Project Number: IST-1999-11387

Project Title: Layers Interworking in Optical Networks (LION)

Deliverable Type: (P/R/L/I)* R

CEC Deliverable Number: IST-1999-11387/CSELT/LION_D9

Contractual Date of Delivery to the CEC: Month 11

Actual Date of Delivery to the CEC: 22/12/00

Title of Deliverable: Preliminary Architecture and Design of the Umbrella Management System

Workpackage contributing to the Deliverable: WP5

Nature of the Deliverable: (P/R/S/T/O)** R

Author(s): Ralf Geerdsen, Ulrike Hartmer, Giuseppe Ricucci

Abstract:

This deliverable describes the preliminary architectural design of the LION umbrella network management system to be implemented for the LION field trial. The management system is designed for a multi-vendor, multi-technology environment. Furthermore, its architecture supports the co-operation of two management systems via a network level management interface. Therefore, the requirements for the design reflect the context of the LION field trial, as well as typical situations in today's telecommunication networks.

The document gives a description of the architectural concepts used and the existing frameworks adopted. Furthermore, their impact on the network level interface, the decisions made, the logical architecture and the software architecture are discussed.

The LION network management system consists of two (CSELT and T-Nova) sub-NMSs which interoperate via a CORBA interface. This network level interface connects mutually the network element level and the network level operating systems, and for its definition the coarse grained CORBA approach promoted by ETSI has been chosen.

For the software architecture design of the two NMSs, it had to be considered that the sub-NMS are based on different management technologies (WBEM for CSELT, CORBA for T-Nova). To obtain interoperability, a DCOM/CORBA bridge has been included on the CSELT side. The architecture also accounts for the fact that both NMSs have to deal with network elements managed by different management technologies (CMIP, SNMP, and WBEM). The preliminary architecture will be revised when the management requirements and first implementation results are available.

Keyword list: network management system, CORBA, WBEM, DCOM, network level interface, network level information model, network level operating system, network element level operating system

* Type: P-public, R-restricted, L-limited, I-internal

** Nature: P-prototype, R-report, S-specification, T-tool, O-other

Preliminary Architecture and Design of the Umbrella Management System

Status and Version:	Version 5	
Date of issue:	20. December 2000	
Distribution:	LION, Deliverable Type: Restricted	
Author(s):	Ulrike Hartmer	T-Nova
	Ralf Geerdsen	T-Nova
	Giuseppe Ricucci	CSELT
Checked by:	Lampros Raptis	NTUA
	Georg Lehr	T-Nova

Table of Contents

1 Introduction 4

1.1 Purpose and Scope 4

1.2 Reference Material 4
1.2.1 Reference Documents 4
1.2.2 Abbreviations 5
1.2.3 Definitions 7

1.3 Document History 9

1.4 Document overview 9

2 The LION Network Management System: Overview 10

3 Architectural Concepts used for the LION Umbrella Management System 11

3.1 WMI/DCOM Architecture 11

3.2 Common Object Request Broker Architecture 15
3.2.1 Introduction on CORBA 15
3.2.2 CORBA for Telecommunications Management 17
3.2.3 Existing TMN-CORBA Frameworks 20

3.3 DCOM / CORBA Mapping 23

4 Architecture of the LION Umbrella Management System 27

4.1 Overall Architecture 27

4.2 CORBA based Network Layer Interface 30
4.2.1 Adopted CORBA Approach and Framework 30
4.2.2 CORBA Object Services Used in LION 34

5 Architecture of the LION Subnetwork Management Systems 37

5.1 Corba Standard Version and Products Used 37

5.2 CSELT NMS 38
5.2.1 Architecture 38
5.2.2 Functionality 42

5.3 T-NOVA NMS 42
5.3.1 Architecture 42
5.3.2 Functionality 43

1 Introduction

1.1 Purpose and Scope

The main goal of LION is to design and test a resilient and managed optical transport network carrying different client layer networks (such as IP), with interworking between client-server layers and between domains (transparent and administrative domains). To test the identified requirements a field trial will be set up, consisting of two administrative domains and their network management systems. For an integrated network view, an umbrella management system will be implemented.

This deliverable describes the logical architecture of the umbrella management system as well as the software architecture of the two sub-network management systems. This comprises a description of the architectural concepts used and the existing frameworks adopted, their impact on the network level interface, the decisions made and their influences on the network level interface, the logical architecture and the software architecture. This preliminary architecture will be revised according to the management requirements to be defined in WP5 and according to first implementation results. The final architecture will be described in deliverable 12, due in May 2001.

This deliverable is based on the work done in WP5 and documented in versions 1 to 7 of CD9_WP5_T-Nova_02 and on the discussion in CD9_WP5_NTUA_01_v5.doc.

1.2 Reference Material

1.2.1 Reference Documents

[1] ITU-T; Working Draft of CORBA Framework for TMN; SG4 Q19; Editor K.Allan; 08/1999 Revised and divided into two documents [7] and [8] for editorial reasons.

[2] ETSI WG TMN OMI; LIASON Statement: CORBA based management framework; Contact H.Ploeger; Antwerpen 30 Nov.-02 Dec. 1999

[3] OMG; The Common Object Request Broker: Architecture and Specification, V. 2.1; Sept. 1997

[4] OMG; CORBAservices: Common Object Services Specification; Dec. 1998

[5] OMG; Notification Service Specification; V. 1.0; June 2000

[6] OMG; Interworking Between CORBA and TMN Systems Specification; V. 1.0; August 2000

[7] ITU-T; DRAFT NEW RECOMMENDATION Q.816: CORBA Based TMN Services; August 2000

[8] ITU-T; Draft Rec. X.780 “TMN Guidelines for Defining CORBA Managed Objects”; August 2000

[9] Distributed Component Object Model Protocol - DCOM 1/0 (MSDN Library, Specifications)

[10] COM Technologies Home Page at http://www.microsoft.com/com/comintro.htm

[11] “COM versus CORBA: A Decision Framework”; Owen Tallman, J. Bradford Kain, 12/1998, http://www.quoininc.com/quoininc/COM_CORBA.html

[12] "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer” http://www.cs.wustl.edu/~schmidt/submit/Paper.html

[13] LION document CD9_WP5_T-Nova_02_v7.doc

[14] LION document CD9_WP5_NTUA_01_v5.doc

[15] EN 300 820-1, FINAL DRAFT "Asynchronous Transfer Mode (ATM); Management information model for the X-type interface between Operation Systems (OSs) of a Virtual Path (VP) / Virtual Channel (VC) cross connected network; Part 1: Configuration management aspects", September 1997.

1.2.2 Abbreviations

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CIM	Common Information Model
CIMOM	CIM Object Manager
CLI	(Cisco) Command Language Interpreter; Command Line Interface
CMIP	Common Management Information Protocol
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSELT	Centro Studi E Laboratori Telecomunicazioni
DCOM	Distributed COM
DMTF	Distributed Management Task Force
ETSI	European Telecommunications Standard Institute
GDMO	Giudelines for the Definition of Managed Objects
GIOP	General Inter-ORB Protocol
GSR	Giga Switch Router
GUI	Graphical User Interface
IDL	Interface Definition Language (of OMG)
IFR	Interface Repository
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IOS	Internetwork Operating System
ITU-T	International Telecommunication Union - Telecommunication standardisation sector
JIDM	Joint Inter-Domain Management
LAN	Local Area Network
MB	Management Broker
MIB	Management Information Base
MIDL	Microsoft Interface Definition Language
NE	Network Element
NEL-OS	Network Element Level Operating System
NL	Network Level

NL-IF	Network Level Interface
NL-IM	Network Level Information Model
NL-OS	Network Level Operating System
NMS	Network Management System
NNI	Network Network Interface (also: Network Node Interface)
MO	Managed Object
OADM	Optical Add-Drop Multiplexer
OLA	Optical Line Amplifier
OMG	Object Management Group
ONE	Optical Network Element
ORB	Object Request Broker
ORPC	Object Remote Procedure Call
OS	Operating System
OSI	Open Systems Interconnection
OSS	Operating System Support
OTN	Optical Transport Network
OXC	Optical Cross Connect
PDU	Protocol Data Unit
SDH	Synchronous Digital Hierarchy
SNMP	Simple Network Management Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TMN	Telecommunications Management Network
UML	Unified Modelling Language
UNI	User Network Interface
WAN	Wide Area Network
WBEM	Web-Based Enterprise Management
WDM	Windows Driver Model
WMI	Windows Management Instrumentation

1.2.3 Definitions

1.2.3.1 DCOM Definitions

Aggregation relationship	A relationship in which one entity is made up of the aggregation of some number of other entities.
Association class	A class that describes a relationship between two classes or between instances of two classes. The properties of an association class include pointers, or references, to the two classes or instances.
CIMOM repository	A central storage area managed by the CIMOM. This repository contains the definitions of classes and instances that represent managed objects and the relationships among them.
CIM	The common information model is a common data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment. It is comprised of a Specification and a Schema.
CIM specification	Defines the details for integration with other management models (i.e. SNMP's MIB)
CIM schema	A collection of class definitions used to represent managed objects that occur in every management environment.
CIMOM	The CIM object manager is a component in the CIM management infrastructure that handles the interaction between management applications and providers. It supports services such as event notification, remote access and query processing. It also grants access to the CIMOM repository.
COM	The component object model is an open architecture for cross-platform development of client/server applications based on object-oriented technology.
DCOM	The distributed COM extends the COM to support communication among objects on different computers – on a LAN, a WAN, or even the Internet. With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application.
Dynamic class	The CIM class whose definition is supplied by a provider at runtime as needed. Dynamic classes are used to represent provider-specific managed objects and are not stored permanently in the CIMOM repository. Instead, the provider responsible for a dynamic class stores information about its location. When an application requests a dynamic class, the CIMOM locates the provider and forwards the request. Dynamic classes support only dynamic instances.
Instance	A representation of a real-world managed object that belongs to a particular class, or a particular occurrence of an event. Instances contain actual data.
Microsoft IDL (MIDL)	A generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

Managed Object Format (MOF)	A compiled language for defining classes and instances. A MOF compiler offers a textual means of adding data to the CIMOM repository.
MOF file	A text file that contains definitions of classes and instances using the MOF language.
Namespace	A unit for grouping classes and instances to control their scope and visibility. Namespaces are not physical locations; they are more like logical databases containing specific classes and instances.
Object path	A formatted string used to access namespaces, classes and instances. Each object on the system has a unique path which identifies it locally or over the network. Object paths are conceptually similar to Universal Resource Locators (URL).
Provider	A COM server that communicates with managed objects to access data and event notifications from a variety of sources, such as the system registry or an SNMP device. Providers forward this information to the CIMOM for integration and interpretation.
Static class	The CIM class whose definition is persistent. The definition is stored in the CIMOM repository until it is explicitly deleted. The CIMOM can provide definitions of static classes without the help of a provider. Static classes can support either static or dynamic instances.
Static instance	An instance that is persistently stored in the CIMOM repository and remains valid until explicitly deleted, even across system reboots.

1.2.3.2 CORBA Definitions

server	A server is an entity, which is able to answer a client request with a suitable reply. The problem is the different understanding of the term "server".
(CORBA) server object	A server object (implementation) is an instance which is able to react and answer on operation calls defined in the related IDL interface. The operation request is called by a client object (implementation).
(CORBA) server application	A server application holds one ore more server objects (implementations).
fine grained CORBA model	In the fine grained CORBA model exists a one-to-one relationship between CORBA interface instances (with their unique IOR) and managed object instances.
instance grained CORBA model	Corresponds to fine grained CORBA model
class grained CORBA model	The class grained CORBA model contains one CORBA interface for each managed object class. The managed object instance is identified by its unique name.
grain neutral CORBA model	The grain neutral CORBA model uses a structure holding both the managed object name and the IOR to provide access to each managed object instance.
coarse grained CORBA model	This approach defines a set of "first level objects" which allow access to the managed object instances via an internal API. This API can be defined in any (useful) language and represents the "information model".

1.3 Document History

Version	Date	Authors	Comment
1	28.11.2000	Hartmer, Geerdsen, Ricucci	Initial document, derived from CD9_WP5_T-Nova_02_V7.doc
2	05.12.2000	Ricucci	DCOM/CORBA chapter added, minor changes was done in parallel to version 2.1
2.1	01.12.2000	Hartmer, Geerdsen	introductory chapters added was done in parallel to version 2
3	06.12.2000	Hartmer	versions 2 and 2.1 merged
3.1	07.12.2000	Geerdsen	CORBA chapter 3.2.1 revised, not distributed
4	14.12.2000	Hartmer, Geerdsen	comments incorporated, some explanatory sentences added, chapter 3.2.1 revised
5	20.12.2000	Hartmer	comments incorporated

1.4 Document overview

This document is structured as follows:

- Section 2 gives a general overview of the LION testbed configuration and architecture of the LION network management system. The concepts and protocols used are listed and the main functionality of the LION NMS is described.
- Section 3 describes the architectural concepts used for the LION network management system, i.e. CORBA and WBEM.
- On that basis, section 4 reports the logical design of the umbrella management system architecture, as well as some existing CORBA/TMN frameworks to be considered for the network level interface.
- Finally, the software architecture, implementing the logical architecture and support for the different protocols used, is described in section 5.

2 The LION Network Management System: Overview

The main goal of LION is to design and test a resilient and managed optical transport network carrying different client layer networks (such as IP), with interworking between client-server layers and between domains (transparent and administrative domains). To test the identified requirements a field trial will be set up, consisting of two administrative domains, each with OTN and IP network resources and a network management system managing them. For the overall network view an umbrella management system will be implemented that integrates the sub-management systems of the two domains. Figure 1 shows the current design of the LION testbed.[1]

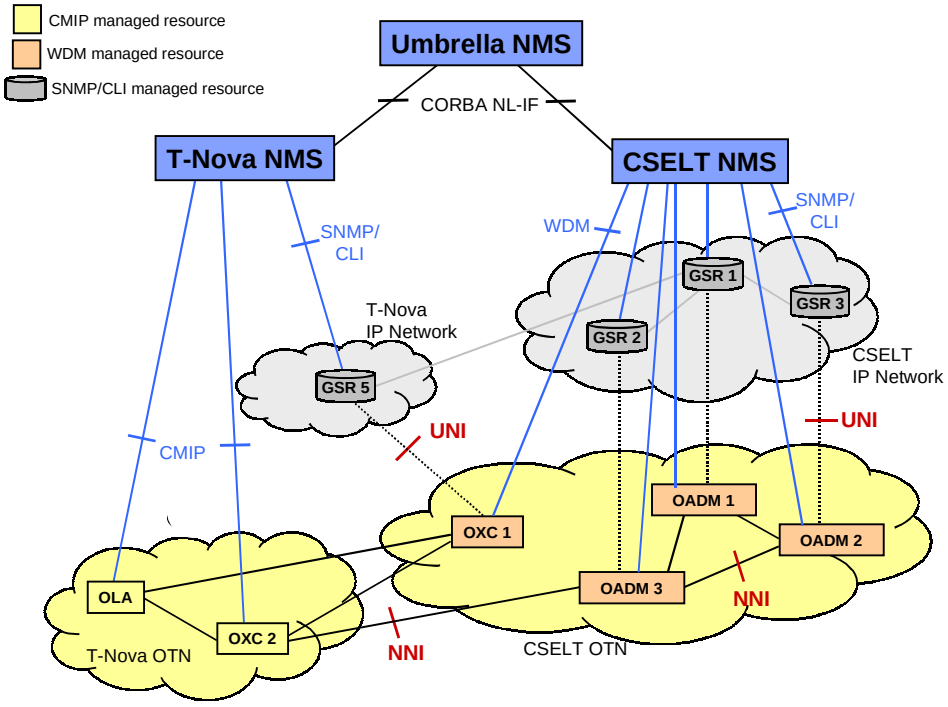

Figure 1: LION testbed configuration (see footnote 1)

The umbrella NMS and the sub-NMSs interoperate via a CORBA network level interface (in the following the term NMS and sub-NMS will be used interchangeably). The CSELT NMS is based on WBEM, i.e. it is built using the DCOM architecture. Thus, a DCOM/CORBA bridge is needed on CSELT side to be able to communicate (via the CORBA interface) with the CORBA based T-Nova NMS.

Both sub-NMSs are used for managing optical network elements (ONEs) as well as IP network elements. The ONEs on T-Nova side are managed using CMIP, the ONEs on CSELT side are managed using WDM (Windows Driver Model). The Gigaswitch Routers (GSRs) are managed by SNMP and, as SNMP does not allow full access to the GSRs, the Cisco command language interpreter is additionally used (CLI, may also stand for (and is a) command line interface). For a more detailed description of the logical architecture please refer to section 4.1, for a description of its implementation see chapter 5.

[1] For readability purposes one detail is not depicted in Figure 1: in fact there will be two umbrella NMSs, one on T-Nova side and one on CSELT side. This will considerably ease testing on both sides.
This also fits to the scenario of separate network providers for the subnetworks and the overall network.

The main task of the umbrella management system is to provide an integrated view for client and server layers, i.e. IP layer and OTN, spanning the two domains. There are for example optical channel trails in the OTN that realise the links between GSRs. Thus for an integrated view an association between link and realising optical channel trail is needed. To be able to provide the integrated view and to allow integrated management, the umbrella management system needs access to all network elements, i.e. it has to be able to supervise and configure the resources. Supervision of the resources includes monitoring traffic, performance and statistical data (of e.g. ports), supervision of user network interface (UNI) and network network interface (NNI), alarm and fault management, etc. Configuration includes setting parameters of routing and signalling protocols, initialising UNI and NNI data, establishing end-to-end connections in the OTN, etc. The following list gives a rough overview on the core functionality to be implemented in the LION NMS. A more detailed description of management requirements will be done in the ongoing work of WP5.

The core management functionality to be implemented in the LION NMS comprises:

- provisioning of an integrated network view
- provisioning of end-to-end connections between 2 GSRs ("lightpaths") spanning the OTN
- supervision and configuration of the UNI
- supervision and configuration of the NNI
- supervision and configuration of routing / signalling protocols for the optical layer
- management of resilience
- alarm correlation and fault localisation
- cooperation of management systems

As the main focus of LION are interworking aspects between the OTN and multiple clients, it has been agreed in LION to not consider Xcoop functionality. The focus of Xcoop are security and confidentiality aspects between multiple network providers [15]. The scenario for LION is to have two management domains of "trusted partners". Another point is, that the network level interface to be defined in LION is located between the umbrella management system and sub-NMSs, but an Xcoop interface is located between two NMSs on the same hierarchical level.

3 Architectural Concepts used for the LION Umbrella Management System

This section describes in a quite general way the concepts and techniques used for the design and implementation of the LION umbrella management system.

3.1 WMI/DCOM Architecture

Web-Based Enterprise Management (WBEM) is an industry initiative that establishes management infrastructure standards for managing the enterprise network and provides a way to combine information from various hardware and software management systems.

These standards:

- define the structures and conventions necessary to access information about the managed objects;
- support centralisation of information so that different clients and management tools can provide, retrieve, and analyse data;
- support authorised access to managed objects from anywhere in the network so that these objects can be analysed and manipulated.

WBEM results in technology that enables customers to collect, associate, and aggregate management data from diverse sources, thus creating richer and more accurate views of their enterprise environment.

The Desktop Management Task Force (DMTF) is now the focal point for WBEM initiative efforts.

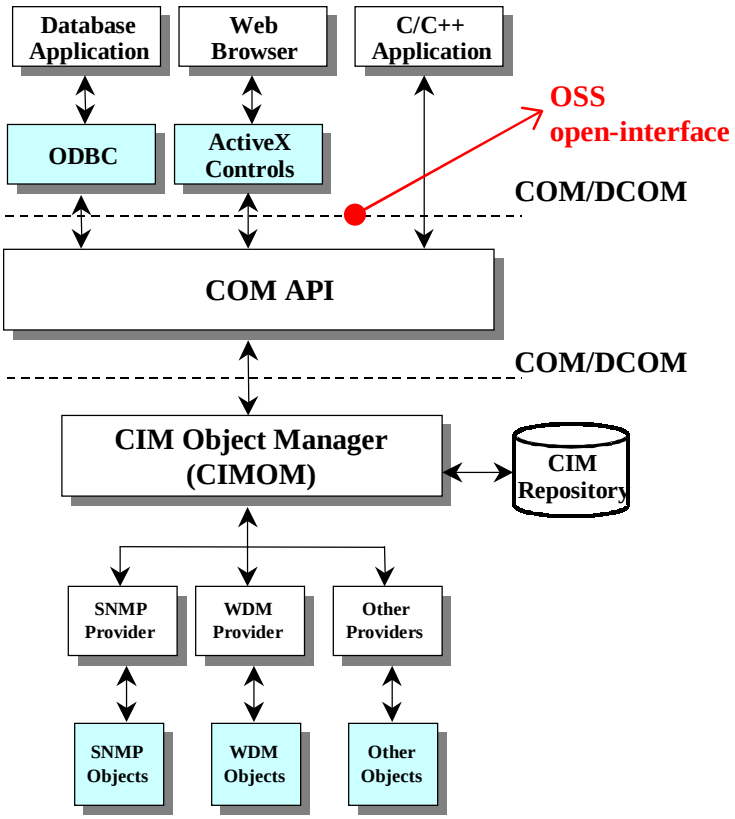

Figure 2: Diagram of WMI architecture

Windows Management Instrumentation (WMI), which is WBEM-compliant, consists of the following components (see Figure 2):

- Management applications
- Management infrastructure, consisting of the CIMOM (WMI software, that is Winmgmt.exe) and the CIM repository
- Providers
- Managed objects

Management applications are Microsoft Windows-based applications or Microsoft Windows NT/Windows 2000 services that process or display data from managed objects. A management application can perform a variety of tasks such as measuring performance, reporting outages, and correlating data.

Some examples of management applications are depicted at the top of Figure 2. Some of these applications access the COM API directly to interact with the CIMOM and the CIM repository to make management requests. Other applications use access methods such as Open Database Connectivity (ODBC) and the Hypertext Markup Language (HTML) to make their requests.

The ***management infrastructure*** consists of the CIMOM and the CIM repository. The CIMOM enables users to handle communications between management applications and providers. Users store their static data in the CIM repository. Applications and providers communicate through the CIMOM using a common programming interface, the COM API. The COM API, which supplies event notification and query processing services, is available in the C, C++ and J++ programming languages.

The CIM repository holds static management data. Static data is data that does not regularly change. The CIMOM also supports dynamic data, which is data that must be generated on demand because it is frequently changing. Data can be placed in the CIM repository by the CIMOM or by the network administrators. Third-party developers can place information in the CIM repository with either the MOF language (and its compiler) or the COM API.

Managed objects are logical or physical enterprise components. Managed objects are modelled using the CIM, and they are accessed by management applications through WMI. A managed object can be any enterprise component, from a small piece of hardware such as a cable to a large software application such as a database system.

WDM providers act as intermediaries between CIMOM (CIM Object Manager) and one or more managed objects. When the CIMOM receives a request from a management application for data that is not available from the CIM repository or for notifications of events that it does not support, it forwards the request to the provider. The provider then supplies the data or event notification requested.

Providers are standard COM or DCOM servers that can be implemented using the full range of supported server types:

- in-process Dynamic-Link Libraries (DLLs);
- local or remote Microsoft Windows NT/Windows 2000 services;
- local or remote standard executables (.exe files).

The WMI Software Development Kit (SDK) includes several built-in providers. The built-in providers, also referred to as standard providers, supply data from well-known sources such as the system registry (Registry Provider) and the Windows Driver Model (WDM) devices.

The WMI technology also supports the creation of custom providers by third-party developers. These providers service requests relating to managed objects that are environment-specific. Providers typically use the MOF language to define and create classes. They use the COM API to access the CIM repository and to respond to CIMOM requests made initially by applications.

The custom providers supply CIMOM and applications with data from custom managed objects. Both the standard and custom providers can access data from the CIM repository through the COM API.

The WMI technology categorises providers by the different features that they support. Every provider can support one or more feature set. For each feature, a different method is implemented. There are basically two types of providers: those that support data retrieval and

modification, and those that support event notification. Providers can be categorised further as shown in Table 1.

Provider type	**Description**
Class	Retrieves, modifies, deletes, and/or enumerates a provider-specific class. It can also support query processing.
Event	Generates notifications of events.
Event consumer	Supports event notification by mapping a physical consumer with a logical consumer.
Instance	Retrieves, modifies, deletes, and/or enumerates the instances of system and/or provider-specific classes. It can also support query processing.
Method	Invokes methods for a provider-specific class.
Property	Retrieves and/or modifies individual property values.

Table 1: Categorisation of WDM providers

The protocol used for communication between local and remote components is the Distributed Component Object Model (DCOM). It is an extension of the COM (Component Object Model), which defines how components and their clients interact. When client and component reside on different machines, DCOM simply replaces the local interprocess communication with a network protocol. As an extension of COM, DCOM enhances COM's security model and allows applications to be built from COM objects that reside on different machines, defining the process of data transfer across machines (for instance, load balancing technique). DCOM's architecture involves five main components that are added on top of COM: remote object instantiation, security provider, data marshalling between clients and objects, and the other two-level components that deal with the physical network connection and data transfer.

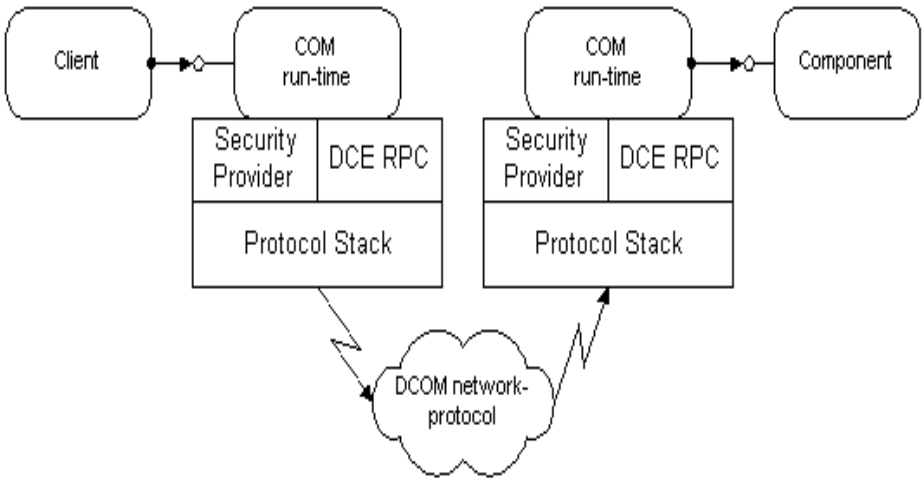

Figure 3: DCOM: COM components on different machines

Figure 3 illustrates the DCOM architecture. The COM run-time provides object-oriented services to clients and components, and uses RPC (Remote Procedure Call) and the security provider to generate standard network packets that conform to the DCOM wire-protocol standard.

For more details see [9] and [10].

3.2 Common Object Request Broker Architecture

The Common Object Request Broker Architecture is specified by the Object Management Group (OMG) which is an international organisation supported by over 750 members, including information system vendors, software developers and users. Primary goals of OMG are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments.

3.2.1 Introduction on CORBA

CORBA consists of several components which provide functionality needed to communicate between distributed software applications.

The Reference Model of the CORBA architecture consists of the following components:

- **Object Request Broker (ORB)**, which enables objects to transparently make and receive requests and responses in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments.
- **CORBA Object Services (COS)**, a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains (e.g. programming languages). For example, the Naming Service IDL specification defines operations to register and locate objects at a central name server; but it does not dictate how the operations are implemented in an application. Other examples of object services are Notification Service, Event Service, Security Service etc.
- **Common Facilities**, a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility.
- **Application Objects,** which are products of a single vendor or in-house development group which controls their interfaces. Application Objects correspond to the traditional notion of applications, so they are not standardised by OMG. Instead, Application Objects constitute the uppermost layer of the Reference Model.

The Object Request Broker, then, is the core of the Reference Model. It is like a telephone exchange, providing the basic mechanism for making and receiving calls. It provides the transport mechanism for the communication partners, the CORBA client and CORBA server (refer to top of Figure 4).

Usually the client object starts the communication and the server object responds. If it is needed that an application in a server role sends information without a previous invocation of a client, then special Object Services must be used. These are the Event Service or the Notification Service.

The interface between the server and the client is defined in **IDL** (Interface Definition Language). It describes which operations can be requested and how the server has to respond.

The client application contacts a server object, which responds on an operation request, via a unique object reference of the server object. This unique object reference is called Interoperable Object Reference (**IOR**).

Generally, the communication protocol between client and server is the General Inter-ORB Protocol (**GIOP**). But the prevalent protocol is the Internet Inter-ORB Protocol (**IIOP**) which is the GIOP based on TCP/IP (see Figure 4).

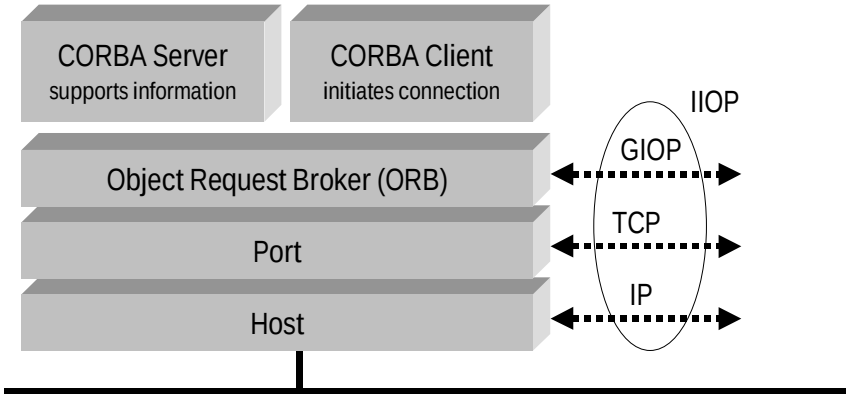

Figure 4: CORBA protocol stack

As stated above the interface definition specifies the attributes and operations of an object (in the server role), but it specifies not how the behaviour of the object should be implemented. For the implementation of client and server objects (i.e. their running code), a compiler automatically generates client stubs and object skeletons from the IDL definition of the interface. Stubs and skeletons serve as proxies for clients and servers respectively and are needed to support communication between client and object implementation (see Figure 5).

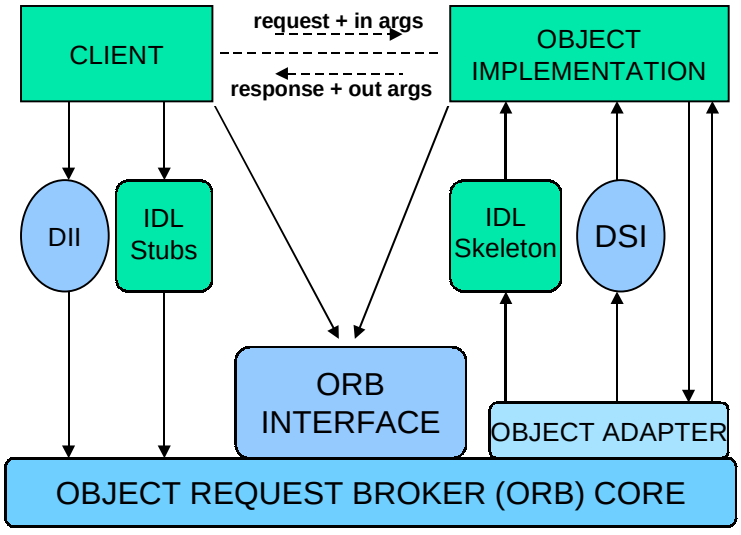

Figure 5: ORB: Client and Object Implementation in the server role

IDL Stubs: Generally, the stubs will present access to the OMG IDL-defined operations on an object. For the mapping of a non–object–oriented language, there will be a programming interface to the stubs for each interface type.

IDL Skeletons: For a particular language mapping, and possibly depending on the object adapter, there will be an interface to the methods that implement each type of object. The interface will generally be an up-call interface, in that the object implementation writes routines that conform to the interface definition and the ORB calls them through the skeleton.

Dynamic Invocation Interface (DII): The DII allows the dynamic construction of object invocations. That means not calling a stub routine that is specific to a particular operation on a particular object, but specifying the object to be invoked, the operation to be performed, and the set of parameters for the operation through a call or sequence of calls. The client code must supply information about the operation to be performed and the types of the parameters being passed (perhaps obtaining it from an Interface Repository or other run-time source).

Dynamic Skeleton Interface (DSI): The DSI allows dynamic handling of object invocations analogous to the client side's Dynamic Invocation Interface.
The implementation code must provide descriptions of all the operation parameters to the ORB, and the ORB provides the values of any input parameters for use in performing the operation. The implementation code provides the values of any output parameters, or an exception, to the ORB after performing the operation.

Object adapter: It is the primary way that an object implementation accesses services provided by the ORB. There are expected to be a few object adapters that will be widely available, with interfaces that are appropriate for specific kinds of objects. Services provided by the ORB through an Object Adapter often include: generation and interpretation of object references, method invocation, security of interactions, object and implementation activation and deactivation, mapping object references to implementations, and registration of implementations.

ORB Interface: The interface that goes directly to the ORB which is the same for all ORBs and does not depend on the object's interface or object adapter. Because most of the functionality of the ORB is provided through the object adapter, stubs, skeleton, or dynamic invocation, there are only a few operations that are common across all objects.

ORB Core: It is that part of the ORB that provides the basic representation of objects and communication of requests. CORBA is designed to support different object mechanisms, and it does so by structuring the ORB with components above the ORB Core, which provide interfaces that can mask the differences between ORB Cores.

3.2.2 CORBA for Telecommunications Management

The roots of CORBA are in the IT world and distributed applications. CORBA is commonly used for one-to-one communication between a client and a server, which may reside on different platforms. A standard example for distributed applications of this kind are banking applications. However, this one-to-one communication is not the main use case in the area of telecommunications.

The architecture developed for management purposes in telecommunications is TMN and its protocol CMIP, which in contrast to CORBA is specialised knowledge of telecom experts. Consequently, there is a trend to use CORBA for telecommunication management to profit from the bigger market of CORBA products and developers. To use CORBA for Telecommunications Management Networks special boundary conditions of network management have to be considered. Examples of such boundary conditions are:

- large size of management domains
- large number of objects to be managed
- huge amount of information exchange

As a conclusion, CORBA is less complex than CMIP, but CMIP offers a better performance and more functionality for TMN tasks. Thus CORBA has to be adapted to fit to the requirements of applications for network management.

This chapter presents a short introduction of two different approaches for a CORBA based telecommunications management framework, the "coarse grained approach" and the "fine grained

approach". This information is extracted from [2] which makes a statement to the CORBA framework described in [1].

3.2.2.1 The TMN Manager Agent Approach

The TMN architecture uses the manager-agent entity as a fundamental component of large scale, hierarchical TMN system. This TMN approach is well known and shown for comparison reasons in Figure 6.

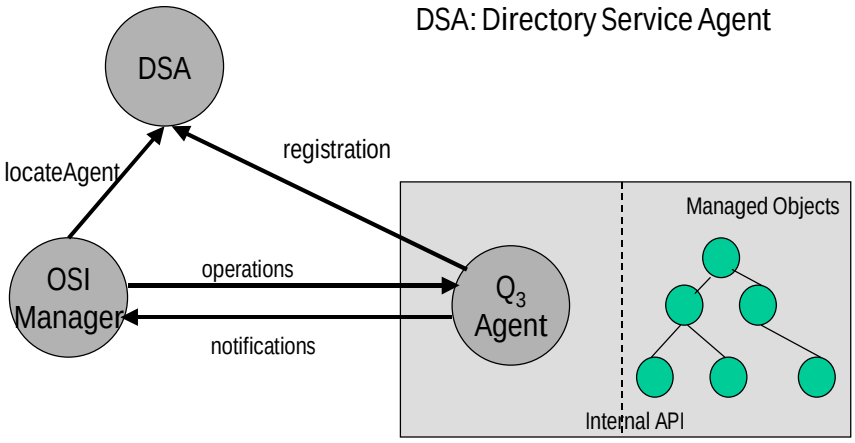

Figure 6: TMN approach

3.2.2.2 "Fine Grained" Corba-based Manager Agent Approach

The standard CORBA approach only supports one type of CORBA object, i.e. objects which are all accessed via an own IOR. To be able to distinguish these kind of objects from the objects without an own IOR, which will be introduced in the next section, these objects are called "first class" CORBA objects. If there are a lot of first class objects, the CORBA model is called "fine grained".

Using the CORBA paradigm in the network management context, each managed object is mapped into a CORBA server object (1st Class Object).

Figure 7 illustrates that associations exist from the Name Server as well as from the CORBA-manager to each managed object (1st Class Objects).

Consequently, the fine grained approach leads to

- a high number of interactions needed to perform management operations;
- a non-hierarchical structure of the 1st Class Objects.

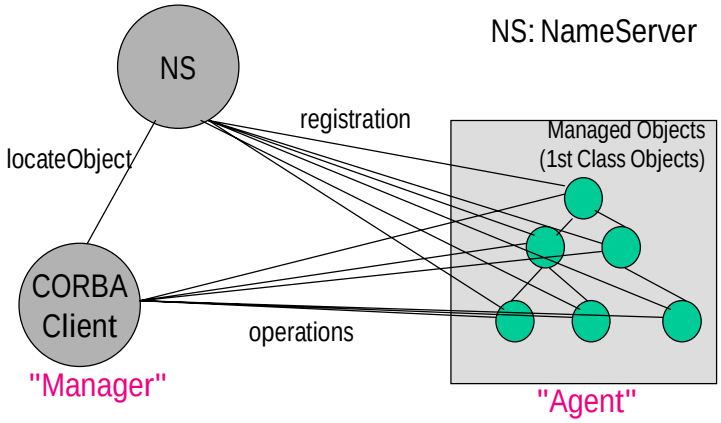

Figure 7: "Fine grained" approach

3.2.2.3 "Coarse Grained" Corba-based Manager-Agent Approach

"Coarse Grained" models are produced by introducing two concepts – "second class objects" and "collections" of second class objects. These collections are modelled as first class objects.

The second class objects are identified within their collection by some form of "identity". The identity of the second class objects is expressed using CORBA data types. Consequently, the ORB does not recognise 2nd class objects as objects. 2nd class objects do not have an interface nor do they support operations (they have pseudo-interfaces and operations as part of the collection object's interfaces and operations). Only a 1st class object can be accessed via its IOR.

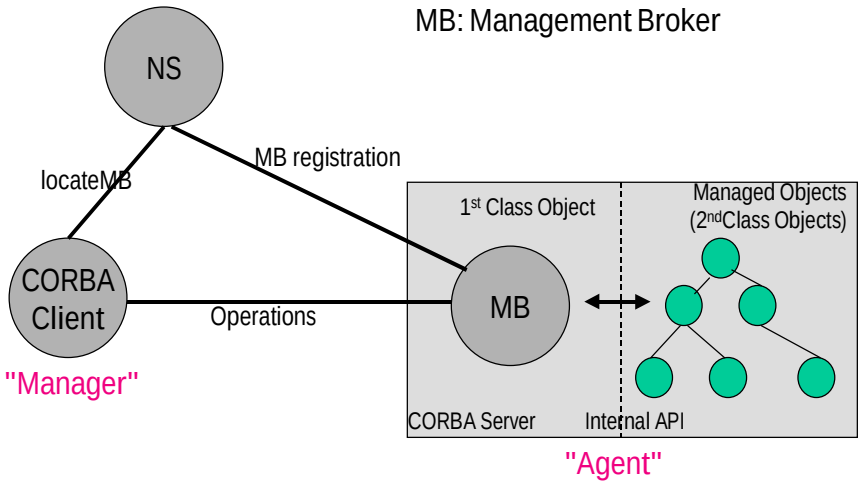

Figure 8: "Coarse grained" approach

The coarse grained approach depicted in Figure 8 needs to support only agent discovery, i.e. only one request to the Naming Service is required per MB, and CMIS interactions through CORBA, without individual 1st Class Objects (IDL interfaces) for managed objects.

This essentially means that

- the Management Broker (MB) object with a CMIS-like interface will act as an agent that provides access to managed objects which may be implemented by existing TMN platform infrastructure
- this architectural approach is almost identical to the TMN framework architecture depicted in Figure 6.
- the CORBA manager gets access to the MB interface reference through the CORBA naming services.
- Corba Object Services (COS) cannot be used for 2nd class objects, i.e., the conveniences CORBA services offer for naming, trading, transaction, etc. can only be applied on 1st class objects.

The MB-interface is unique, that means it is independent of the management information model, which is advantageous if different versions of the information model exist in parallel within the distributed management system. Mechanisms for Scoping and Filtering have to be realised by the MB.

3.2.2.4 Overview of Coarse / Fine Grained Approach Characteristics

The following table gives an overview of the characteristics of both approaches:

Characteristic	Fine Grained Framework	Coarse Grained Framework
CORBA interface granularity (no. of CORBA interfaces)	high	low (Managed Objects are represented by 2nd Class objects)
Scalability	low	high
Degree of information interchange	high	low (comparable to conventional CMIP based frameworks)
generic/specific communication interface	information model specific	generic
support of different versions of information models within the same distributed Mgmt.-System	practicable but hard to implement	native feature
implementation effort (costs)	low (framework = CORBA)	high (coarse grained framework has to be provided)
Specification Language	IDL	GDMO/ASN.1 or IDL

Table 2: Framework characteristics

3.2.3 Existing TMN-CORBA Frameworks

As described in the previous section there are special tasks for a transport mechanism in TMN which have to be covered by CORBA. Standardisation organisations like ITU-T or other organisations like OMG treat this problems and published frameworks to solve the needs of TMN.

Recommendations concerning this subject are the "Interworking Between CORBA and TMN Systems Specification" [6] published by OMG and Q.816 [7] and X.780 [8] published by ITU-T.

The OMG and the ITU-T CORBA framework are both based on the fine grained approach and offer IDL modules for the interworking between CORBA and TMN systems. This chapter introduces briefly these frameworks and reflects the influence on the LION CORBA architecture considerations.

3.2.3.1 OMG: Interworking Between CORBA and TMN Systems Specification

This OMG framework [6] supports common services (e.g. creation of objects, invocation of operations, event reporting and distribution) used for interworking scenarios and for pure Java environment scenarios. It can be divided into three levels of interfaces:

1. Generic interfaces, management model independent[2]: These "JIDM Facilities" (Joint Inter-Domain Management) provide a generic framework to access a managed domain. For example an ProxyAgent is defined which allows access to the managed object domain (Agent Application).
2. Generic interfaces, management model dependent: These interfaces represent the different "management philosophies" and support the necessary operations which are translated in protocol dependent PDUs. One of the two considered management model references are the "OSI Management Facilities", which provide the CMIS interactions in CORBA and support the OSI concepts such as scoping and filtering. The other one are the "SNMP Management Facilities", which support the SNMP interactions and Internet specific concepts (e.g. getNext).
3. Specific interfaces, information model[3] and management model dependent: These interfaces reuse and extend the generic CORBA facilities of the corresponding management reference model (OSI or SNMP Management Facilities) in an information model specific way. It is beyond the OMG framework [6] to specify any information specific interfaces (e.g. X.721, SDH; MIB2)

For this interworking framework the OMG makes an important assumption, assuming that future ORB implementations will be able to deal with a huge amount of objects and information exchange [6]. Quotation: "*The design of CORBA/TMN interworking facilities assumes good ORB and Service implementations. Specifically, it is assumed that CORBA-compliant ORB implementations are being built that support efficient local and remote access to a very high number of objects and have performance characteristics that place no major barriers to the pervasive use of distributed objects for virtually all service and application elements*."

Based on this assumption the OMG uses for its interworking framework the fine grained CORBA approach. However as will be stated later (see section 4.2.1), LION has to stick to actual ORB and Service implementations, which do not fulfil the assumptions made by the OMG.

3.2.3.2 ITU-T: Framework for CORBA-based TMN interfaces

The guideline for defining CORBA managed objects X.780 describes the rules for modelling manageable resources. Some important subjects are listed below:

Access granularity: X.780 uses a instance-grain (= fine grained) modelling approach, which means that each modelled resource is accessible using a unique CORBA object reference. This granularity approach bases on the Portable Object Adapter (POA) as well as for passing objects by value which is offered by CORBA 2.3.1! The POA is important to the framework because it

[2] Possible management models are OSI Management or Internet Management (SNMP).

[3] An Information model specifies the specific functionality of a certain management model.

enables implementations based on this framework to scale up to millions of instantiated objects, a magnitude required for network management applications.

Containment: Containment has traditionally been a very important relationship in network management applications because it is a convenient means to systematically arrange resp. to identify the large number of resources that typically must be managed. X.780 guidelines require that a unique name be assigned to each managed object, based in part on the name of the object that contains it. One possible solution is the use of the Naming Service, but it is stated that there is no requirement to use exclusively the Naming Service

Object Creation and Deletion: The CORBA ORB does not provide clients with a means to create objects on remote systems. Instead, *factory* objects are instantiated by remote systems, and these factory objects provide operations that may be invoked by clients to create objects on the remote system. X.780 specifies that a factory IDL interface shall be defined for each managed object IDL interface in an information model.

The Q.816 framework for CORBA-based TMN interfaces provides a collection of network management functions. A central piece of the framework is a set of CORBA Common Object Services. This framework defines their roles in network management interfaces, and defines conventions for their use. The framework also defines support services that have not been standardised as CORBA Common Object Services, but are expected to be standard on network management interfaces conforming to this framework.

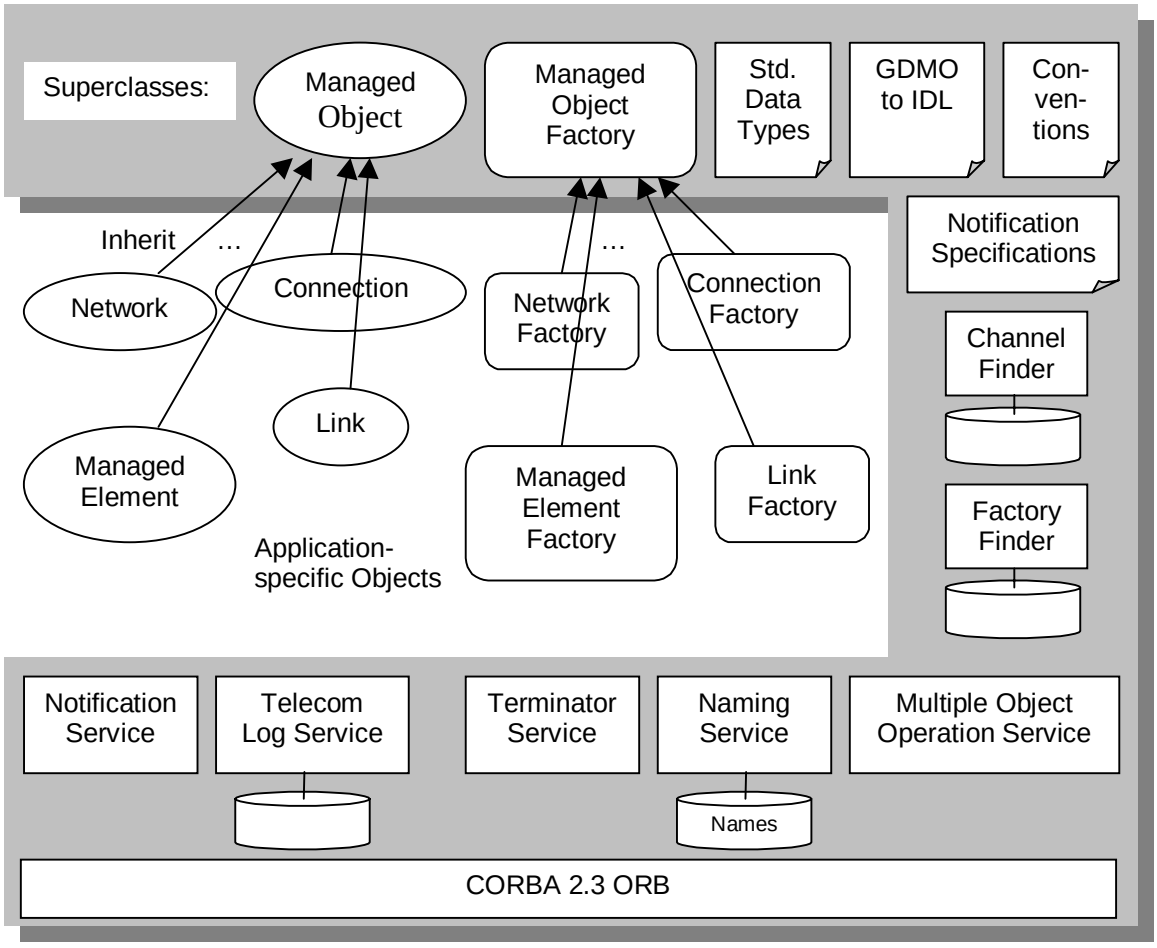

Figure 9: Overview of ITU-T Framework; Source Q.816 [7]

In Figure 9 the framework as proposed by ITU-T is depicted in grey. Part of the framework are the CORBA ORB and the services that compose the framework (e.g. Notification Service, Telecom Log service, Naming Service, Channel Finder, etc.). Some of these services have to maintain a database to be able to perform their tasks. Along the top of Figure 9 icons representing two superclasses are shown, one for managed objects and one for managed object factories. Each of the managed objects and managed object factories supported by this framework must ultimately inherit from these superclasses, respectively. In the middle of the Figure some examples for these application-specific objects supported by the framework are depicted. The icons of pages with up-turned corners represent standard object modelling conventions.

Table 3 summarises the versions of the services that are required to support this ITU-T framework.

Service	**Version**
ORB	2.3.1
Naming Service	1.0
Notification Service	1.0
Telecommunications Logging Service	1.0
Asynchronous Messaging	(determined by client system)
Transaction Service	1.1

Table 3: CORBA Services Versions; Source[7]

3.3 DCOM / CORBA Mapping

In LION two sub-NMS, one based on CORBA the other based on DCOM have to interact. Thus some kind of mapping or transformation between DCOM and CORBA is necessary. For the implementation of this mapping a commercial DCOM/CORBA bridge will be used. In the following a more general description of the relationships between CORBA and DCOM is given. This information is an abstract of the white paper reported in [12], where is done an overall architectural comparison of DCOM and CORBA.

Object, Client and Server:

A COM object can support multiple interfaces, each representing a different view or behaviour of the object. An interface consists of a set of methods related to a certain functionality. A COM server can create object instances of multiple object classes. A COM client interacts with a COM object by acquiring a pointer to one of the object's interfaces and invoking methods through that pointer, as if the object resides in the client's address space. COM specifies that any interface must follow a standard memory layout (which is the same as the C++ virtual function table). Since the specification is at the binary level, it allows integration of binary components possibly written in different programming languages such as C++, Java and Visual Basic.

A CORBA object is represented to the outside world by an interface with a set of methods. A particular instance of an object is identified by an object reference, the IOR. The client of a CORBA server object acquires its object reference and uses it as a handle to make method calls, as if the object is located in the client's address space. The Object Request Broker (ORB) acts as the object bus over which objects transparently interact with other objects located locally or remotely. It is responsible for all the mechanisms required to find the object's implementation, prepare it to receive the request, communicate the request to it, and carry the reply (if any) back to the client. The object implementation interacts with the ORB through either an Object Adapter (OA) or through the ORB interface (see also Figure 5).

Proxy/Stub/Skeleton:

To invoke a remote function, the client makes a call to the client stub. The stub packs the call parameters into a request message, and invokes a wire protocol to ship the message to the server. At the server side, the wire protocol delivers the message to the server stub, which then unpacks the request message and calls the actual function on the object.

In DCOM, the client stub is referred to as the proxy and the server stub is referred to as the stub.

In contrast, the client stub in CORBA is called "stub" and the server stub is called "skeleton". Sometimes, the term "proxy" is also used to refer to a running instance of the stub in CORBA.

Interface Definition Language (IDL) and Proxy/Stub/Skeleton Code:

The interface of an object is described in an interface definition language (IDL). The interfaces defined in an IDL file serve as a contract between a server and its clients. Clients interact with a server by invoking methods described in the IDL.

CORBA also supports multiple inheritance at the IDL level, but DCOM does not. Instead, the notion of an object having multiple interfaces is used to achieve a similar purpose in DCOM.

An IDL file defines the interfaces and its methods. Running the IDL file through an IDL compiler in both DCOM and CORBA generates the proxy/stub/skeleton code and the interface header file that are used by both the server and the client. The DCOM IDL file also associates multiple interfaces with an object class. Note that, in DCOM, each interface is assigned a Globally Unique Identifier (GUID) called the Interface ID (IID). Similarly, each object class is assigned a unique class ID (CLSID). Also, every COM interface must inherit from the IUnknown interface that consists of a QueryInterface() method for navigating between different interfaces of the same object, and two other methods AddRef() and Release() for reference counting. Reference counting provides a lifetime control mechanism that allows a COM object to keep track of its clients and can delete itself when it is no longer needed.

In the CORBA implementation, the IDL compiler generates from the interface definition the interface class in the header file. A way of associating the implementation class with the interface class is the inheritance approach. Since current CORBA does not specify what the skeleton class looks like and what the name of the base class is, the server code is not portable to other ORB products. To resolve this issue, Portable Object Adapter (POA) was introduced in CORBA 2.3. POA corrects this problem and specifies the name for the base class.

The DCOM program creates an event and waits on that event which is signalled when all active server objects are deleted and so the server can exit. The actual client requests are handled concurrently by different threads from a thread pool.

Similarly, the CORBA server program instantiates an instance of the base class and then blocks to receive the incoming client requests. If the server does not receive any requests during a default timeout period (which can be set by the programmer), it gracefully closes down. The client requests are handled either serially or by different threads, depending on the activation policy used for the object server.

The DCOM client code tends to be longer than CORBA client code due to the additional IUnknown method calls. This may not be true for DCOM clients written in Java or Visual Basic, where the virtual machine layer takes care of the IUnknown method calls and hides them from the programmers. Even in a C++ client, smart interface pointers can be used to hide the reference counting.

Registration process for the server:

After compiling and before executing the programs, both DCOM and CORBA require a registration process for the server.

In CORBA, the association between the interface name and the path name of the server executable is registered with the implementation repository.

In DCOM, the association between the CLSID and the path name of the server executable is registered with the registry. In addition, since a DCOM interface proxy/stub is itself a COM object, its associated server (in the Dynamic Link Library (DLL) form) also needs to be similarly registered.

Type library and Interface Repository:

In DCOM, type information for interface methods is stored in a type library generated by the IDL compiler and assigned a GUID. It can be used through the IDispatch interface to perform dynamic invocation. It can also be used for type library-driven marshalling; instead of using a separate proxy/stub DLL that contains information specific to an interface, a generic marshaller can perform marshalling by reading type library information.

In CORBA, the IDL compiler generates the type information for each method in an interface and stores it in the Interface Repository (IR). A client can query the interface repository to get run-time information about a particular interface and then use that information to create and invoke a method on the object dynamically through the Dynamic Invocation Interface (DII). Similarly, on the server side, the Dynamic Skeleton Interface (DSI) allows a client to invoke an operation on an object that has no compile time knowledge of the type of object it is implementing.

The programmers' view of DCOM and CORBA:

How a client requests an object and invokes its methods, and how a server creates an object instance and makes it available to the client is totally hidden from the programmers. The client and the server programs interact as if they reside in the same address space on the same machine. The main differences between DCOM and CORBA at this layer include how a client specifies an interface and COM's class factories and the IUnknown methods.

In DCOM, the use of class factories in COM is optional. A server object can actually call CoRegisterClassObject() to register any interface pointer, and clients can invoke another COM API named CoGetClassObject() to retrieve that pointer. Besides, CoCreateInstance() does not necessarily create a fresh instance. Inside IClassFactory::CreateInstance(), a server can choose to always return the same interface pointer so that different clients can connect to the same object instance with a particular state.

In CORBA, an object can be activated by invoking any method on an existing object reference[4]. The client may attach to an existing instance instead of a new instance, if there is any existing instance matching the requested type.

How server objects are registered and when proxy/stub/skeleton instances are created:

To send data across different address spaces requires a process called marshalling and unmarshalling. Marshalling packs a method call's parameters (at a client's space) or return values (at a server's space) into a standard format for transmission. Unmarshalling, the reverse operation, unpacks the standard format to an appropriate data presentation in the address space of a receiving process.

[4] Some vendors provide special method calls, e.g. _bind() operation in Orbix, to activate a server object and obtain its object reference.

Note that the marshalling process is called standard marshalling in DCOM terminology. DCOM also provides a custom marshalling mechanism to bypass the standard marshalling procedure. By implementing an IMarshal interface, a server object declares that it wants to control how and what data are marshalled and unmarshalled, and how the client should communicate with the server. In effect, custom marshalling provides an extensible architecture for plugging in application-specific communication infrastructure. It can be useful for client-side data caching, for fault tolerance, etc.

The OA is responsible for connecting the object implementation to the ORB. OAs provide services like generation and interpretation of object references, method invocation, object activation and deactivation, mapping object references to implementations. Different object implementation styles have different requirements and need to be supported by different object adapters, e.g. object-oriented database adapter for objects in a database. The BOA defines an object adapter, which can be used for most conventional object implementations. CORBA specifications do not mandate how the ORB/BOA functionality is to be implemented[5].

Wire protocols:

At this layer, the main difference between DCOM and CORBA includes how remote interface pointers or object references are represented to convey the server endpoint information to the client, and the standard format in which the data is marshalled for transmission in a heterogeneous environment. Note that CORBA does not specify a protocol for communication between a client and an object server running on ORBs provided by the same vendor. The protocol for inter-ORB communication between the same vendor ORBs is vendor dependent. However, in order to support the interoperability of different ORB products, a General Inter-ORB Protocol (GIOP) is specified. A specific mapping of the GIOP on TCP/IP connections is defined, and known as the Internet Inter-ORB Protocol (IIOP).

DCOM wire protocol is mostly based on OSF DCE RPC specification, with a few extensions. That includes remote object reference representation, an IRemUnknown interface for optimising the performance of remote IUnknown method calls, and a pinging protocol. Pinging allows a server object to garbage-collect remote object references when a remote client abnormally terminates. When a client obtains an interface pointer to a remote object for the first time, the ping client code on the client machine adds the object to a ping set and periodically sends a ping to the server machine to let it know that the client is still alive. Missing a predetermined number of consecutive pings indicates that the client has abnormally terminated and the interface pointers that it holds can be released. To optimise performance, pings are sent on a per-machine basis and in an incremental way. They can also be piggy-backed on normal messages. Whenever necessary, the ping functionality can also be turned off to reduce network traffic.

[5] Orbix builds the ORB/BOA functionality into two libraries and a daemon process (orbixd). The daemon is responsible for location and activation of objects. The two libraries, a server-side library and a client-side library, are each linked at compile time with server and client implementations, respectively, to provide the rest of the functionality. It is important to note that the recently introduced POA will be a replacement for BOA. The POA specifications provide portability for CORBA server code and also introduce some new features in the Object Adapter.

4 Architecture of the LION Umbrella Management System

4.1 Overall Architecture

The present chapter describes the high level architecture of the LION network management system in more detail. For this description a top down approach has been chosen, i.e. it starts from a quite schematic architecture to more and more details.

For the architecture design it had to be considered, that there are two administrative domains, each managed by a separate NMS implemented by a different technique (CORBA and WBEM). Furthermore, it had to be taken into account that each NMS has to deal with different management technologies for accessing the network elements (CMIP, WDM, SNMP, CLI). For an integrated network view spanning both administrative domains, an umbrella network management system is required. This leads to the need for a network level interface, for which the exact location and layout has to be defined. The preliminary results for the design of the logical architecture and the location of the NL-IF will be described in the present chapter, while the "layout" of the NL-IF is described in chapter 4.2.

In the LION field trial, there will be two network management systems (a T-Nova NMS and a CSELT NMS) interoperating via a CORBA network level interface (see Figure 10). Both NMS are used for managing CMIP resp. WDM (Windows Driver Model) resources as well as SNMP resources. The CSELT NMS is based on WBEM, i.e. it is built using the DCOM architecture. Thus a DCOM/CORBA bridge is needed on CSELT side to be able to communicate (via the CORBA interface) with the CORBA based T-Nova NMS.

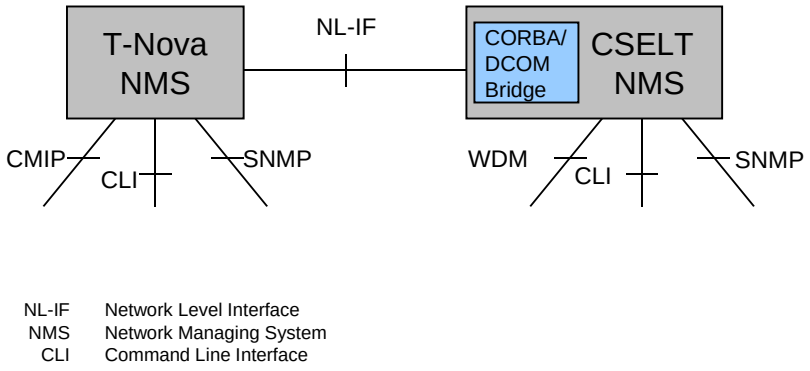

Figure 10: Overall Architecture

As already stated in chapter 2, both NMSs have to provide a view on their respective domain subnetwork and an integrated view on both domains. Thus each NMS has been divided into two functional blocks, a so called network level operating system (NL-OS) and a network element level operating system (NEL-OS) as shown in Figure 11.

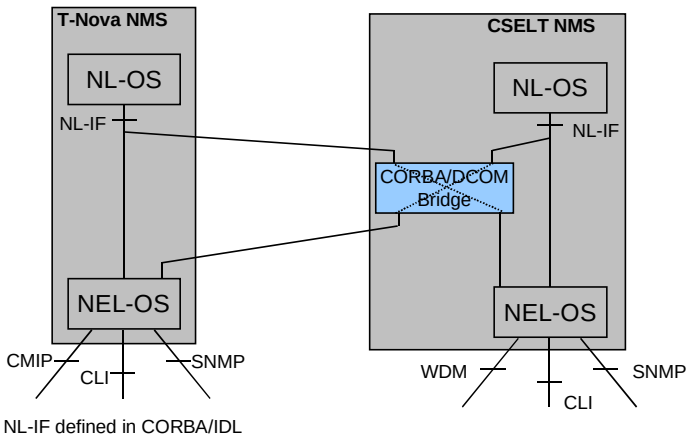

Figure 11: Logical Architecture

In the following, the idea behind NEL and NL operating system and their functionality will be described further. The functionality of both NMS, that has to be realised through this functional components, comprises element manager functionality (address single network elements) as well as network manager functionality (provide overview on the whole network).

A result of the LION project will be the definition of a Network Level Information Model (NL-IM). The NL-IM specifies in a technology (IP, OTN) and protocol (CMIP,WDM) independent way objects on network level, like e.g. "network", "subNetwork", "link", "trail" etc., with their attributes and operations. The Network Level Interface (NL-IF) defines the protocol dependent realisation of the NL-IM. In case of LION the protocol is IIOP (the "CORBA protocol") and the core of the NL-IF will be defined in IDL. (For further reflections on NL-IM and NL-IF see also sections 4.2.1.3.)

For the present chapter an exact distinction between NL-IM and NL-IF is not important. The only important facts here are at the one hand, that the interface between NL-OS and NEL-OS comprises a collection of network level objects with their attributes and operations. And on the other hand an important fact is, that only a single NL-IM exists, i.e. each label "NL-IF" in the figures stands for the same collection of NL objects. Anyway, seen from the implementation point of view there is a crucial difference between the NL-IF located between two NMSs and the NL-IF inside each NMS: the former is a physical interface between two management systems. And the second will most probably be (dependent on the realisation of the NMS) just a logical interface.

The decomposition of each sub-NMS into NL-OS and NEL-OS implies, that the NEL-OS mainly acts as a server for the NL-OS. This means the NEL-OS presents a network level view of its administrative domain to the NL-OS as indicated in Figure 12. Thus the NEL-OS has the following main tasks:

1. to access network elements (NEs) of its administrative domain.
2. to implement functionality (attributes and operations) of NL objects, e.g. "subNetwork" and "trail". For this it has to:
   - collect information from NEs needed to build the network level view
   - build the network level view for its administrative domain by creating e.g. "trail" objects

In LION there are two administrative domains, a CSELT domain and a T-Nova domain. Thus there will be most probably two NEL-OSs, implementing the two domains and representing them by e.g. a "subNetwork" object. As there is no interface between the NEL-OSs each NEL-OS can only "see" its own subnetwork / its own domain.

While the NEL-OS provides a view of a single subnetwork, the NL-OS should give an overall view of the entire network (e.g. for establishing end-to-end connections). For this it collects information from all connected NEL-OSs about the "subNetwork" objects and creates e.g. "link" objects to build up a network level view for the whole network (see Figure 12). The NL-OS acts mainly as a client of the NEL-OS.

Figure 12 is strongly related to Figure 1: the NL-OS here corresponds to the umbrella NMS there, and the two NEL-OSs correspond to the T-Nova resp. CESLT NMS in Figure 1. But for the following explanations we will stick to the terms NL-OS and NEL-OS instead of umbrella NMS and sub-NMSs, as we are going to talk about functional blocks and their functionality rather than distinct network management systems.

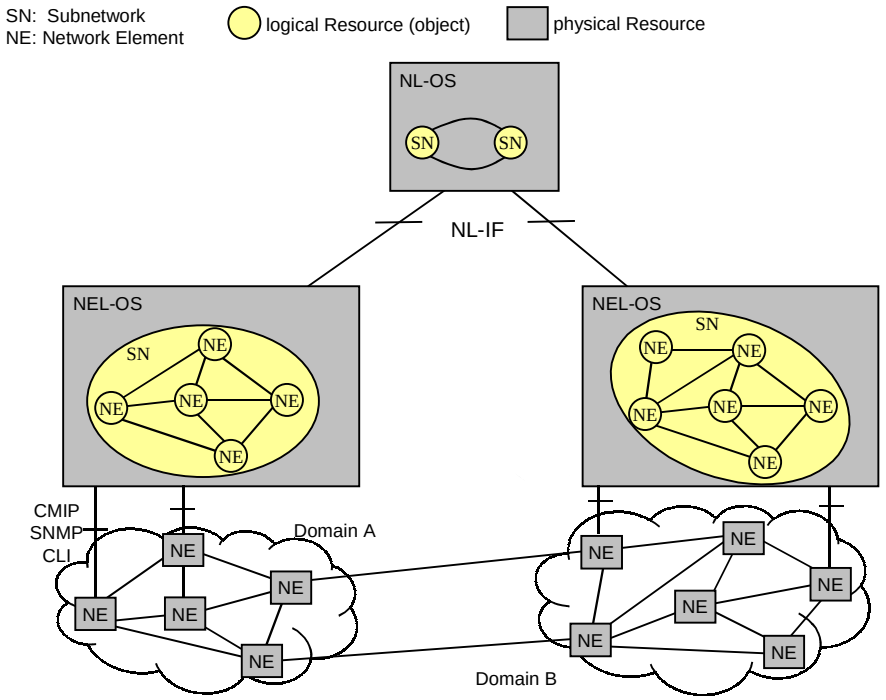

Figure 12: Network Views of NEL-OS and NL-OS (logical architecture)

The above description focussed on the functionality of NL-OS and NEL-OS and not on the LION field trial configuration. Thus Figure 12 depicts only a single NL-OS to make the relationship between NL-OS and NEL-OSs easier to understand. This also would fit to a scenario, where the subnetwork providers provide their infrastructure to a third provider (compare to Figure 1 and footnote 1 on page 10). But as for the LION field trial there are two complete network management systems, there will be also two NL-OSs as depicted in Figure 13. However, there will be no interface between the two NL-OSs. From a theoretical point of view one can think of both NL-OS providing exactly the same view on the whole network. But from a practical point of view this cannot be guaranteed, as there is no interface between the NL-OSs. This is because each NL-OS has to create and manage the object instances representing the physical resources (e.g. subnetworks and the links between them) for its own. As the NL-OSs cannot be synchronised it is not guaranteed that (during instantiation of the objects) the attributes are given the same values by both NL-OSs. As a consequence, the NL-IM has to be defined in such a way, that no common view is needed for logical resources representing the overall network view. For example it is not possible that one NL-OS restricts access on a link between subnetworks for the other NL-OS.

The creation of logical resources, that represent the physical resources building an administrative domain, should only be allowed for the respective NEL-OS. This also holds for putting a resource out of service or in service. All connected NL-OSs should be notified of object creations or attribute

value changes. Reservation of resources, service creation etc. is done via the NL-OS and is allowed in both domains. This also has to be considered for the definition of the NL-IM.

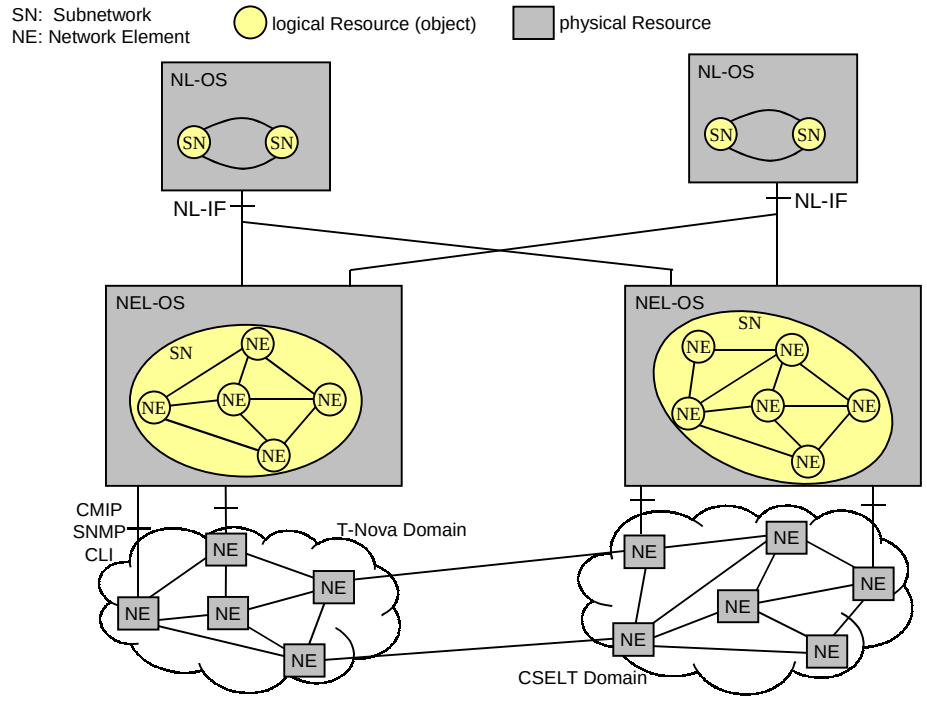

Figure 13: Network Views of NEL-OS and NL-OS (realisation in LION)

4.2 CORBA based Network Layer Interface

As stated above not only the overall and detailed software architecture has to be defined, but also the location and the "layout" of the network level interface. The location of the NL-IF has been defined in the previous chapter. This chapter is concerned with the framework of the NL-IF, i.e. which of the CORBA approaches and frameworks described in section 3.2 will be adopted and what consequences this has for the definition of the NL-IF and NL-IM.

4.2.1 Adopted CORBA Approach and Framework

LION follows the opinion of ETSI [2] and uses the "coarse grained" framework for the Network Level Interface. Choosing the "coarse grained" framework LION intentionally doesn't follow the "fine grained" recommendations of OMG [6] and ITU-T [7,8].

The chosen coarse grained framework has the advantages (see also Table 2) of:

- reduced Number of (remote) IIOP operations;
- different versions of NL Information Models can coexist. The IDL specification of the NL-IF does not need to be changed, if new managed objects or operations are added. Only the UML part of the NL Information Model (and consequently the implementation) has to be adapted. For the distinction of NL-IF and NL-IM see Figure 16.

Another reason for not following ITU-T, which chooses the fine grained approach is, that the ITU-T framework bases on CORBA 2.3.1, while LION has to stick to CORBA 2.1 due to the restrictions of current software implementations (compare with section 5.1).

Besides, the OMG and ITU-T rely on the assumption that the performance of (future) ORB implementations (e.g. Naming Service) will be sufficient for telecommunication purposes. In our opinion this is presently not guaranteed.

4.2.1.1 LION Benefits of the OMG/ITU-T Frameworks

OMG [6]:

Generally speaking, a benefit of [6] can be seen for the LION NEL-OS tasks. It is possible for LION to make use of the IDL defined in [6] for the so-called Manager Side Gateways. In LION this will be (for the CORBA based sub-NMS) a SNMP and a CMIP gateway. The IDL will not be part of the IDL definition of the LION NL-IF, but it could be used as a starting point for an NEL-OS internal mediation and translation from a common management operation to protocol specific commands (and PDUs) for SNMP and CMIP. The IDL definition within the NEL-OSS can be translated and used for an internal API for the gateways towards the network elements.

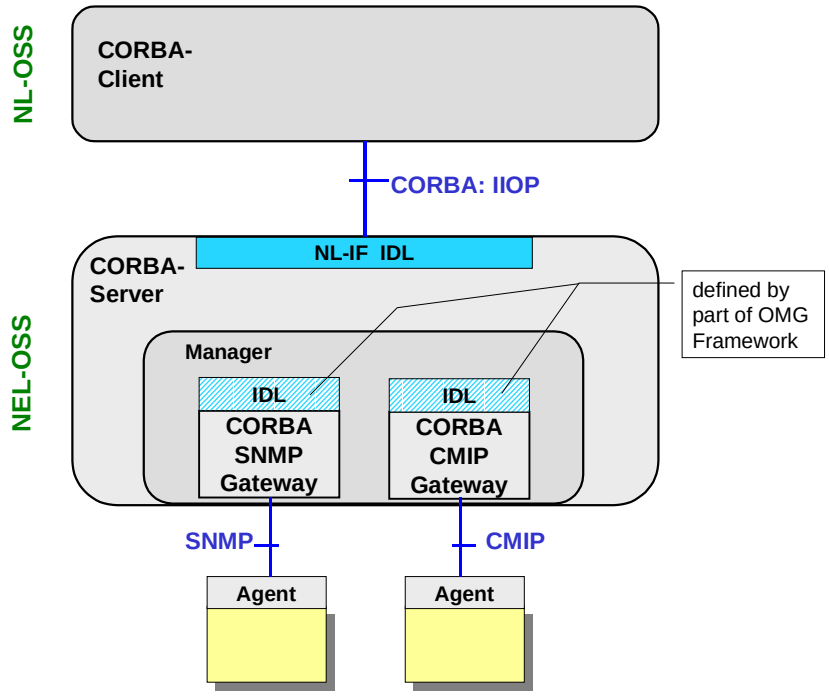

Figure 14: "Location" (logical) of a possible OMG Gateway IDL in LION architecture

ITU-T [7,8]

Generally, a lot of the framework defined in [7] and [8] can be reused. However, it is necessary to consider the differences caused by using the coarse grained approach for LION. Some parts of the IDL definitions given in [7,8] can be used for the first level and second level objects; while other parts can give ideas for the needed functions for the second level objects.

Reusable parts:

- naming conventions of Naming Service.
- specified notification format.
- ITU-T defined Services: Termination Service, Factory Finder (not necessary for the prototype), Channel Finder (not necessary for the prototype).
- possibly, the superclasses managed object, managed object factory.

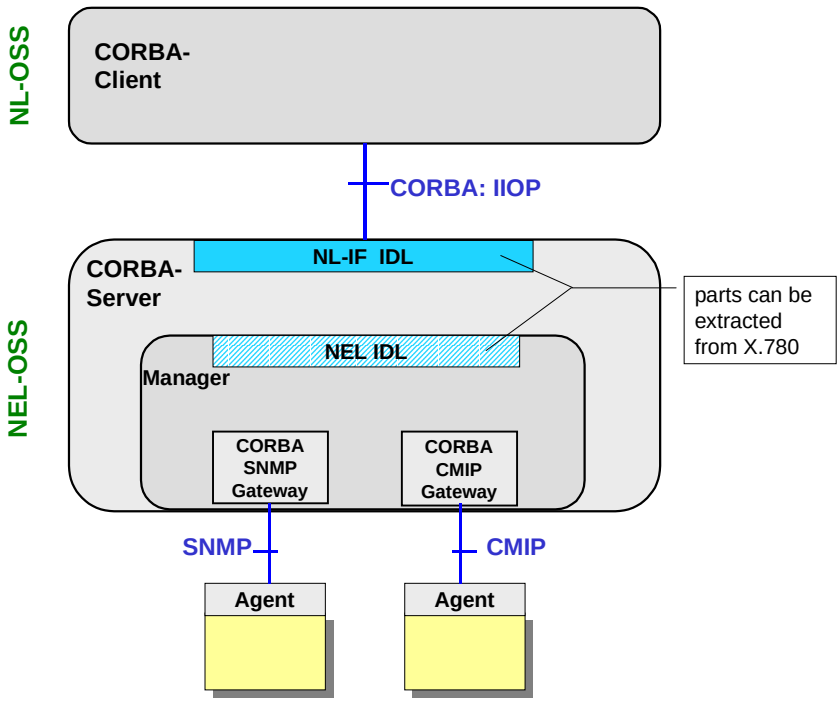

Figure 15: "Location" (logical) of ITU-T fine grained IDL in LION architecture

4.2.1.2 Consequences for the NL Interface

The use of the "Coarse grained" framework for the LION Network Level Interface (NL-IF) has the following consequences:

- Only the first level objects, which are independent of the LION NL-IM, have to be defined in IDL. They form the "Coarse grained" framework.
- The second level objects, which form the LION NL-IM, are **not** defined in IDL (note: distinguish between NL Interface (NL-IF) and NL Information Model (NL-IM)).
- This NL Information Model, which is represented by the second level objects, should be defined in UML. It defines the internal API of Figure 16 (see also Figure 8). Part of the NL-IM will be also the definition of data types, which may be defined in ASN.1.

In other words, the "coarse grained framework" defines possible operations to access the second level objects (objects of the NL-IM) via CORBA. The information what should be done is exchanged with help of parameters which transport the information within an "ANY" data type container.

The definition of which kind of data is allowed to be used for this container is defined in the NL-IM and its data type definitions. Thus it is possible to process the operation request at the server side.

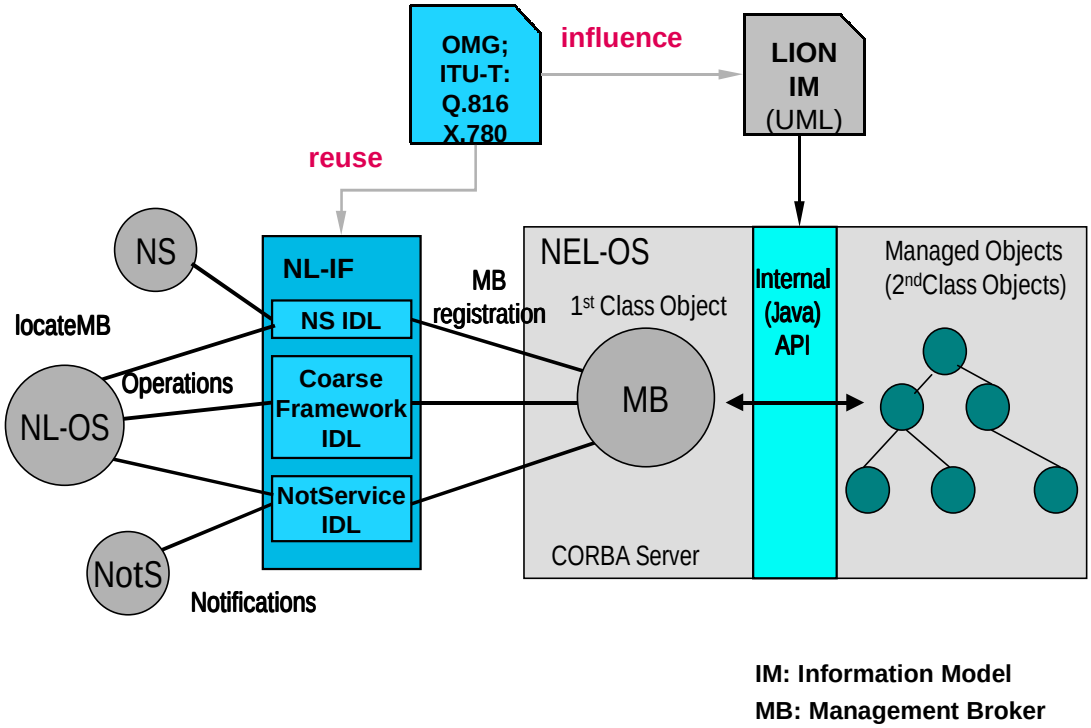

Figure 16: Network Level-Interface and Network Level Information Model

4.2.1.3 Working steps for the NL Interface

To reach the goal of a Network Level Interface (NL-IF) definition the following working steps are necessary:

1. Identification of the NL management requirements
2. Description of these requirements by UML use cases ("TMF methodology")
3. Definition of the NL-IM by UML class diagrams; so the NL-IM is protocol independent[6].
4. Definition of the NL-IF, i.e. definition of the coarse grained framework in IDL (see section 3.2.2.3)

A complete specification of the NL-IF will consist of the IDL definition of the coarse grained framework and the UML definition of the NL-IM, which is needed to define which kind of information can be exchanged across the NL-IF. Specifications of the Naming Service and the Notification Service in IDL are also needed. The following list summarises which definitions should build the complete NL-IF:

1. NL-IM as UML class diagrams, containing attributes, operations and data types.
2. Coarse grained framework in IDL (including e.g. Factory Finder, Factories).
3. CORBA Notification Service Framework in IDL (see [5]).
4. CORBA Naming Service Interface in IDL (see [4]).

[6] In case of the fine grained approach, the NL-IF would be the "translation" of the UML classes of the NL-IM into IDL.

4.2.2 CORBA Object Services Used in LION

As described in the above section, the functionality specified in the NL-IF also comprises a naming and a notification service, besides the functionality defined in the NL-IM, which is mainly realised by the NEL-OS. In CORBA the naming and notification service are realised by separate software components, which communicate via the ORB. In the present chapter the overall physical architecture is described. Furthermore, as there are two NMSs based on different technologies, it has to be decided on the number, location and realisation (CORBA or DCOM) of the services. Thus the present chapter presents the results for the design of the overall architecture. The preliminary architecture of each sub-NMS (T-Nova and CSELT) is described in chapter 5.

4.2.2.1 Naming Service

To execute an operation of a remote server object the client needs the IOR of the server object. There are several solutions to receive the IOR. One possible way is (and this is the one chosen for the LION NMS) to store the IOR at a Naming Service implementation, which is known by server and client. For LION it has to be decided in addition whether one centralised Naming Service should be used, or whether two Naming Services should be instantiated, one in each management domain.

The OMG CORBA standard offers the IDL specification of this Naming Service [4]. Either a commercial Naming Service implementation or a self implemented solution can be used. Obviously, different commercial solutions could present interoperability problems.

Comparison of Centralised/Distributed Naming Service
The main advantage of a centralised Naming Service implementation is to avoid consistency problems. This solution could be implemented on T-NOVA side, as shown in Figure 17.

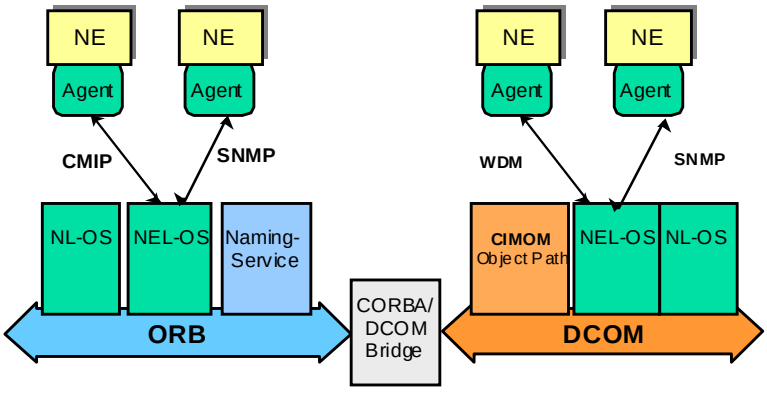

Figure 17: Centralised Naming Service on T-NOVA side

A distributed Naming Service consists of two Naming Services as illustrated in Figure 18. This has the advantage that CSELT and T-Nova are independent of the status of the single (remote) Naming Service. Every domain has its own CORBA Naming Service. This solution simplifies the tests within one domain. For a prototype implementation the disadvantage of inconsistency problems is not of relevance.

For this reason, the distributed Naming Service solution is adopted for testing purposes, while the centralised Naming Service is meant as a target solution for the final management configuration of the LION test bed.

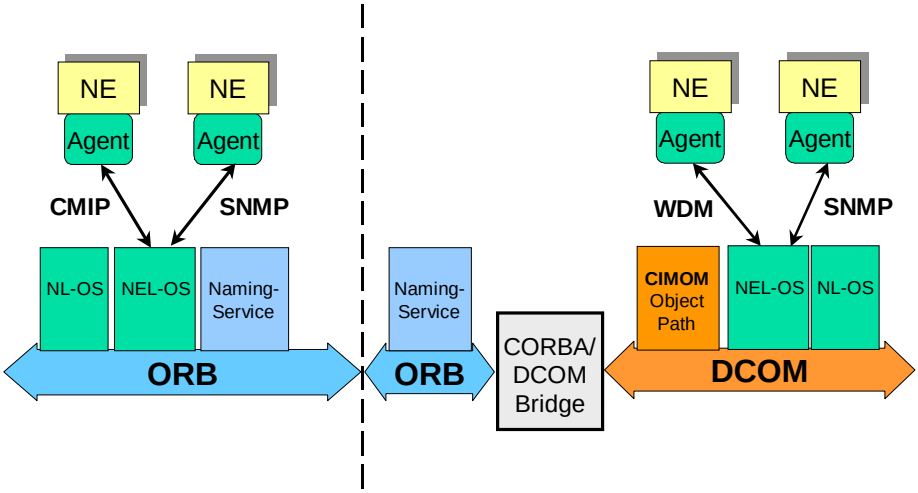

Figure 18: Distributed Naming Service on both sides

In any case: seen strictly from the T-Nova subsystem side, the CSELT subsystem is just another CORBA system (see Figure 19).

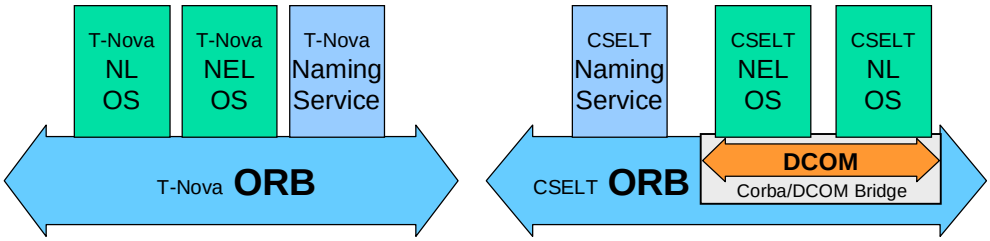

Figure 19: CSELT subsystem as seen from T-Nova subsystem

Mapping of Containment Tree on Management Broker Objects

In the coarse grained approach the Management Brokers (MBs) are the only objects that are registered at the naming service. Accessing second class objects is done via the management broker. The most obvious mapping of MB to 2nd class objects is: one MB representing the whole containment tree of 2nd class objects. However it might be useful to divide the containment tree on more than one management broker. The coarse grained approach doesn't state how many MBs are allowed. Thus some further study is needed to decide how many of these MB instances should exist (and have to be registered at the Naming Service) and which part of the MIB they shall represent. That means which MB instances allow the access to which Managed Objects.

The answer to this question should be found during the definition of the information model of the NEL.

4.2.2.2 Notification Service

In CORBA the normal order of information exchange is that the client initiates the communication. If the server should start the information exchange, a special CORBA Object Service (COS) has to be used. Generally, for this COS two choices exist:

- Event Service;
- Notification Service.

The Event Service is not considered because the ITU-T recommends the Notification Service and has already defined the notification structure in Q.816 [7].

“Basic” CORBA implementation products do not include the COS Notification Service, so in the LION umbrella management system the notifications could be realised either with

1. a commercial CORBA Notification Service [e.g. from IONA] or
2. a self implemented “small” Notification Service.

In LION, a CORBA Notification Service should be provided these following functions:

- Distribution of Notifications from Server to Client;
- Filtering and profiling of the notifications;
- De-coupling Notification supplier and receiver.

The main advantage of the CORBA Notification Service is the handling of a big amount of suppliers and receivers. It is able to filter and queue the received notifications before they are distributed to the different receivers.

The relevant situation in the LION testbed is:

- the number of CORBA Notification receivers is given by only 2 NL-OSs;
- the number of CORBA Notification suppliers is given by only about 15 NEs.

Consequently, filtering and queuing is not the main functionality required for LION implementation.

On that basis, it was agreed to implement an own small Notification Service and not to buy an oversized commercial solution. Besides that, a commercial solution has to be configured as well. Operations like (un)subscribing at the Notification Service have to be implemented at any case.

Nevertheless, the standardised IDL of the CORBA Notification Service will be used, because ITU-T (Q.816) recommends to use the Notification Service and has already defined the notification structure.

Additionally, it is possible to use a commercial product with the LION prototype after small modifications later on because the self implemented Notification Service uses the standardised IDL of the CORBA Notification Service.

Comparison of Centralised/Distributed Notification Service
Also for the Notification Service it has to be decided, whether one centralised Notification Service should be used or two instances on both sides. For the same reasons of using two Naming Service implementations, it has been decided to have two distributed Notification Services. It simplifies the tests within one domain.

The disadvantage is the double cost of implementation. On the other hand only by using a Notification Service in each domain it is possible to isolate software faults due to CORBA-DCOM interworking.

During the development it will be evaluated whether it is necessary to have a CORBA Notification Service implementation on CSELT side or if it is enough to have only a DCOM "Notification Service".

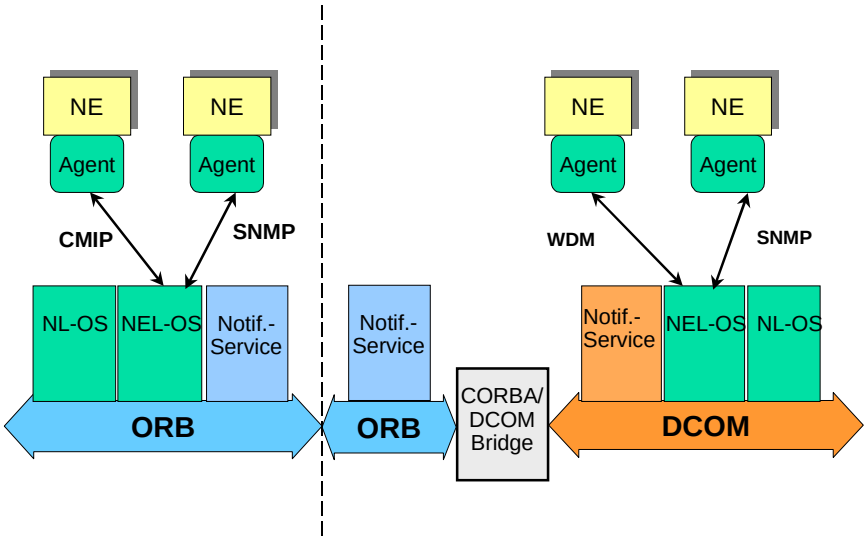

Figure 20: CORBA and DCOM Notification Service on CSELT side

5 Architecture of the LION Subnetwork Management Systems

Up to now only the overall logical and physical architecture has been described, i.e. the functional components interoperating via the NL-IF have been identified and location and layout of the NL-IF has been defined. However, the implementation of the functionality has to be done in the sub-management systems of CSELT and T-Nova. This is to be done mainly in the NEL-OSs, which implement most of the functionality specified in the NL-IM. Thus, the questions answered in the present chapter are concerned with the programming language and tools used. Also it has to be considered how access to the network elements can be realised. Based on these considerations a more detailed (preliminary) software architecture, for the CSELT and T-Nova sub-systems has been developed. Especially the architecture of the sub-NMSs has to be revised as soon as first implementation experiences are available.

5.1 Corba Standard Version and Products Used

To exclude some of the possible causes for interworking problems, the LION partners had to decide which CORBA version should be used. This has been done under the aspect of available CORBA products.

The LION Network Level Interface (NL-IF) is based on CORBA standard 2.1 because of the following reasons:

- CORBA 2.3 products are not stable
- available DCOM/Corba bridges (i.e. IONA COMet) support only CORBA 2.1

The CORBA products used in LION are reported in Table 4.

Products	CSELT	T-Nova
ORB	IONA Orbix 3.0.1	SUN jdk 1.2 ORB
DCOM/CORBA Bridge	IONA COMet	-
IDL Compiler	IONA Orbix 3.0.1	SUN IDL to Java Compiler
Programming Tool	Microsoft Visual C++ 6.0	IBM VisualAge 3.5 for Java2.0

Table 4: CORBA products used at CSELT and T-Nova

5.2 CSELT NMS

5.2.1 Architecture

The client-server architecture of CSELT NMS is shown in Figure 21.

The client application comprises the GUI, written in Visual J++.

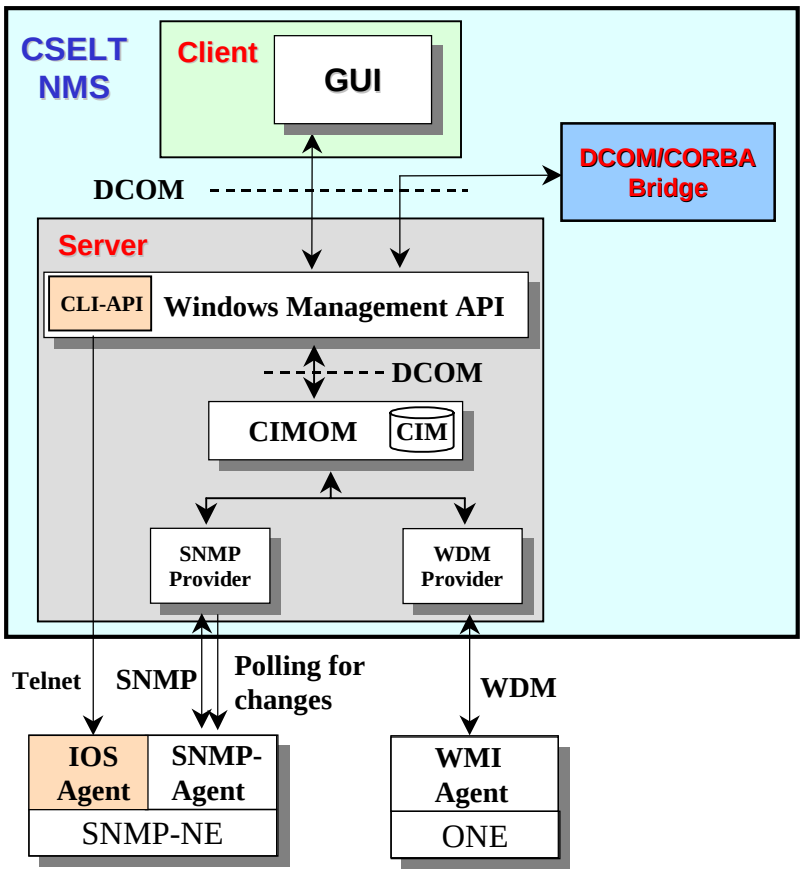

Figure 21: CSELT NMS Architecture

The server application consists of:

- the manager entity, that is the management logic (Windows Management API), written in Visual C++. It is made up of some management modules; that is a CLI-API dedicated to configure the IP Nodes, the NEL-OS represents the specific code's module for the entire sub-network within CSELT domain and, finally, other modules designed in order to make available a CORBA open-API towards an external NMS (i.e. T-NOVA system). For more details see section 5.2.1.1.

- the Agent/Manager[7] entity, that is CIMOM and two built-in providers, called WDM and SNMP providers.

WDM provider

The WDM provider is a group of standard WMI providers, which consists of Class provider, Instance provider, Method provider and Event provider. The first one has not been implemented, while the other ones have been implemented using Visual C++.

SNMP provider

The SNMP provider allows client applications to access static and dynamic SNMP information through WMI. There are two SNMP data providers that allow applications to access and modify the data relating to SNMP devices and two SNMP event providers that generate WMI events from SNMP traps and notifications. The data providers are class and instance providers. The two event providers report the same types of events, but they report them in a different format. The event providers support separate sets of extrinsic event classes to represent the event information. Consumers choose a format by registering for a particular class of events. The two event providers are called the SNMP Encapsulated Event Provider (SEEP) and the SNMP Referent Event Provider (SREP).

5.2.1.1 CORBA open-API Design

The CSELT NMS should also be able to provide a northbound CORBA open-API towards other OSs, in the case of LION towards the T-Nova NMS. Figure 22 illustrates the architectural design of this interface.

To make feasible that, the DCOM framework's management logic will be grown up of two modules, so-called MB Encoder/Decoder and northbound DCOM API. Besides, OrbixCOMet and Orbix 3.0.1 will be used for the translation between northbound DCOM/CORBA APIs.

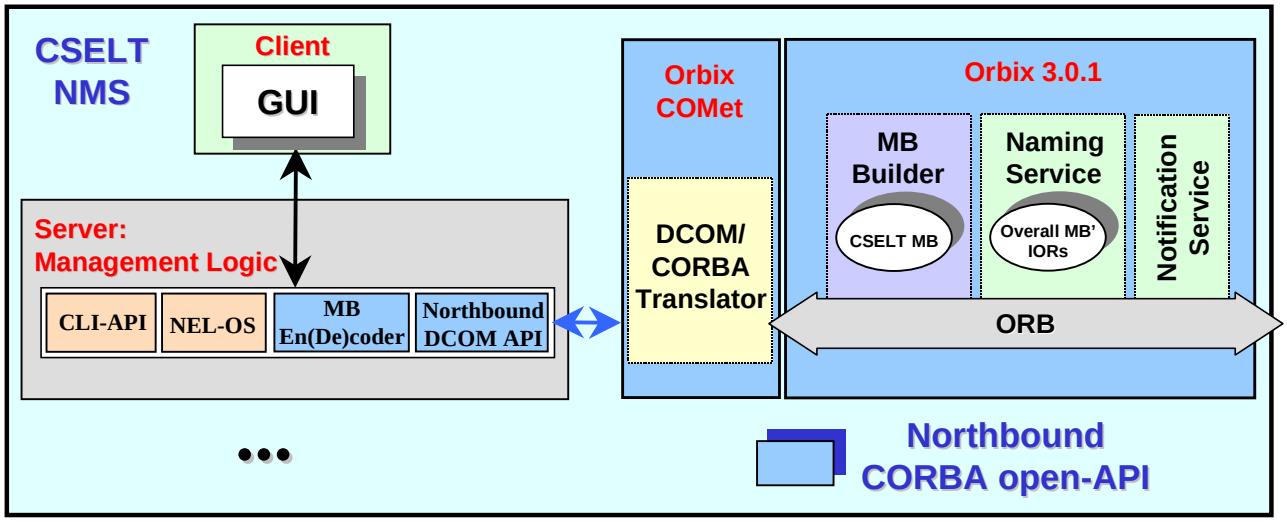

Figure 22: CORBA open-API Architecture

MB Encoder/Decoder:

[7] It is a dual-role entity, in fact it presents an agent interface to managers (client application and/or OSS), and a manager interface to agents (NEs).

This module allows to encode/decode the DCOM MB objects, which represent a "view" of the external NMS's MB objects, in this case T-Nova MB objects. This module should be designed taking into account that the client application should remain unchanged. Its implementation also depends on how the external NMS makes these MB objects visible to the CSELT NMS.

Northbound DCOM API:

At the DCOM layer, the northbound DCOM API represents the open-API towards external OSs. Its implementation should not depend on the external system; but it should be designed in order to export DCOM objects, with assigned attributes and operations, according to which management service profiles are allowed through this interface.

Obviously, this interface must also be designed according to the NL-IM specification.

DCOM/CORBA Translator:

This module is necessary for the automatic translation (mapping) functions:

- from DCOM objects (available on the DCOM API) to CORBA objects, and vice versa;
- from external NMS's MB objects (in this case T-NOVA MB objects) to DCOM MB objects, and vice versa.

For more details see section 5.2.1.2.

MB Builder:

The task of the MB builder module is to build the MB objects that must be associated to the CORBA objects (the 2nd class objects shown in Figure 16), which represent the "view" of the DCOM objects.

Obviously, this module must be designed according to the NL-IM specification and which management service profiles are allowed for the CORBA interface. Its implementation will be done using Orbix 3.0.1 product.

CORBA Services:

The CORBA interface will be able to provide two services:

- naming service, which must contain all the MB IORs;
- notification service, without using a specific commercial product.

Obviously, both these services must be designed according to the decision described in sections 4.2.2.1 and 4.2.2.2, that is to use two Naming Services and two Notification Services, one for each management domain respectively.

5.2.1.2 DCOM/CORBA Bridge: OrbixCOMet Product

The CORBA/DCOM bridge will be implemented by the OrbixCOMet product, which is a bi-directional bridge that integrates these two technologies, enabling VB (Visual Basic) and PowerBuilder applications to integrate with CORBA distributed applications. It enables COM/Automation[8] clients to communicate with CORBA servers with support for callbacks, and vice versa. Besides, it is CORBA 2.1 compliant.

[8] *Automation* is a sub-set of COM for issuing commands to COM components and thus controlling applications or components.

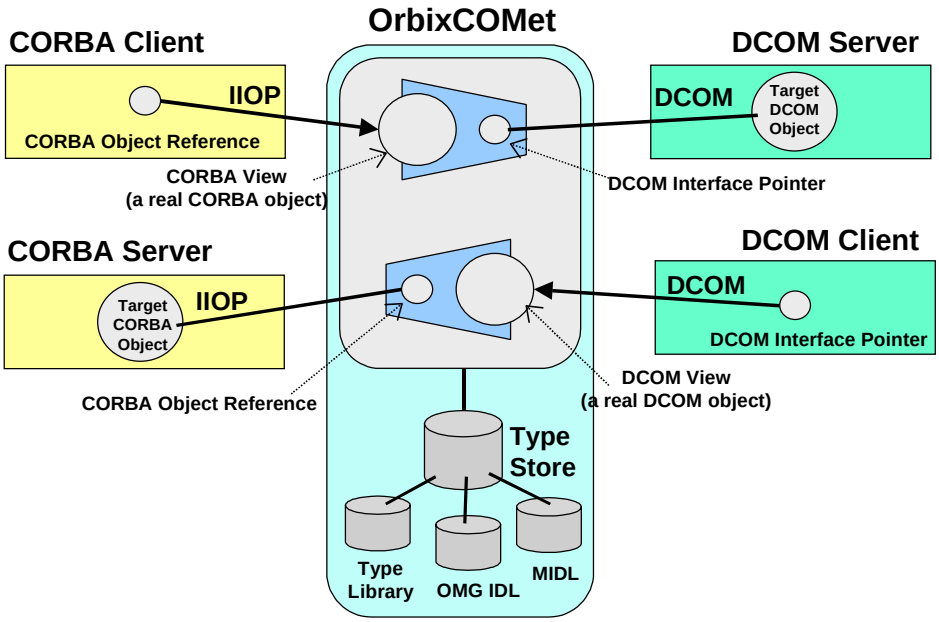

Figure 23: Typical Usage Models of OrbixCOMet

Orbix COMet is a completely dynamic bridge[9] with no requirement for C++ code generation (i.e. a Visual Basic developer can build the applications within its own environment, without the need for a C++ compiler to handle data type translation). The dynamic bridge will simply discover changes at runtime.

Since the OrbixCOMet supports both the DCOM protocol and the CORBA protocol (precisely: IIOP), it can be deployed on:

- every DCOM client application host (one-way and IIOP on the wire);
- every DCOM server application host (two-way, DCOM and IIOP on the wire);
- every central bridge host (two-way, DCOM and IIOP on the wire).

Figure 23 shows the typical usage models of OrbixCOMet, excluding those related to the Automation Client and Server.

The Type Store stores meta type information, which is a low level data type translation information, basically a superset of the common information of CORBA IDL and Microsoft IDL, so that full two-way bridging is possible. OrbixCOMet comprises a set of utilities that allow to store and retrieve information from the type store.

These two interworking models are described below:

- **Invoking on CORBA Object from DCOM:** when the DCOM client application asks OrbixCOMet for a “view” of a remote object of a given interface, OrbixCOMet looks in its local Type Store. If it has not seen the interface before, then there will be no information stored locally and then it will automatically query the CORBA Interface Repository to find out which operations (methods) and attributes (properties) the interface supports. Knowing this information, it can then create the DCOM “view” object, which supports a DCOM interface with DCOM methods and properties for each CORBA operations and attribute. The DCOM client

[9] One of the main benefits of the dynamic bridge is that the applications don’t need specific code to handle different user defined data types. This means that there is no code, which needs to be rebuilt when changes are made to the application or when the application is upgraded.

can then invoke on this DCOM object and all the invocations will automatically be forwarded to the CORBA server object, and any return parameters returned, of course. The DCOM client can use the DCOM protocol to contact a CORBA server.

- **Invoking on DCOM Object from CORBA:** when the CORBA client application can use the CORBA IIOP to contact OrbixCOMet. It can then connect to, and invoke methods on, a "CORBA view object" in the bridge via a CORBA object reference. The bridge makes a corresponding operation call on the target object in the DCOM server via a DCOM interface pointer. When the bridge receives this DCOM object reference, it checks to see if it has already created a CORBA view of that object, and if not, will dynamically generate one. Then any subsequent invocations from a CORBA application will be automatically forwarded on to that DCOM object.

The bridge will be installed on the DCOM server (running on Windows NT) host and its usage models[10] will be both

- DCOM client to CORBA server;
- and CORBA client to DCOM server.

5.2.2 Functionality

The management functionality of CSELT NMS comprises both Network Creation and Service Activation. Network creation is mainly done during initialisation, i.e. the MIB is initialised by creating the logical objects representing physical and logical resources. Service activation is concerned with the realisation of the services offered by the OTN, e.g. creation of a lightpath.

The first one is made up of domains, considered technology-independent, OTN and IP nodes, section trails; that is:

- OTS (Optical Transmission Section) and OMS (Optical Multiplexing Section) trails, lying both between OTN nodes;
- PS (Physical Section) trails, lying between OTN and IP nodes;
- IPS (IP Physical Section) trails, lying between IP nodes.

The service activation is made up of service trails, that is OCh (Optical Channel) and DCh (Digital Channel) trails, both lying on optical network layer.

Besides, other management functionality should be developed in order to manage the new OTN and IP functions (i.e. Digital Wrapper, UNI/NNI, etc.), IP Node Configuration, IP Policy-based Configuration & Monitoring and Service Assurance (i.e. some functions of alarms localisation and correlation).

5.3 T-NOVA NMS

5.3.1 Architecture

The architecture of the T-Nova network management system NMS is shown in Figure 24. The T-Nova NMS is divided into two layers, the Network Layer and the Network Element Layer. These layers are connected via an Object Request Broker.

[10] The implementation does not need to include the models concerning the Automation Client and Server (consequently the Automation interface pointer, called IDispatch pointer).

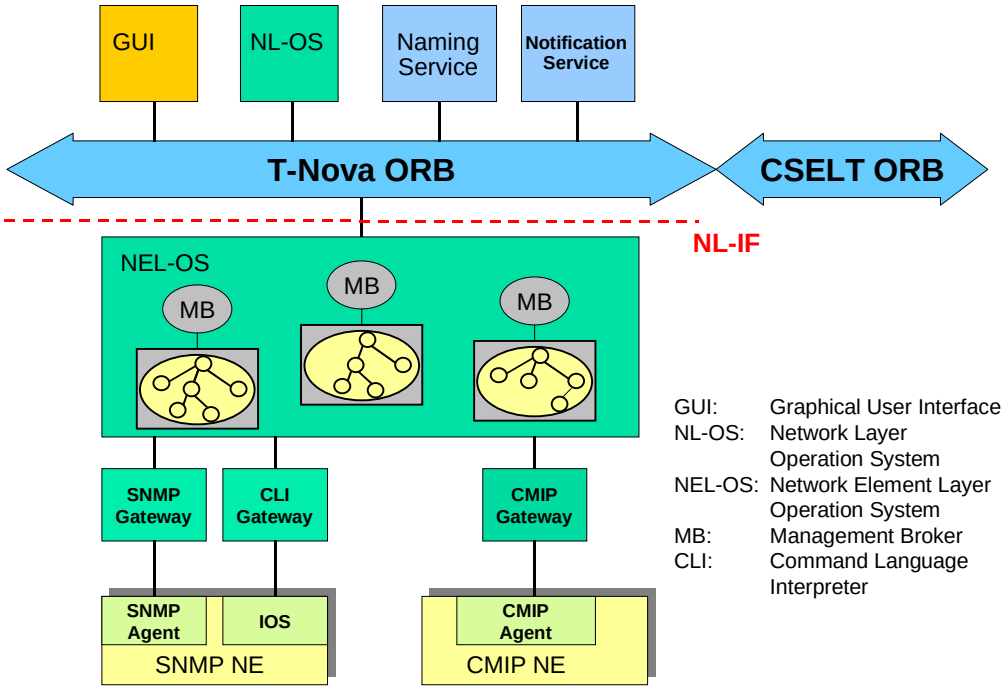

Figure 24: Architecture of the T-Nova Network Management System

Network Layer

The network layer consists of the actual operation system (NL-OS) and its GUI and the Corba Object Services, which are the Naming Service and the Notification Service.

Network Element Layer

The Network Element Layer OS and Gateway Modules (SNMP, CLI, CMIP) are parts of the NEL. The gateway modules are connected to the related Agents of the Network Element.

5.3.2 Functionality

5.3.2.1 Network Layer

The network layer is the highest layer of the NMS and provides the interface to the human operator (GUI). To fulfil the management requirements (configuration, alarm handling, traffic engineering etc.) the NL-OS uses the Corba Notification Service and the functionality services of the subordinated layer.

GUI:

The interface to the human operator displays the information of the NL-OS in a convenient way (Map, Alarm List, Configuration Tables). The API to the NL-OS will be defined in CORBA IDL (see Figure 24) or will be a Java API.

Platform	Windows NT 4.0
Programming Language	Java 2.0

NL-OS:

The NL-OS module implements the operations to fulfil the management requirements like provisioning of the overall network view, alarm handling, traffic engineering. The NL-OS contains the logical objects which represent the state of the network and is connected to two NEL domains (T-Nova, CSELT).

Platform	Windows NT 4.0
Programming Language	Java 2.0

Naming Service:

The Naming Service implementation makes it possible to find/locate the first level logical NEL objects (MBs) which are able to guarantee a needed functionality. A self-implemented naming service based on the OMG IDL is available at T-Nova.

Platform	Windows NT 4.0
Programming Language	Java 2.0

Notification Service:

The Notification Service implementation deals with notifications which are sent from the NEL to the NL. T-Nova is going to implement a small notification service based on the OMG IDL definition.

Platform	Windows NT 4.0
Programming Language	Java 2.0

5.3.2.2 Network Element Layer

NEL-OS:

The NEL-OS implements the network creation functionality, i.e. initialising the logical resources representing the T-Nova subnetwork and contained network resources.

It represents the state of all network elements in the T-Nova subnetwork domain. The NEL-OS gets the necessary information via the gateway modules and is able to offer this information or a subset of it to the NL-OS. The state of the subnetwork is offered by the logical objects. The NL-OS accesses these objects via Naming Service and the Management Broker (MB, first level objects, see section 3.2.2.3)

The NEL collects all alarm of the connected NE's and transport them after processing to the NL-OS (e.g. alarm correlation).

Platform	Windows NT 4.0
Programming Language	Java 2.0

SNMP Gateway:

The SNMP gateway translates the management operations into SNMP PDU's and vice versa. It receives SNMP traps and maps them into Java operations/notifications.

Platform	Windows NT 4.0
Programming Language	Java 2.0

CLI Gateway:

The CLI gateway implementation translates the management operations into Cisco CLI commands and sends them via telnet to the NE. In the opposite direction it translates the Cisco CLI responds into Java operations.

Platform	Windows NT 4.0
Programming Language	Java 2.0

CMIP Gateway:

The CMIP gateway translates the management operations into CMIP PDUs and vice versa. It also manages the connections and communication between TMN manager and agents (OSI communication), and sends and receives CMIP Protocol Data Units.

Platform	HP UX Vertel TMN++
Programming Language	C++