Object Adaptor for Activation
Platform constraints	Independent	Operates only in Microsoft and Solaris platforms	Independent

ITU has adopted CORBA technology to solve the interoperability problems that exist due to the multi-vendor environment but CORBA is difficult to seamlessly traverse firewalls, which is crucial for applications that need to span across enterprises. A proposed solution is a special security gateway, which adds an IIOP Domain Boundary Controller component to the firewall. This approach is not standardized and not widely used [HENN06].

DOT technologies use message passing in any distributed systems. CORBA messages are IIOP encoded, DCOM uses Java Remote method

Protocol and RMI uses the Object Remote Procedure Call for the message encoding. The messages underneath are based on the flow of bytes that are received by TCP/IP and ultimately reformed into a packet that is sent to a server stub. These messages are based on objects and method invocations that put restrictions to the higher level requirements that a service needs to provide. The developer of the stub dispatching code knows about the higher levels requirements that a service needs to provide, but he is restricted to objects that are specified by the service. The code of the distributed objects is tailored specifically to the end receiving object. The result is that in case of a change in the object interface, the dispatching code needs to be changed. This shows that the interfaces in the DOT architectures are tightly-coupled to the objects that a service provides. When the object changes, then the interface needs to be coded again both for service consumer and service provider.

Appendix C : Service Oriented Architecture

C.1 From Distributed Approach to Service Oriented Approach

Service-based architectural approach is a natural evolution of application development. Service-oriented platforms align business processes with coarse-grained services. Service granularity depends on the functionality that a service exposes. For instance, in distributed architectures such as CORBA-based architectures, functionality is exposed as remote objects. Objects hide the behavior and the data exchanged between applications. One method calls a particular object that exposes a particular functionality. Consequently, one object forms a fine-grained service because the functionality that provides has a small amount of business-process usefulness. Fine-grained services address a relatively small unit of functionality or exchange a small amount of data among applications. As a result, they require multiple invocations of operations to achieve a simple process but multiple invocations, add extra overhead to the network. When grouping together large number of objects, although access to objects is controlled through interfaces, the granularity at the object level still makes dependencies between them difficult to control in large systems.

In contrast, coarse-grained services abstract large unit of functionality within a single interaction based on messaging paradigm that formulates the concept of service orientation.

Coarse-grained services can formulate business functions that when working together are able to achieve a business goal. The services participating in an SOA communication exchange messages through documents based on XML. Document-based services exchange large coarse-grained documents (messages) among applications that allow loose-coupling communication. These services can offer business-based transactions that add value to business needs.

Figure C.1 illustrates the evolution of application development paradigms over the years. Applications have evolved over the years from tightly-coupled to more loosely-coupled providing more flexibility and adaptability.

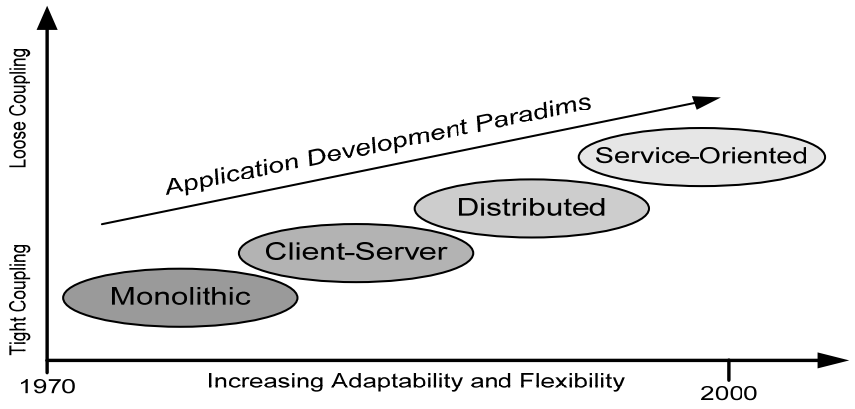


Figure C.1: Application development shifts

Table C-1 shows the movement from distributed architecture to Service Oriented Architecture.

Table C-1: Differences between Distributed Architectures and SOA

DOT-based Approach	SOA-based Approach
Function Oriented	Business Process Oriented
Designed to Last	Designed to Change
Cost Centered	Business Centered
Application Block	Service Orientations
Tight Coupling	Loose Coupling
Homogeneous Technology	Heterogeneous Technology
Object Oriented	Message Oriented

Coarse-grained services interact with each other via self-contained messages that minimize the service dependencies and allow loose-coupling. Loose-coupling deals with the requirements of scalability, flexibility and fault tolerance [ERL05]. The aim of loose coupling is to minimize dependencies among applications. With fewer dependencies, modifications or faults in one system will have fewer consequences on other systems. The main concept of loose-coupling is that two communicating parties (systems or services) make minimal assumptions about each other; the less the applications need to know about each other to cooperate properly the better. Loosely coupled services can be modified independently, which means that if changes are made within one service then the coupled service will not be affected and will not enforce changes. Loose coupling principles make an integration solution more flexible and change tolerant due to the fact that it is based on messaging. Flexibility derives from the fact that connected services do not have to be adjusted

after changes are made in one of the systems taking part in the communication.

Tight coupling systems use local method invocation for communicating with each other. The local method invocation has restrictions and is not capable of providing integration capabilities to the implementation. These restrictions are the following:

- The calling method must be written in the same programming language as the called method.
- The method must run in the same process.
- Both calling and called method must use the same internal data representation format.
- The exact number and type of the arguments of called method must be known.

In table C-2, the differences between tight coupling and loose coupling are listed.

Table C-2: Tight coupling versus Loose coupling

	Tight coupling	**Loose coupling**
Physical connections	Point-to-point	Via a mediator
Communication style	Synchronous	Asynchronous
Data model	Common complex types	Simple common types only
Interaction pattern	Navigate through complex object trees	Data-centric, self contained messages
Control of process logic	Central control	Distributed control
Binding	Statically	Dynamically
Platform	Strong platform dependencies	Platform independent

SOA constitutes a very promising approach for integrating enterprise applications. The most general principles of the term ‘service’ in SOA are:

- Service is a view of a resource (e.g. a software asset, business, a hard disk), basically anything that provides some capability. Implementation details are hidden behind the service interface.
- The communication among services is based on messages. The structure of the message and the schema, or form, of its contents is defined by the interface.
- Services are stateless. This means that all the information needed by a service to perform its function is encapsulated in the messages used to communicate with it.

Services discover and communicate with each other using the publish, find, bind [ERL05] paradigm. A service publishes its interface definition to the network, a service consumer finds the definition and by using the information in the definition, is able to bind (resolve the address and send messages) to the service. An important aspect of SOA is the just-in-time integration of applications facilitated by these three operations. In other words, the interface definition, which describes the form of messaging combined with facilities for publishing and discovering it, enables late-binding between entities to create dynamic aggregations of services.

SOA actually provides a high level of scalability and flexibility that is required in heterogeneous environments. The main drivers for SOA-based architectures are to facilitate the growth of large scale enterprise systems, to facilitate provisioning and use services in order to reduce the costs in the organization’s cooperation. Through these drivers, SOA-based

architectures have the ability to scale and evolve, making these architectures adaptable to the different needs of specific domain or process. Moreover, SOA encourages the architectures to become more agile and responsive than architectures built on an exponential number of pair-wise interfaces [OASIS06]. Therefore, SOA can provide a solid foundation for telecommunication business agility and adaptability.

According to [OASIS06] SOA is "*a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"*. These distributed capabilities are supporting a solution for the business needs of an entity or other collaborative parties. In this context, services are the mechanisms by which business needs and capabilities are brought together. Services are using service description that contains the necessary information to interact with other services. A service description describes the service inputs, the service outputs, and the associate semantics of that service. In general, entities are people and organizations that offer capabilities and act as service providers. Entities with needs and are making use of services are referred to as service consumers. Service description allows potential consumers to decide if the service is suitable for their needs and establishes whether a consumer meets any requirements applied by the service provider.

C.2 SOA UNDERLYING TECHNOLOGIES

The technology that enables service-oriented implementations is the Web Services technology. Web Services are interfaces describing a collection of operations that can access the network through standardized XML

messages. Web Services use a standard, formal XML notion (its service description) which covers all the details needed to interact with the service, including transport protocols, message formats and location. Services can be independent from the software or hardware platform on which they are implemented and they are independent from the programming language in which they are written. This happens due to the fact that the interface hides the implementation details of the service. Hiding the implementation details allow Web Services to be loosely coupled, with cross-technology implementations. Web Services perform a specific task or a set of tasks/operations. They can be used independently or with other Web Services to complete a business transaction or a complex aggregation [KREG01]. Web Services provide a way of communication among applications running on different operating systems, written in different programming languages and using different technologies whilst using the internet as their transport.

C.2.1.1 WEB SERVICES

Figure C.2 demonstrates the basic Web Service model and the interaction between its components [GOTT02].

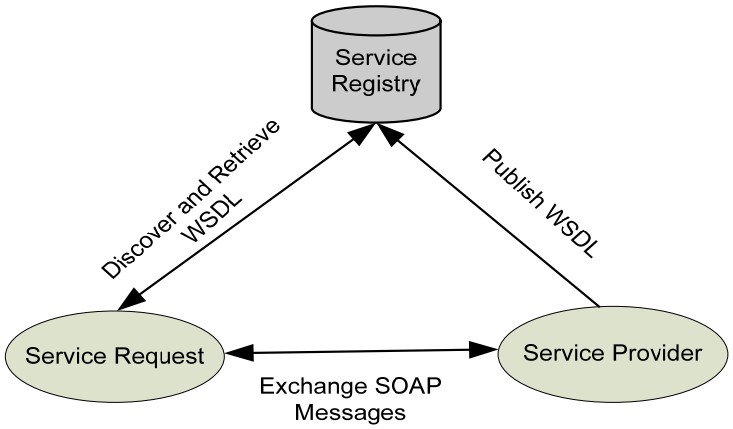


Figure C.2: Find bind and execute paradigm

As can be seen in figure C.2 the Web Services involve three different interactions. Those interactions use the publish, find, and bind paradigm. The first interaction is between the service provider and the service registry. The service provider hosts a network-accessible software module (an implementation of a Web service). It publishes the service description (WSDL) for the Web Service to a service registry (UDDI). The second interaction is between the service requestor and the service registry. The former retrieves the service description by using the find operation from the service registry. The last interaction is between the service requestor and the service provider, in which the former uses this service description in order to interact with the service provider by using the bind operation. Due to the fact that the roles of the service provider and the service requestor are logical constructs, the service can display characteristics of both.

XML, SOAP, WSDL and UDDI, which are the technologies that allow the creation of Web Services and are the underlying technologies that will enable the Service-Orientated implementations.

C.2.1.2 EXTENSIBLE MARKUP LANGUAGE (XML)

The XML is a World Wide Web Consortium's (W3C) recommended [W3C06b] general-purpose, simple, flexible and text format markup language for creating special-purpose markup languages, able to describe many different kinds of data. XML is a method of exchanging information between applications in documents that simultaneously identifies the data fields and contains the data in those fields. XML documents have been widely accepted due to their ability to define documents or schemas for application domains. The easy readability of XML documents by humans has also aided acceptance [CARE02a]. The main purpose of XML is to facilitate data sharing across different systems, particularly systems that are connected via the Internet. Languages that are based on XML (i.e., RDF/XML, SVG, RSS, XHTML and Atom) are defined in a formal way, enabling programs to modify and validate documents in these languages without previous knowledge of their particular form. XML documents represent data objects that have a hierarchical structure. This hierarchical structure must exist for each XML document and is called XML tree structure. The XML tree structure consists of nodes also called elements and the leaves of the tree structure contain other nodes that are referred to as children nodes. XML can be seen as a concrete syntax for describing

such tree structures using mark-up texts. An example of an XML document is as follows (figure C.3):

```
<NetworkElement>
<ManagementSystem>NMS1</ManagementSystem>
<Events>
        <Fault>
        <Location>Router at IP 143.53.20.78</Location>
        <Description>Router is Down</Description>
        </Fault>
        <Fault>
        <Location>Server at IP 143.53.20.53</Location>
        <Description>Memory is critical</Description>
        </Fault>
</Events>
</NetworkElement>
```

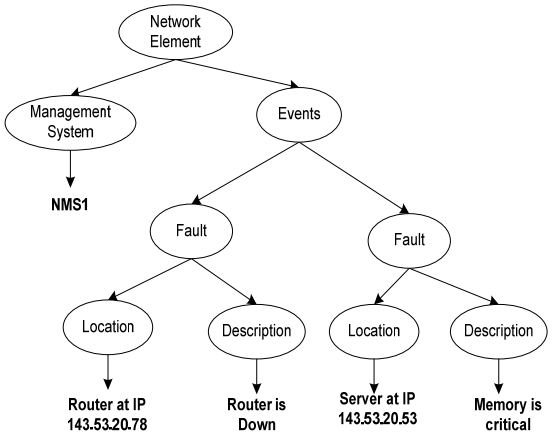


Figure C. 3: Sample of a well-formed XML message

Figure C.3 demonstrates a well-formed XML message. There are two levels of correctness that can distinguish an XML document:

- *Well-formedness* which applies to documents that obey the necessary and sufficient syntactic condition for being interpreted as tree.
- *Validity* which applies to documents that conform to the additional constraints described by a schema.

XML is a family of technologies that have been standardized by the W3C. Figure C.4 illustrates the relationship between the XML specifications.

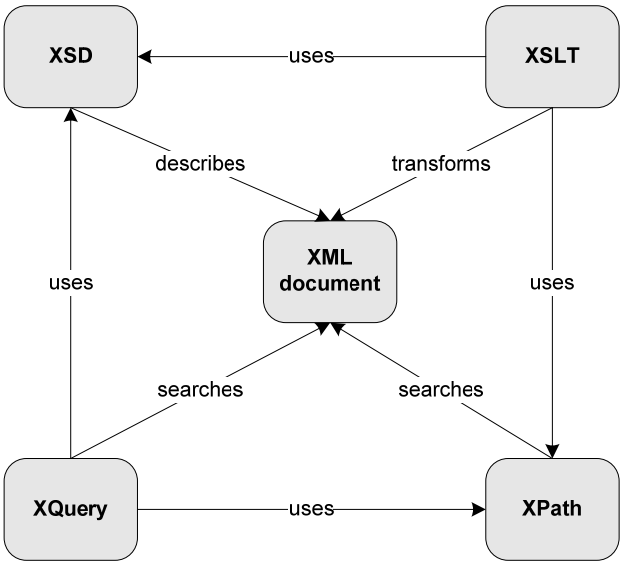


Figure C.4: Relationship between XML specifications

XML Schema Definition Language (XSD) [W3C04], [W3C09] is a data modeling language for XML documents. XSD provides the structural and validation-related features in order to describe an XML document. The schema document expresses a set of rules to which an XML document must conform in order to be considered valid according to that particular schema. The XML schema document is flexible and extendible that is capable of containing multiple schemas documents that can be combined or individually processed. Each schema can be dynamically extended with supplementary constructs. This allows schemas to adapt different data representation requirements.

Extensible Stylesheet Language Transformation (XSLT) [W3C99b] performs XML message transformation. It allows for efficient conversion of XML documents into a number of different output formats. XSLT manipulates, and filters the XML document data to provide alternative

views and versions of information for any number of document transformation scenarios.

Applications that storing and exchanging information based on XML messages, require to intelligently query them. One of the great strengths of XML is its flexibility in representing many different kinds of information from diverse sources. To exploit this flexibility, XML query is required in order to provide features for retrieving and interpreting information from these diverse sources. XML Query language (XQuery) [W3C07c] is a query and functional programming language that is designed in order to query collections of XML documents. XQuery is W3C recommendation that extracts and manipulates data from XML documents.

XML Path Language (XPath) [W3C99a] is the standard language for selecting nodes in XML documents. It is based on a description of paths, by series of steps to be followed in order to reach the selected nodes. For instance, consider the expression: //NetworkElement[Events]/Fault. XPath considers all the NetworkElement nodes in an XML document, tests whether these nodes have an Event child node ([…] defines test expression), and if it is true, output their Faults. Moreover, XPath allows filters to be applied in these steps. Filter is a Boolean combination of path expressions, and is satisfied if a node matches the combination. XSLT uses XPath expressions to match and select particular elements in an XML input document for copying into an output XML document.

C.2.1.3 SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

The communication between services in the SOA concept is message-based, and it should be standardized so that all services can use the same format and transport protocol. SOAP is the standard transport protocol for messages processed by Web Services [BIH05]. This protocol exchanges XML-based messages over a computer network, using Hypertext Transport Protocol (HTTP) or Java Messaging Service (JMS). SOAP is an XML-based protocol that exchange information in a decentralized, distributed environment. It consists of three parts:

- Envelope: It defines the framework for describing a message contains and how to process it.
- A set of encoding rules: The encoding rules are used in order to express the instances of application-defined data types.
- A convention for representing Remote Procedure Calls (RPC) and responses.

SOAP forms the foundation layer of the Web Services stack, providing a basic messaging framework that abstract layers can build on. It enables applications running on different operating systems, with different technologies and programming languages to communicate. SOAP is fundamentally stateless and a one way message exchange paradigm, but applications can make more complex message exchange patterns by using application specific information inside the SOAP envelope or by using features provided by the underlying protocols. RPC is the most common type of messaging pattern in SOAP, where the network node A (i.e. client) sends a request message to the network node B (i.e. server), and the network node B immediately sends a response message to the

network node A. From a network transport perspective, using the SOAP over HTTP gives the ability to the SOAP messages not to be filtered by the network firewalls whereas, using other distributed protocols like DCOM or GIOP/IIOP the messages are normally filtered by firewalls [W3C07a]. Another method of transporting SOAP messages is through the JMS protocol that allows asynchronous communication. Next sections present the use of two different approaches for exchanging SOAP messages.

SOAP messages are contained in the SOAP envelope, which consists of an optional header, and a body. The header contains extension to the SOAP protocol (WS-*) or application specific information (e.g. authentication, payment). The SOAP body contains the actual SOAP message intended for the endpoint of the message. Figure C.5 illustrates the SOAP message.

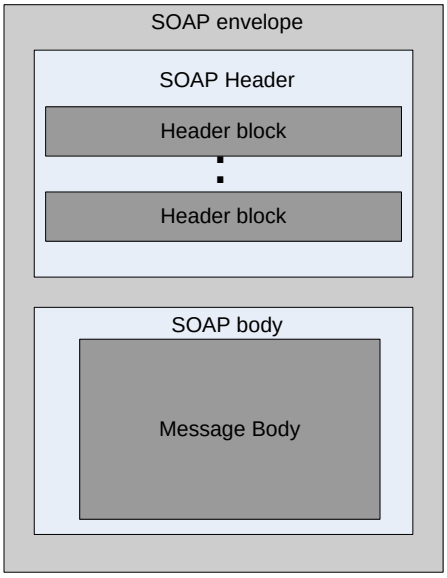


Figure C.5: SOAP message

Figure C.6 illustrates the relationship between XML, SOAP and the transport protocols such as HTTP or JMS. In this example, Application A

sends a SOAP message to Application B. In the application domain the Document Type Definition or an XML schema define the tags and structure of the document. XML is the method of exchanging information between Application A and Application B. The application requires the support of a processor that uses DTD or Schema to extract data from and insert into the XML instance document. The XML instance is encapsulated into a SOAP message between <envelope> and </envelope>. In the request/response type of application, the SOAP message is transported by a HTTP request or a JMS request in the body section. Finally, the message is delivered to the Application B over the network.

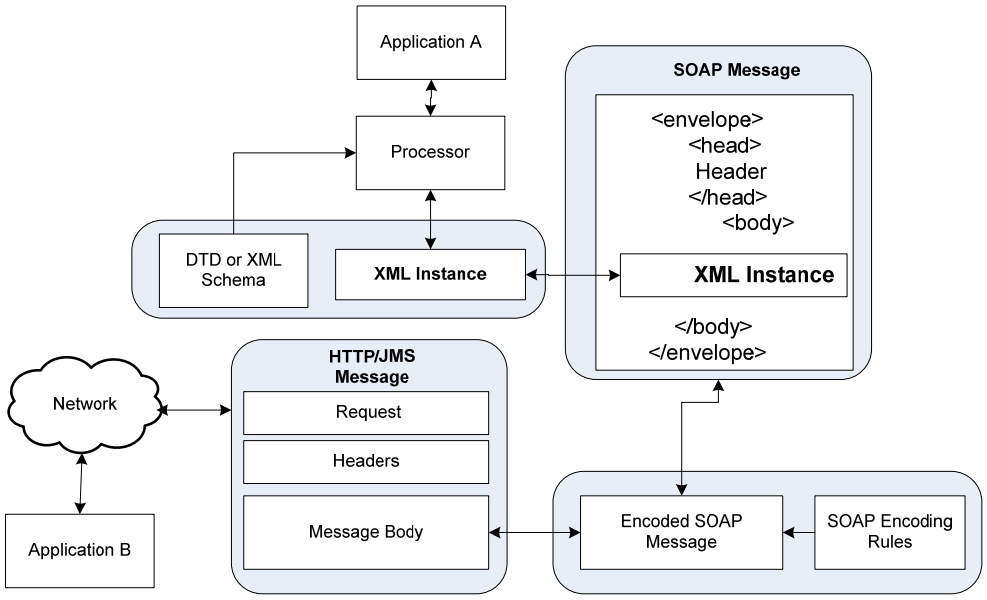


Figure C.6: Relationship between XML, SOAP and transport protocols

SOAP uses two application protocols for transporting SOAP messages, namely, HTTP and JMS. Table C-3 illustrates the modes of transporting SOAP messages.

Table C-3: Modes transporting SOAP messages

	point-to-point	publish/subscribe
Synchronous	HTTP, JMS	JMS
Asynchronous	JMS	JMS

SOAP over HTTP

HTTP is the most widely used protocol for transporting SOAP messages. However, HTTP is limited to synchronous communication pattern that results point-to-point integration techniques. The consequence of using SOAP over HTTP is that the services do not provide any simultaneous notification to multiple recipients. In the NGN environment, a management system may need to notify multiple management systems that a step in a process has been completed. There is a clear need for asynchronous communication in an NGN management implementation. Asyncronicity allows loose-coupling among services that they interact with each other via messages. HTTP with the synchronous nature waits for a response to a request, consuming communication resources until it receives one. Furthermore, HTTP requires both sender and receiver to be connected at the same time in order for the message to be successfully sent. If the network or the receiving service is unavailable, HTTP cannot deliver the message. HTTP offers limited reliability due to the fact that it has limited set of error codes that can be used to identify error conditions [EGGE03]. The protocol cannot guarantee that a message will be delivered to its destination. One solution of improving the HTTP protocol's reliability is to build additional error handling and recovery techniques into the services

themselves. WS-ReliableMessaging standard has been developed by the Organization for the Advancement of Structured Information Standards (OASIS) in order to improve the reliability issues of the SOAP message that uses the HTTP [OASIS07b]. This specification involves coding at the SOAP layer to provide additional error handling techniques. However, these measures can be expensive and may introduce additional complexity to the NGN infrastructure. SOAP messages transmitted over HTTP lack efficient scalability. HTTP imposes a finite limit on the number of socket connections that can coexist at a given time. The connections use significant machine resources and therefore restrict scalability. To solve the scalability issues, additional capacity is achieved by adding other hardware equipment such as Web server or applying load balancing techniques to the resources.

As seen above, adopting HTTP as the transportation protocol for SOAP message can introduce additional complexity. Moreover, the NGN infrastructure will require additional development resources to implement a solution that is based on HTTP. Consequently, the cost of the implementation will increase.

SOAP over JMS

JMS is a specification that provides a standard application program interface for exchanging messages. JMS supports both synchronous and asynchronous communication [EGGE03]. The specification specifies the methods that messages are delivered, security mechanisms, error handling techniques and the underlying protocols between clients and

servers. JMS has been widely adopted as the messaging transport for both application integration and SOA [SWIM], [CHAP04]. JMS supports ‘fire and forget’ communication mode that allows the message to be sent without waiting for reply and placed in a persistent store, or a queue. The queue enables asynchronous communication in the sense that the message producer sends the message to the queue and the consumer acquires the message from the queue and not from the message producer. Furthermore, JMS supports a publish/subscribe model in which a provider can communicate with multiple consumers simultaneously.

JMS is more reliable than the HTTP due to the fact that it uses the concept of queues that ensures message delivery from the sender to the receiver. JMS in the case of guarantee delivery can resend the messages to the destinations. Error recovery and retransmission of the messages are built into the JMS compared to the HTTP and does not require coding into the application or at the SOAP layer. JMS makes more efficient use of system resources allowing scalability by using a single connection between the message producers and the message consumers. This eliminates the scalability issues that HTTP imposes by requiring a separate socket connection for each service request and service reply. Occupying less socket connections can reduce the system’s resources therefore, improving the scalability. Another difference between JMS and HTTP is that JMS separates the destination address from the physical destinations. This independent namespace enables implementations based on JMS messaging to scale systems dynamically. For instance, the producer requires only one destination address to connect with multiple consumers.

Compared to HTTP, JMS provides better message delivery, flexibility, reliability and scalability. These capabilities are implemented inside the scope of JMS specification and do not need to be developed into the services or at the SOAP layer. Thus, using JMS as the transport protocol can provide less complexity to the SOA implementation.

C.2.1.4 WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

WSDL is one of the essential parts of the SOA framework for service description. The service description provides the key ingredient to establishing a consistently loosely coupled form of communication between services implemented as Web Services. For this purpose, description documents are required to accompany any service wanting to act as an ultimate receiver. The primary service description document is the WSDL definition. WSDL is an XML-based format that describes network services as a set of endpoints operating on messages containing either procedure-oriented or document-oriented information. The messages and operations are described abstractly, and they are then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of the message formats or network protocols that are used to communicate. The typical bindings with WSDL are SOAP, HTTP GET/POST, and MIME [W3C01].

The WSDL file consists of six elements. These elements are:

- *definitions*, defines the name of the Web Service, declares multiple namespaces
- *types* describe all the data types used between the server and the client
- *message* describe a one-way message such as request or a response message
- *portType,* combines multiple message elements to form a complete one-way or request-response operation
- *binding* describes how the service will be implemented on the transport layer
- *service*. defines the address for invoking the specified service

C.2.1.5 SOA REGISTRY AND REPOSITORY

Differences between SOA Registry and Repository

In SOA, a registry stores information about services in an SOA. It includes information that other participants can look up to find out the location of the service and what it does. A registry may also include information about policies that are applied to the service, such as security requirements, quality of service commitments and billing.

A repository Stores all services-related artifacts in the enterprise-wide SOA implementation. The repository should also provide cooperation capabilities (the ability to search, modify, etc.) to all the SOA stakeholders. The repository contains all of the design and development artifacts of services that the design tools may need at design and build time. The

service repository is optimized to store large amounts of assets and to enable a large user population to make ad-hoc queries to find these assets. Access to the repository takes place within the enterprise boundaries.

The registry contains a subset of the repository information that is required at runtime binding. The registry often needs to be accessed from within and from the outside of these boundaries. The service registry is optimized for runtime lookups of services endpoint addresses.

Universal Description, Discovery and Integration (UDDI)

UDDI is a registry, where Web Services can be registered and it describes the programming interfaces for publishing, retrieving, and managing information about services. Actually, UDDI itself consists of Web Services. The UDDI specification identifies services that support the description and discovers:

- The Web Services they make available.
- Businesses, organizations, and other Web services providers.
- Technical interfaces that are used to access and manage those services.

UDDI is based on established industry standards, like HTTP, XML, XSD, SOAP and WSDL [OASIS08a].

C.2.1.6 RESTFUL

Another architectural style for implementing SOA is the RESTful Web Service [FIEL00]. It is an alternative solution that implements Remote

Procedure Calls across the Web. Representational State Transfer (REST) is gaining increased attention not only because it is used by many Web 2.0 services, but also because it provides a simple API to implement Web Services. It was originally introduced as an architectural style for building large-scale distributed hypermedia systems. The REST architecture is based on the following four principles [FIEL07]:

- **Resource identification through URI:** The resources of the REST Web Service are identified by URIs (Uniform Resource Identifier) [BERNE05], which provides a global addressing space for resources and service discovery.
- **Uniform interface:** The REST resources are manipulated by using fixed set of operations. These operations are influenced by the CRUD (CREATE, READ, UPDATE, DELETE) operations from the HTTP protocol. The REST operations follow similar patterns with the HTTP operations. These operations are: PUT, GET, POST, and DELETE. PUT operation creates a new resource that can be deleted by using the DELETE operation. GET operation retrieves the current state of a resource and the POST operation transfers a new state onto a resource.
- **Self descriptive messages:** The resources are decoupled from their representation so that their content can be accessed in a variety of formats (e.g. XML, HTML, PDF, etc.).
- **Stateless interactions through hyperlinks:** All RESTful interacts with resources statelessly. Stateless applications can be easier to scale up. In REST stateless means that there is no client session

data stored on the server. The server only records and manages the state of the resources it exposes. If there needs to be session specific data, it should be held and maintained by the client and transferred to the server with each request as needed. A service layer that does not have to maintain client sessions can be easily scaled as it has to do with less replication in a clustered environment.

REST/WS are perceived to be simple and provide a lightweight infrastructure, where services can be built with minimal tooling. This approach allows the developers to work with inexpensive tools and develop platforms that can serve a large number of clients with low cost.

C.3 COMPARING SOAP WEB SERVICES WITH RESTFUL WEB SERVICES

Table C-4 illustrates the differences between RESTful Web Services and SOAP based Web Services [PAUT08].

Table C.4: REST/WS and SOAP/WS comparison

	REST/WS	SOAP/WS
Transport Protocol	HTTP	HTTP, TCP, SMTP, JMS, MQ, IIOP
Payload format	JSON, XML, RSS	XML
Service description	Text, XSD, WSDL	WSDL, XSD
Security	HTTPS	HTTPS, WS-Security, XML security, XML signature
Discovery	Resource, identified by URI	UDDI
Integration styles	URI with standardized interface (put, post, get, delete)	RPC, Messaging
Communication style	asynchronous and Synchronous	Synchronous and asynchronous

Message exchange patterns	Request/response	Request/response, Publish/Subscribe
Architectural focus	Focus on scalability and performance of large scale distributed hypermedia systems	Focus on design of integrated distributed applications
Bandwidth consumption	Low	High
Performance	High	Acceptable
Complexity	Low	High

As can be seen from the table above, SOAP allows messages to be exchanged by using a variety of transport protocols. The WSDL binding element is used to select the appropriate transport protocol to bind the operation messages. REST is using only the HTTP protocol for transporting messages, thus it can only use request/response as a communication pattern compared to SOAP/WS that can use JMS for asynchronous communication. REST/WS is capable of serving resources in multiple representation formats such as JavaScript Object Notation (JSON), XML, and Really Simple Syndication (RSS) [JSON], [RSS]. SOAP/WS can only use XML for representing resources. SOAP/WS provides the UDDI registry for service discovery whereas REST/WS leaves it to the developer to implement service registry. SOAP/WS can be used as a gateway technology to enable interoperability for applications that work both over HTTP and other protocols. Moreover, most of legacy systems are not designed to operate over HTTP protocols, multicast, asynchronous messaging, etc. SOAP/WS can encapsulate their information into transport protocols such as TCP and IIOP and make the integration with other systems possible. Furthermore, SOAP/WS allows the same interface to be bound to different transport protocols as business and technological requirements change.

REST/WS on the other hand is simpler to develop due to the lightweight infrastructure and has been supported by major Web 2.0 applications (Amazon, Google, etc.). REST/WS has better performance compared to SOAP/WS due to the absence of intermediaries, message wrapping, and serialization that are required by the SOAP/WS. Due to the fact that REST/WS is lightweight and the messages that exchanged are less verbose than SOAP messages, it could be used by portable devices that have limited bandwidth and processing power. The major drawback of the REST/WS is that it cannot deliver enterprise-wide capabilities such as message verification, message validation, message transaction etc. that are required by enterprise systems. SOAP/WS provides better support for security, reliable messaging and transaction management [MacV06] that are vital functions for the back end systems in enterprises. REST/WS are mostly used for front-end interactions between applications and consumers. Thus, SOAP/WS are more commonly used for the back-end systems that require more sophisticated functions that REST/WS cannot deliver.

Appendix D : Enterprise Service Bus (ESB)

D.1 THE ESB IN THE SOA CONTEXT

For the NGN convergence, system integration patterns and strategies are vital for a long term lasting integration framework. There are two significant options for system integration: The direct point-to-point integration and the Bus integration. In the first approach, each connection between applications is individually designed and cooperatively implemented, deployed, and administered. The responsibility for the connectivity issues such as location, naming and security of services is distributed among the applications. In the Bus approach, the interaction among services is mediated by a brokering component that is used as messaging backbone for message propagation. In the SOA context, this component is referred to as ESB. Each application is designed to interact with the ESB, allowing it to manage routing and transformation of the messages exchanged between applications. Figure D.1 shows those two different integration approaches.

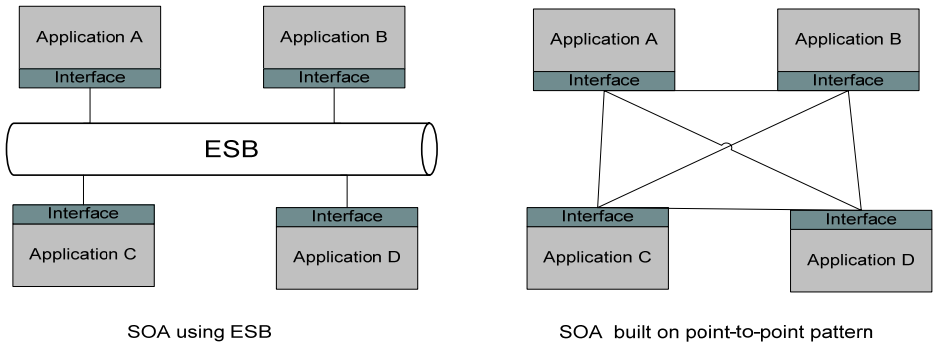


Figure D.1: Comparison of ESB and point-to-point integration

ESB is an emerging middleware that provides technological solutions to intercept messages among services. It provides the fundamental support for Web Services. ESB incorporates the concept of mediation and solving the problem of interoperability between clients and data sources in Information Systems [JOSU07]. An ESB is actually a middleware providing integration facilities built on top of industrial standards such as XML, SOAP, WSDL, WS-Addressing, and WS-Security. ESB provides a communication channel mostly asynchronous (Publish/Subscribe), a trading service in order to find appropriate services and an orchestration service [CHAP04]. In addition to transformation functionalities, ESB provides dynamic routing and dispatch of requests to multiple receivers, which is an important functionality when using heterogeneous systems and other QoS management functions such as quality measurement, tracing, data management, caching or failure detection and recovery. Moreover, the ESB functionality can be distributed across multiple servers, which are centrally managed. Other middleware solutions such as ORB cannot distribute their functionality. ESB provides support for use of proprietary or custom adapters to connect to legacy and COTS systems.

Implementing ESB in an SOA framework increases the interoperability among applications due to the fact that ESB allows connected applications with disparate technologies and data formats to interoperate as service users and service providers without changing their internal functions. Moreover, ESB improves the modifiability of the framework by allowing many types of changes or replacements of service providers without

impacting the service users. ESB provides extensibility by allowing service to be connected with each other easily via standardized and open interfaces. Thus enterprises can have fast changes according to the business needs.

On the other hand, there are some issues that need to be considered when designing an SOA framework that is based on the ESB. First, the performance may be negatively impacted due to additional message hops and message transformations performed by the ESB. To solve the performance issues, ESB functionality can be distributed and implemented in separate servers. For instance, transformation functions, routing functions and validation functions can be implemented into different servers forming a cluster that acts as one service. Hence, the performance can be improved. Another issue that arises is that adopting ESB may not be feasible in environments with a small number of applications and services. ESB should be implemented in environments where there are many heterogeneous services and applications. The purpose of NGN is to use different transport technologies in order to provide a unified access to the service users. As a result, the NGN management plane is a complex heterogeneous environment that requires the many different services and applications to be interconnected.

Figure D.2 illustrates the ESB as a Reliable asynchronous Secure Messaging pipe and the connected services that provide different functionalities.

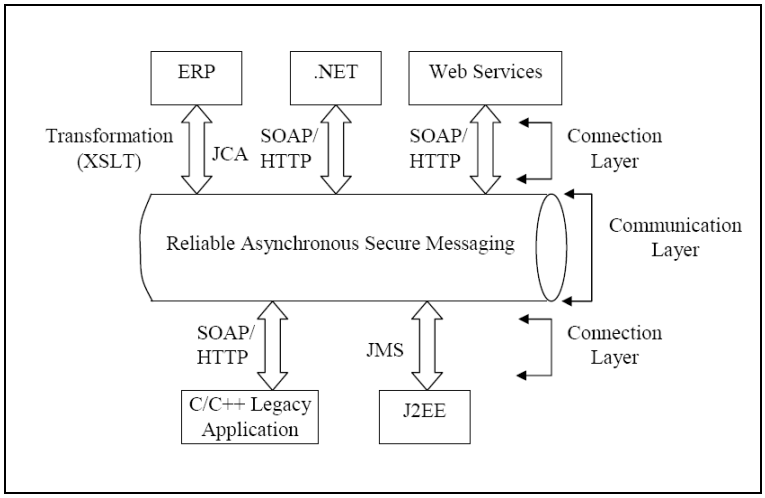


Figure D.2: Enterprise Service Bus (ESB)

ESB offers the following key features:

- Support of SOAP, WSDL and UDDI, as well as emerging standards such as WS-Reliable messaging and WS-Security.
- Messaging: asynchronous store-and-forward delivery with multiple qualities of service.
- Content-based routing.
- Data transformation.
- Platform-neutral: connects to any technology. For example, Java, .Net, databases and mainframes.

D.2 COMPARING CORBA WITH ESB

Even though CORBA technology has been adopted by real-time mission critical environments such as air traffic control and military embedded systems, its adoption is declining over the last years [ABEE06]. The telecommunication industry as stated in previous sections is shifting towards the SOA through the use of Web Services. The combination of

Web Services with ESB technology can provide solutions to the complex heterogeneous environments that require today.

Today, CORBA is used mostly to 'wire' together components that run inside the companies' networks, where communication is protected from the outside world by firewalls. From an architectural point of view, NGN is specifying the decoupling of the network from the service functionalities. NGN tries to make services independent from the underlying technologies where the enterprises are required to 'open up' their boundaries and operate in an open B2B environment [HENN06]. The open B2B transactions among enterprises need to conform to open and standardized interfaces that are loosely coupled in order to minimize the dependencies between the communication parties. NGN management should facilitate this decoupling and should offer operational services taking into account the layers defined by the NGN.

Table D-1 presents the differences between CORBA middleware and ESB.

Table D-1: ESB and CORBA characteristics

	ESB	**CORBA**
Communication Infrastructure	SOAP as messaging payload	Binary message payload over IIOP
Interface definitions	WSDL	IDL
Messaging styles	• **One-way:** (SOAP over HTTP or SOAP over JMS) • **Request-response:** (SOAP over HTTP or SOAP over JMS) • **Document-oriented:** (SOAP over HTTP or SOAP over JMS) • **Publish-Subscribe:** (SOAP over JMS)	• **One-way:** (IIOP) • **Request-response:** (IIOP) • **Document-oriented:** (IIOP) • **Publish-Subscribe:** (using event notifications)
Data Validation	Custom programming routines	SOAP message payload

	perform validation.	can be validated using XML schemas.
Complexity	Easy when using specialized tools such as WSDL converter.	Complex server-side programming model
Performance	Acceptable performance.	CORBA systems can offer greater performance
Technology adoption	Adopted by many industry leading companies such as Microsoft, IBM Technology support: J2EE .Net	CORBA future is uncertain. If CORBA fails to achieve sufficient adoption by the industry, then CORBA implementations become legacy systems.

ESB typically uses SOAP as a messaging payload where messages are self-describing due to the fact that messages are based on XML. The payload of the SOAP message is transmitted over HTTP or JMS protocols. CORBA uses a binary message payload where messages are not self-described and the payload is transmitted over the IIOP protocol. Both CORBA and ESB support the same messaging styles but are using different protocols to achieve it. CORBA has more complex APIs compared to Web Services. CORBA APIs are far larger than necessary. For instance, the CORBA's object adapter requires more than 200 lines of interface definition code, even though the same functionality can be provided in about 30 lines [HENN06].

Another problem is that the language mappings in CORBA are difficult to implement due to the complex and poorly designed API [HENN06]. On the other hand, CORBA-based systems can achieve better performance compared to ESB because they use remote objects and method invocations for the communication resulting in lower overheads. ESB has enormous adoption by many enterprises. According to a study conducted by the AberdeenGroup, involving 120 organizations and their adoption of

ESB and SOA technologies concluded that only 7% of large companies have no plans of using ESB in their SOA infrastructure. The majority of the medium size and the 80% of small companies have not used ESB yet due to the fact that they are still in the designing phase of their architecture [ABEE06]. On the other hand, CORBA fails to achieve sufficient adoption by the industry; only 29% of the involved organizations were considering CORBA as an alternative technology for implementing SOA.

Appendix E : IMPLEMENTATION CODE

E.1 CORE NMS SERVICE BUS ROUTING RULES

The following code is a part of the Routing Service. It contains the rules to route the management messages to the specified topic or queue according to XPath rules.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans                          xmlns:rule="http://servicemix.apache.org/eip/1.0"
xmlns:esb="http://esb.com/localhost">

<rule:xpath-splitter service="esb:RRouter" endpoint="RRouterEndpoint"
 xpath="/*/*" namespaceContext="#nsContext">
  <rule:target>
  <rule:exchange-target service="esb:AppInput" />
  </rule:target>
</rule:xpath-splitter>
<rule:content-enricher                       service="esb:ContentEnrichingFunction"
endpoint="EnrichingEndpoint">
  <rule:enricherTarget>
    <rule:exchange-target service="esb:DecisionPoint" />
  </rule:enricherTarget>
  <rule:target>
    <rule:exchange-target service="esb:TransformationService" />
  </rule:target>
</rule:content-enricher>
<rule:content-based-router service="esb:AppInput"
endpoint="AppInputEndpoint">
<rule:rules>
       <rule:routing-rule>
              <rule:predicate>
              <rule:xpath-predicate xpath="//NMS1:severity='1'| //NMS1:NMS='1'"
              namespaceContext="#nsContext" ></rule:xpath-predicate>
              </rule:predicate>
              <rule:target>
              <rule:exchange-target       service="esb:wireTap1"></rule:exchange-
target>
              </rule:target>
              </rule:routing-rule>
              <rule:routing-rule>
              <rule:predicate>
              <rule:xpath-predicate                xpath="//NMS2:severity='High'|
//NMS2:NMS='2'"
              namespaceContext="#nsContext" ></rule:xpath-predicate>
             </rule:predicate>
              <rule:target>
              <rule:exchange-target       service="esb:wireTap1"></rule:exchange-
target>
              </rule:target>
              </rule:routing-rule>
              <rule:routing-rule>
              <rule:predicate>
              <rule:xpath-predicate xpath="//NMS1:severity='2'| //NMS1:NMS='1'"
              namespaceContext="#nsContext"></rule:xpath-predicate>
              </rule:predicate>
              <rule:target>
              <rule:exchange-target       service="esb:wireTap2"></rule:exchange-
target>
              </rule:target>
              </rule:routing-rule>
```

```
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//NMS2:severity='Low'| //NMS2:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target service="esb:wireTap3></rule:exchange-target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//NMS1:severity='3'| //NMS1:NMS='1'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:wireTap4"></rule:exchange-
target>
       </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate         xpath="//NMS2:severity='information'|
//NMS2:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:wireTap5"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS1:NMS='1'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS1queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS1:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS2queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS1:NMS='3'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS3queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS1:NMS='4'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS4queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS2:NMS='1'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
```

```
                <rule:target>
                <rule:exchange-target       service="esb:MS1queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS2:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS2queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS2:NMS='3'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS3queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS2:NMS='4'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS4queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS3:NMS='1'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS1queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS3:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS2queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS3:NMS='3'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS3queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS3:NMS='4'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS4queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
```

```
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS4:NMS='1'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS1queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS4:NMS='2'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS2queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS4:NMS='3'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS3queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:predicate>
                <rule:xpath-predicate xpath="//MS4:NMS='4'"
                namespaceContext="#nsContext" ></rule:xpath-predicate>
                </rule:predicate>
                <rule:target>
                <rule:exchange-target       service="esb:MS4queue"></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                <rule:routing-rule>
                <rule:target>
                <rule:exchange-target   service="esb:SoftAppIn"   ></rule:exchange-
target>
                </rule:target>
                </rule:routing-rule>
                </rule:rules>
        </rule:content-based-router>
<rule:wire-tap service="esb:wireTap1" endpoint="wireTapendpoint1">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:wireTap6" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap6" endpoint="wireTapendpoint6">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder1" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap2" endpoint="wireTapendpoint2">
  <rule:target>
    <rule:exchange-target service="esb:wireTap7" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:wireTap8" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap7" endpoint="wireTapendpoint7">
  <rule:target>
    <rule:exchange-target service="esb:Topic2" />
```

```
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder2" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap8" endpoint="wireTapendpoint8">
  <rule:target>
    <rule:exchange-target service="esb:Topic3" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder3" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap3" endpoint="wireTapendpoint3">
  <rule:target>
    <rule:exchange-target service="esb:wireTap9" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:wireTap10" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap9" endpoint="wireTapendpoint9">
  <rule:target>
    <rule:exchange-target service="esb:Topic2" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder2" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap10" endpoint="wireTapendpoint10">
  <rule:target>
    <rule:exchange-target service="esb:Topic4" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder4" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap4" endpoint="wireTapendpoint4">
  <rule:target>
    <rule:exchange-target service="esb:wireTap11" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:wireTap12" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap11" endpoint="wireTapendpoint11">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder1" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap12" endpoint="wireTapendpoint12">
  <rule:target>
    <rule:exchange-target service="esb:Topic3" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder3" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap5" endpoint="wireTapendpoint5">
  <rule:target>
    <rule:exchange-target service="esb:wireTap13" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:wireTap14" />
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap13" endpoint="wireTapendpoint13">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder1" />
```

```
  </rule:inListener>
</rule:wire-tap>
<rule:wire-tap service="esb:wireTap14" endpoint="wireTapendpoint14">
  <rule:target>
    <rule:exchange-target service="esb:Topic1" />
  </rule:target>
  <rule:inListener>
    <rule:exchange-target service="esb:Folder4" />
  </rule:inListener>
</rule:wire-tap>
       <rule:namespace-context id="nsContext">
              <rule:namespaces>
<rule:namespace prefix="NMS1">http://esb.nms1.com</rule:namespace>
<rule:namespace prefix="NMS2">http://esb.nms2.com</rule:namespace>
<rule:namespace prefix="MS1">http://esb.ms1.com</rule:namespace>
<rule:namespace prefix="MS2">http://esb.ms2.com</rule:namespace>
<rule:namespace prefix="MS3">http://esb.ms3.com</rule:namespace>
<rule:namespace prefix="MS4">http://esb.ms4.com</rule:namespace>
              </rule:namespaces>
       </rule:namespace-context>
</beans>
```

E.2 FILE ARCHIVE SERVICE

The following code is a part of the Archive service that creates folders as well as the connection points that the routing rules are specified to sent.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:sm="http://servicemix.apache.org/config/1.0"
       xmlns:file="http://servicemix.apache.org/file/1.0"
xmlns:esb="http://esb.com/localhost">

       <file:poller service="esb:routingPoller" endpoint="routingEndpoint"
              targetService="esb:JMSSender"             targetEndpoint="Endpoint"
       file="file:Routing_Inbox"></file:poller>

       <file:poller service="esb:AppIn" endpoint="AppEndpoint"
              targetService="esb:AppInput"      targetEndpoint="AppInputEndpoint"
       file="file:App_Input"></file:poller>

       <file:sender service="esb:Folder1" endpoint="folder1Endpoint"
              directory="file:NMS1_F"></file:sender>

       <file:sender service="esb:Folder2" endpoint="folder2Endpoint"
              directory="file:NMS2_F"></file:sender>

       <file:sender service="esb:Folder3" endpoint="folder3Endpoint"
              directory="file:MService1_F"></file:sender>

       <file:sender service="esb:Folder4" endpoint="folder4Endpoint"
              directory="file:MService2_F"></file:sender>

       <file:sender service="esb:Folder5" endpoint="folder5Endpoint"
              directory="file:MService3_F"></file:sender>

       <file:sender service="esb:Folder6" endpoint="folder6Endpoint"
              directory="file:MService4_F"></file:sender>

       <file:sender service="esb:Folder7" endpoint="folder7Endpoint"
              directory="file:Topic1_F"></file:sender>

       <file:sender service="esb:Folder8" endpoint="folder8Endpoint"
              directory="file:Topic2_F"></file:sender>

       <file:sender service="esb:Folder9" endpoint="folder9Endpoint"
              directory="file:Topic3_F"></file:sender>
```

```
      <file:sender service="esb:Folder10" endpoint="folder10Endpoint"
               directory="file:Topic4_F"></file:sender>

        <file:sender service="esb:SoftAppIn" endpoint="SoftEndpoint"
               directory="file:Type_Soft"></file:sender>
</beans>
```

E.3 CREATING MESSAGE QUEUES AND TOPICS

The following code is creating four JMS queues and four JMS Topics.

```
<beans xmlns:sm="http://servicemix.apache.org/config/1.0"
       xmlns:jms="http://servicemix.apache.org/jms/1.0"
xmlns:esb="http://esb.com/localhost">
<jms:provider service="esb:JMSSender" endpoint="Endpoint"
               destinationName="MS" connectionFactory="#connectionFactory" />
<jms:consumer service="esb:JMSConsumerService" endpoint="inQueueReader"
               targetService="esb:RRouter" targetEndpoint="RRouterEndpoint"
               destinationName="MS" connectionFactory="#connectionFactory" />
<jms:provider service="esb:Topic1" endpoint="Topic1Endpoint"
               destinationName="Topic1" replyDestinationName="Topic1"
               connectionFactory="#connectionFactory"
               pubSubDomain="true" />
<jms:provider service="esb:Topic2" endpoint="Topic2Endpoint"
               destinationName="Topic2" replyDestinationName="Topic2"
               connectionFactory="#connectionFactory"
               pubSubDomain="true" />
<jmsprovider service="esb:Topic3" endpoint="Topic33Endpoint"
               destinationName="Topic3" replyDestinationName="Topic3"
               connectionFactory="#connectionFactory"
               pubSubDomain="true" />
<jms:provider service="esb:Topic4" endpoint="Topic4Endpoint"
               destinationName="Topic4" replyDestinationName="Topic4"
               connectionFactory="#connectionFactory"
               pubSubDomain="true" />
<jms:provider service="esb:MS1queue" endpoint="MS1Endpoint"
               destinationName="MS1" replyDestinationName="MS1"
               connectionFactory="#connectionFactory" />
<jms:provider service="esb:MS2queue" endpoint="MS2Endpoint"
               destinationName="MS2" replyDestinationName="MS2"
               connectionFactory="#connectionFactory" />
<jms:provider service="esb:MS3queue" endpoint="MS3Endpoint"
               destinationName="MS3" replyDestinationName="MS3"
               connectionFactory="#connectionFactory" />
<jms:provider service="esb:MS4queue" endpoint="MS4Endpoint"
               destinationName="MS1" replyDestinationName="MS4"
               connectionFactory="#connectionFactory" />
       <bean                                                id="connectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">
               <property name="brokerURL" value="tcp://127.0.0.1:61616" />
       </bean>
</beans>
```

E.4 TROUBLE TICKETING WSDL FILE

The following code illustrates the service contract that has to be used from external services in order to invoke the Trouble Ticketing System.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost.com.ws"
```

```
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost/wsdl/TTWebService.wsdl"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://Trouble.Ticketing.dto"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://Trouble.Ticketing.ws">
    <wsdl:types>
        <schema targetNamespace="http://localhost.com"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
            <complexType name="ticket">
                <sequence>
                    <element name="details" nillable="true" type="xsd:string" />
                    <element name="e-mail" nillable="true" type="xsd:string" />
                                        <element name="ID" type="xsd:long" />
                    <element name="SubmitDate" nillable="true"
                                        type="xsd:dateTime" />
                    <element       name="event       summary"       nillable="true"
type="xsd:string" />
                                <element       name="Location"       nillable="true"
type="xsd:string" />
                </sequence>
            </complexType>
        </schema>
    </wsdl:types>
    <wsdl:message name="getRequest">
        <wsdl:part name="status" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="getResponse">
        <wsdl:part name="getReturn" type="tns1:ArrayOf_tns_ticket" />
    </wsdl:message>
      <wsdl:portType name="TTWebService">
        <wsdl:operation name="getTTs" parameterOrder="status">
            <wsdl:input message="impl:getRequest"
                                name="getRequest" />
            <wsdl:output message="impl:getResponse"
                                  name="getResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="TTWebServiceSoapBinding"
        type="impl:TTWebService">
        <wsdlsoap:binding style="rpc"
                           transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="get">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="getRequest">
                <wsdlsoap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://localhost.com" use="encoded" />
            </wsdl:input>
            <wsdl:output name="getResponse">
                <wsdlsoap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://localhost.com" use="encoded" />
            </wsdl:output>
        </wsdl:operation>
          </wsdl:binding>
    <wsdl:service name="TTWebServiceService">
        <wsdl:port binding="impl:TTtWebServiceSoapBinding"
                         name="TTWebService">
           <wsdlsoap:address
            location="http://localhost:8080/WebService/TTWebService" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```