PROJECTE FI DE CARRERA: FULL DE CONFORMITAT PER A LA LECTURA

<table>
<thead>
<tr>
<th>Dades de l'estudiant</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGNOMS:</td>
</tr>
<tr>
<td>NOM:</td>
</tr>
<tr>
<td>TITULACIÓ:</td>
</tr>
<tr>
<td>ESPECIALITAT (SI ESCAU):</td>
</tr>
<tr>
<td>PLA D'ESTUDIS:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dades del Projecte de Fi de Carrera:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Títol:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DIRECTOR/A/SES:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PONENT (si escau):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data, hora i aula previstes per a la lectura:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IMPORTANT: Si la lectura ia las en el B3, has de tenir la conformitat de l'Àrea de Recursos)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conformitat dels membres del tribunal per a la lectura:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL/LA VOCAL</td>
</tr>
<tr>
<td>EL/LA PRESIDENT</td>
</tr>
<tr>
<td>EL/LA SECRETARIA/A</td>
</tr>
<tr>
<td>Sr/sa: ..................................................</td>
</tr>
<tr>
<td>Sr/sa: ..................................................</td>
</tr>
<tr>
<td>Sr/sa: ..................................................</td>
</tr>
</tbody>
</table>
Development of a Network Management System for the EU DICONET Project

Alumne:
Albert Pagès Cruz

Directors:
Davide Careglio
Jordi Perelló Muntan

ETSETB Curs 2009/2010
Acknowledgments:

Thank my project advisors Davide Careglio and Jordi Perelló for giving me the opportunity to make my project within the framework of the DICONET project. I think that fact helped me to understand how to work within a project of this magnitude. Also, thank them for all the corrections and observations necessary for this project to go forward.

Especially thank all the aid received by my companions from the DICONET project: Fernando Agraz and Marc Fontfreixa. Thank you for your help and all the patience you had.

Finally, mention the support received by my family and friends. I regret the long, possibly boring, explanations about the project every time the subject was mentioned. To them and the rest, thanks for everything!
## TABLE OF CONTENTS

### INTRODUCTION 5

1.1 OBJECTIVES OF THE FINAL PROJECT 7

### FROM OPAQUE TO ALL-OPTICAL NETWORKS 9

2.1 EVOLUTION OF OPTICAL NETWORKS 9

2.2 THE ASON ARCHITECTURE 12

2.2.1 THE ASON PLANES 12

2.2.2 ASON INTERFACES 13

2.2.3 THE OCC ARCHITECTURE 14

2.2.4 CONNECTION SERVICES 16

2.3 GENERALIZED MULTIPROTOCOL LABEL SWITCHING 17

2.3.1 SUMMARY OF GMPLS PROTOCOLS 19

### PHYSICAL LAYER DEGRADATIONS IN ALL-OPTICAL NETWORKS 21

3.1 OVERVIEW OF PHYSICAL LAYER IMPAIRMENTS 22

3.1.1 LINEAR IMPAIRMENTS 22

3.1.1.1 POWER LOSS 22

3.1.1.2 CHROMATIC DISPERSION (CD) 23

3.1.1.3 POLARIZATION MODE DISPERSION (PMD) 24

3.1.1.4 POLARIZATION DEPENDENT LOSS (PDL) 24

3.1.1.5 AMPLIFIER SPONTANEOUS EMISSION (ASE) NOISE 25

3.1.1.6 CROSSTALK (XT) 25

3.1.1.7 FILTER CONCATENATION (FC) AND AMPLIFIER TILT EFFECTS 26

3.1.2 NON-LINEAR IMPAIRMENTS 26

3.1.2.1 SELF-PHASE MODULATION (SPM) 26

3.1.2.2 CROSS-PHASE MODULATION (XPM) 27

3.1.2.3 FOUR WAVE MIXING (FWM) 28

3.1.2.4 STIMULATED BRILLOUIN SCATTERING (SBS) 28

3.1.2.5 STIMULATED RAMAN SCATTERING (SRS) 29

### THE EU DICONET PROJECT 31

4.1 TRANSPARENT OPTICAL NETWORK CHALLENGES 32

4.2 THE DICONET SOLUTION 33

4.2.1 PHYSICAL LAYER MODELING AND MONITORING 34

4.2.2 IMPAIRMENT AWARE LIGHT-PATH ROUTING 34

4.2.3 FAILURE LOCALIZATION 35
Due to the growth of the Internet during the last two decades and also due to the extended use of Virtual Private Networks (VPNs), data traffic demands have experienced a considerable growth. Lots of current applications such as Voice over IP (VoIP), multiconferencing or streaming need a certain quality of service that is no longer efficiently provided by nowadays IP model, mostly based on best-effort policies. Therefore, both core and access networks have to be re-designed to ensure efficient and reliable resource allocation. To this end, a four-layered network architecture was initially thought for the public network infrastructure, as depicted in Figure 1.1:

- An optical layer composed by fibres and other optical equipments that grant the access to the optical bandwidth. Dense Wavelength Division Multiplexing (DWDM) is the technology used in this layer to make good use of the existing bandwidth. DWDM allows the creation of various virtual fibres within the same physical optical fibre, where each one can carry up to various gigabits of traffic.

- Synchronous Digital Hierarchy (SDH) layer used to provide network synchronization, protection and management. Its mechanisms and architectures, namely, Automatic Protection Switching (APS), Unidirectional Path-Switched Ring (UPSR) and Bidirectional Line-Switched Ring (BLSR), allow sub 50 ms failure recovery times.

- Asynchronous Transfer Mode (ATM) layer used to provide an efficient usage of the available bandwidth and better quality of service than the one offered by traditional IP best-effort policies.

- Internet Protocol (IP) layer used due the large diversity of applications that is able to support. Also, it offers worldwide connectivity as, because of its popularity, it is being used for almost every traffic that flows through the network.
Although the use of this multi-layer architecture worked well during the first stages of the distributed optical networks, it could no longer cope with foreseen IP traffic growth. This architecture has its inherent problems such as high frame overhead caused by the transport of IP over ATM over SDH, poor bandwidth granularity due to the poor flexibility of SDH, poor transmission rate scalability due to ATM’s data rate limitations...Because of those reasons, and also because the limitations introduced by the Optical-Electrical-Optical (OEO) processing of the signal (that is why these kind of networks are also called opaque optical networks), many networks started to modify its architecture by flattening the protocol stack and migrating the backbone to All-Optical Networks (AONs) with optical routing and switching.

In fact, the actual tendency is to progress towards a two layer optical network (Figure 1.2): an all-optical DWDM transport network and an IP/Encapsulation Frame with Generalized Multiprotocol Label Switching (GMPLS) as the control plane. These networks, defined by the International Telecommunications Union (ITU) in G.872, have a control plane which directly configures the different transport plane resources. The Automatic Switched Optical Network (ASON) architecture, proposed by the same ITU, fits well in these specifications.

Although ASON aimed at deploying fully transparent AONs, the presence of degradations on the physical layer of the network may prevent the transparency to be achieved completely. The main reason is that, despite the automatic resource allocation that ASON can offer, it is only based on topological and traffic parameters and does not take into account the actual physical transmission in such connections. When establishing a Label Switched Path (LSP), that is, a connection end-to-end between two users, it can happen that the best route offered by ASON control plane becomes unfeasible given the current physical network state, because of the so called Physical Layer Impairments (PLIs), degradations that take place in the physical layer as a
consequence of being a non-ideal medium. The degradations may cause that, at a given point in
time, the usage of a particular wavelength for an LSP is not optimal at all because the presence
of the PLIs lowers the signal quality or increases the end-to-end bit error bit-rate. Therefore,
new routing algorithms that take into account the presence of the PLIs must be thought.

![Figure 1.2 – Protocol stack evolution](image)

The Information and Communication Technologies (ICT) European project DICONET
(Dynamic Impairment Constraint Networking for Transparent Mesh Optical Networks), started
in early 2008 and finishing on June 30 2010, is targeting a novel approach to optical networking
to provide ultra high speed end-to-end connectivity with quality of service and high reliability
through the use of optimised protocols and routing algorithms. These algorithms are being
developed considering the physical impairments that arise in transparent core networks as
constraints, to be incorporated into a novel dynamic network planning tool that would consider
dynamic traffic characteristics, varying physical impairment and component characteristics and
a reconfigurable optical layer. The use of this novel planning tool, in conjunction with proper
extensions to the control plane protocols in core optical networks that will be designed,
implemented and tested by the consortium will make possible to realize the vision of
transparency, while offering efficient resource utilization and strict quality of service guarantees
based on certain service level agreements.

1.1 OBJECTIVES OF THE FINAL PROJECT

The objective of the present final project has been the development of a graphical interface for
the management plane of the DICONET network. The management plane is the responsible for
having a whole network view and for establishing the communication with the control plane
when an establishment or deletion of an LSP is requested. The graphical interface will offer an
intuitive and neat tool in order to perform these desired operations.
The design and development of the interface entails the following aspects:

- **Creation and maintenance of the network resources database**: the database will offer the necessary persistence of the information associated to the network components, such as the *Optical Cross-Connects* (OXC), needed for the development of the interface.

- **Development of the graphical front-end**: the user’s graphical front-end will offer all the necessary functionalities for an appropriate management of the network and its components.

- **Establishment of the dialog with the control plane**: When the creation of an LSP is needed, the graphical interface will establish a communication with the control plane, responsible of the creation of LSPs, notifying the result to the user in a graphical way and updating the database with the new information.

- **Network Physical Parameters visualization**: the graphical front-end will allow the possibility to show the *Physical Parameters* (PP) information associated to the different network components.
Chapter 2 – From Opaque to All-Optical Networks

The first step towards a high bit-rate core network was the creation of electro-optical networks. In these kinds of networks, although the signal travelling through them was optical, all the processing in both ends of the connection and in the intermediate nodes was done in the electric domain. That is why they are also named opaque networks, that is, because the signal does not remain all the way from an end to the other in the optical domain; an OEO conversion must be done at some points to ensure a correct routing and Quality of Service (QoS).

However, despite the improvements introduced by opaque networks, the limitations of the electrical processing of the signal was an important bottleneck for achieving a high bit-rate core network, so that the next step was migrating the backbone towards AONs, where the routing and the processing was done in the optical domain. ASON is mainly the architecture used by these kinds of networks in conjunction with GMPLS as the technology used by the control plane.

In this chapter, the evolution from opaque to AONs will be explained and, in more depth, ASON as the leading architecture to AONs and GMPLS as the technology to implement their control plane.

2.1 EVOLUTION OF OPTICAL NETWORKS

This section illustrates the evolution from static opaque optical networks to the dynamic ASON architecture used nowadays (Figure 2.1).
The first generation of optical networks where based on static point-to-point optical links with only a wavelength channel supported per fibre around the third window (C band), where the attenuation introduced by the fibre was minimum [1]. To enhance the offered transmission capacity per fibre link, the usage of multiple wavelength channels transmitted in parallel over the same fibre was though, leading to the so called \textit{Wavelength Division Multiplexing} (WDM) technology.

The DWDM technology appeared as a variant of WDM, targeting at high capacity transmission systems. In this technology, up to 160 channels per fibre can be transmitted, with spacing between channels of 12.5 to 100 GHz.

The main characteristic of first-generation optical networks is that the transmission links are optical, whereas the end-to-end path not. The reason is that for any incoming data, the signal must be converted from the optical to the electrical domain in order to process the control information and switch the traffic accordingly. And, before transmitting the traffic through the output fibre, the outgoing traffic must be converted back from the electrical to the optical domain. The whole process increases substantially the cost of the network due the extra equipment needed and the complexity of the operations that must be done. Also, it becomes a limiting factor on network scalability, as electronic processing rates does not increase according to optical transmission ones.

These limitations posed the need of reducing processing complexities at intermediate nodes and, following this philosophy, the second-generation optical networks where born. In such networks the data is kept from source to destination in the optical domain by creating end-to-end optical paths (also named light-paths). The key element that made this transition possible has been the \textit{Optical Add and Drop Multiplexer} (OADM), used in ring topologies. In such an element, the local insertion/extraction of certain wavelengths is possible, while the remaining wavelengths pass through it optically (in a transparent way). By performing those operations, the amount of complex calculations done in intermediate nodes for bypassing traffic is reduced drastically, leading to a better scalability of the network.

The second-generation optical networks were mainly used to carry smooth voice-centred traffic, resulting to a multilayer network architecture as the one described in chapter 1: IP over ATM over SDH over DWDM. In this over-layered IP/ATM/SDH/DWDM architecture, each layer was being managed independently of the others. This results on overlapped functionalities between layers. Moreover, a high overhead was being introduced by the usage of ATM/SDH to transport IP, adding up to 25% of overhead. This is especially significant for short packets: the
statistics say that nearly 50% of IP packets are 40-44 bytes long [2]. For a packet of this length, 2 ATM cells will be needed, but the second one goes almost empty. Also, scalability beyond 2.4 Gbps is hard to obtain with this kind of architecture because it has to perform many operations when composing and decomposing the frames, dramatically dropping the latency.

In view of this, a lighter protocol stack enabling an efficient IP data traffic transport is strongly required. Many networks started to progress towards an IP/DWDM architecture where all the traffic engineering and network resilience functionalities provided by ATM and SDH layers are moved to the optical one.

Optical Transport Networks (OTNs) are seen as the first step in this migration. The OTN architecture, described by the ITU in G.872 [3], intends to be more general than SDH when handling different client signals, while introducing switching, multiplexing, management and resilience functionalities directly in the optical layer. Besides the ring topologies based on OADMs, OTNs can also be deployed describing meshed topologies, providing better resource utilization and the possibility of multiple resilience schemas. OXCs are the key devices in this migration, able to optically commute a signal coming from any incoming port to any outgoing port. However, if no wavelength converters are used, AONs should accomplish the wavelength continuity constraint (all the links that compose a route should use the same wavelength).

Despite being an interesting option for supporting traffic transport over DWDM physical layers, OTNs are still static networks, where all connection provisioning and release operations are done by a Network Management System (NMS) in a manual and centralized way. Therefore, rapid resource allocation or protection/restoration operations are not possible in this kind of scenario.

A control plane is introduced in OTNs, aiming to automatically provision and release optical connections over the transport plane, overcoming the limitations of manual provisioning schemas. This dynamic resource provisioning networks are a step forward in the evolution towards flexible optical networks.

Among all the dynamic optical network architectures, the leading one is ASON. ASON goal is the management of all-optical transport planes like OTN. In particular, ASON defines the control plane functional modules and related interfaces, enabling functionalities such as resource discovery, routing, signalling, call control, and connection protection and restoration.
The following section 2.2 explains more in depth ASON architecture and the section 2.3 explains how GMPLS is used for implementing ASON control plane.

### 2.2 THE ASON ARCHITECTURE

The ASON architecture is defined by ITU in the G.8080/Y.1304 [4]. Its aim is to automate the resource and connection management within the network. In an optical network without ASON, whenever a user requires more bandwidth, there is a request for a new connection. The service provider must then manually plan and configure the route in the network. This is not only time consuming, but also wastes bandwidth if the user sparingly uses the connection. Bandwidth is increasingly becoming a precious resource and expectations from future optical networks are that they should be able to efficiently handle resources as quickly as possible.

ASON architecture consists of various optical nodes, which could be OXCs, OADMs or Reconfigurable OADMs (ROADMs). These optical nodes are controlled by an Optical Connection Controller (OCC), which is the responsible for setting up, maintaining and tearing down connections. In addition, there is a NMS with a whole network view.

#### 2.2.1 THE ASON PLANES

The logical architecture of an ASON can be divided into three planes (Figure 2.2):

- **Transport Plane:** The transport plane contains the optical switches and other equipments responsible for transporting user data via connections. These switches are connected to each other via the Physical Interface (PI). Error detection operations are also performed within the transport plane.

- **Control Plane:** The control plane is responsible for the actual resource and connection management within an ASON network. It consists of a series of OCCs interconnected via Internal Network to Network Interfaces (I-NNIs). These OCCs have the following functions: network topology discovery, signalling, routing, address assignment, connection set-up/tear-down, connection protection/restoration, traffic engineering and wavelength assignment. The ASON finality is to create an intelligent control plane capable of setting up, maintaining and deleting connections, also capable of leading the recovery of failures detected in the transport plane.
Management Plane: The management plane is responsible for managing the control and transport planes, including failure, performance, accounting and security management functions. The management plane contains the NMS entity, which is connected to the OCCs in the control plane via the NMI-A (Network Management Interface for ASON Control Plane), and to the transport plane nodes via the NMI-T (Network Management Interface for the Transport Plane).

Mention that the traffic from users connected to an ASON network contains data for both transport and control Plane. The user is connected to the transport plane via a PI, while it communicates with the control plane via a UNI (User Network Interface).

![Figure 2.2 – The ASON architecture](image)

2.2.2 ASON INTERFACES

As seen in 2.2.1, ASON architecture defines a set of interfaces with interaction purposes between the different network elements. There are six different interfaces defined in the ASON architecture:

- **User Network Interface (UNI)**: The UNI is the interface between the control plane and the end user domain. It has to perform the following tasks:
  
  o Connection creation.
  o Connection tearing down.
  o Connection configuration.
  o Connection maintenance.
• **External Network to Network Interface (E-NNI):** The E-NNI interconnects two different networks or two different administrative domains within the same network. E-NNI has to support the different routing and signalling protocols that may implement the different domains or networks connected through it.

• **Internal Network to Network Interface (I-NNI):** The I-NNI interconnects the different OCCs within the same network. Its function is to exchange signalling and routing information between them (i.e., support the signalling and routing protocols implemented inside the network).

• **Connection Controller Interface (CCI):** The CCI is the interface between the control plane and the transport plane. It interconnects an optical node with the OCC responsible for its management. It is responsible for the cross connection creation and deletion, node maintenance and asynchronous event notification from the physical layer (optical nodes).

• **Network Management Interface for ASON Control Plane (NMI-A):** The NMI-A interconnects the management plane with the control plane. For instance, it is used to request soft-permanent connections.

• **Network Management Interface for the Transport Plane (NMI-T):** The NMI-T interconnects the management plane with the transport plane. In contrast to NMI-A, this interface is used when establishing permanent connections, where the NMS calculates itself the route from the origin to the destination and configures each optical node through the NMI-T interface.

### 2.2.3 THE OCC ARCHITECTURE

In order to provide an intelligent control plane capable of doing the tasks described in 2.2.1, the OCC is a key component. The ASON architecture divides it in different modules to separate its functionalities (Figure 2.3):

• **Routing Controller (RC):** The RC is the responsible for providing a valid route towards a destination to support a requested connection. That route may accomplish certain constraints, such as the avoidance of a determined node or even the use of the same lambda all along the links the connection transverses.
Call Controller (CallC): The CallC is used in switched connections. Its functions are to accept or refuse the connection request coming from the UNI. It is also used to manage connection releases.

Connection Controller (CC): The CC is responsible for the coordination among the CallC, RC and LRM for managing and supervising connection set-ups, releases and the modification of connection parameters for existing connections.

Protocol Controller (PC): The PC maps the different parameters into messages that will be send subsequently through the OCC interfaces.

Link Resource Manager (LRM): The LRM component is responsible for the management of the allocation and unallocation of connections, providing topology and status information. When the route is known, the CC asks the LRM if there are the necessary free resources to establish the connection.

Figure 2.3 – The OCC architecture
2.2.4 CONNECTION SERVICES

The control of connectivity is essential for the operation of a transport network. ASON offers three types of connections, each one intended to satisfy particular requirements depending on the expected service. The three types of connection are defined as explained below:

- **Permanent Connection**: These connections are performed by the management plane. The NMS calculates the route that the connection will have to follow and, without any control plane intervention, configures the different involved optical nodes through the NMI-T interface.

- **Switched Connection** (SC): Through the UNI, the client domain requests a service connection to the control plane using a dynamic protocol message exchange in the form of signalling messages. These messages flow across either the I-NNI or E-NNI within the control plane. If using switched connections, the CC module should be implemented in order to accept or refuse the connection requests.

- **Soft-Permanent Connection** (SPC): In this case, the service can be understood as a combination of a permanent connection and a SC; the network provides a permanent connection at the two edges and utilizes a switched connection within the network to connect the edges. When performing an SPC, once the NMS receives the connection request, it delegates the connection creation to the control plane by communicating it to the source node OCC (through the NMI-A interface). The NMS informs to that OCC the connection destination and also the QoS that has to provide.

  Once the CC receives the connection request from the NMS, it requests the RC a route. If there is any route to the destination that accomplishes the required constraints, the RC returns it to the CC. Then, the CC will ask the LRM if the required resources are available. If there are the required resources, the LRM will mark them as allocated and then, the CC, using a signalling protocol, will send the connection request to the following OCC.

Note that the most significant difference between the three methods above is the entity that sets up the connections in the network and the one requesting them.
2.3 GENERALIZED MULTIPROTOCOL LABEL SWITCHING

GMPLS developed by the Internet Engineering Task Force (IETF), is presented in [5] as the technology that will provide the control plane to the ASON architecture.

GMPLS differs from traditional Multiprotocol Label Switching (MPLS) in that it supports multiple types of switching, i.e., the addition of support for Time Division Multiplexing (TDM), lambda, and fibre (port) switching. The support for the additional types of switching has driven GMPLS to extend certain base functions of traditional MPLS and, in some cases, to add functionality. These changes and additions impact basic LSP properties: how labels are requested and communicated, the unidirectional nature of LSPs, how errors are propagated, and information provided for synchronizing the ingress and egress Label Switching Routers (LSRs). The MPLS architecture was defined to support the forwarding of data based on a label. In this architecture, LSRs were assumed to have a forwarding plane that is capable of (a) recognizing both packet or cell boundaries, and (b) being able to process either packet headers (for LSRs capable of recognizing packet boundaries) or cell headers (for LSRs capable of recognizing cell boundaries). The original MPLS architecture is here being extended to include LSRs whose forwarding plane recognizes neither packet nor cell boundaries, and therefore, cannot forward data based on the information carried in either packet or cell headers.

Specifically, such LSRs include devices where the switching decision is based on time slots, wavelengths, or physical ports.

So, the new set of LSRs, or more precisely interfaces on these LSRs, can be subdivided into the following classes:

- **Packet Switch Capable (PSC):** Interfaces that recognize packet boundaries and can forward data based on the content of the packet header. Examples include interfaces on routers that forward data based on the content of the IP header and interfaces on routers that switch data based on the content of the MPLS “shim” header.

- **Layer-2 Switch Capable (L2SC):** Interfaces that recognize frame/cell boundaries and can switch data based on the content of the frame/cell header. Examples include interfaces on Ethernet bridges that switch data based on the content of the Medium Access Control (MAC) header and interfaces on ATM-LSRs that forward data based on the ATM Virtual Path Identifier (VPI)/Virtual Channel Identifier (VCI).
• **Time-Division Multiplex Capable (TDM):** Interfaces that switch data based on the data's time slot in a repeating cycle. An example of such interfaces is SDH interfaces.

• **Lambda Switch Capable (LSC):** Interfaces that switch data based on the wavelength on which the data is received. An example of such an interface is that of a Photonic Cross-Connect (PXC) or OXC that can operate at the level of an individual wavelength.

• **Fibre Switch Capable (FSC):** Interfaces that switch data based on a position of the data in the (real world) physical spaces. An example of such an interface is that of a PXC or OXC that can operate at the level of a single or multiple fibres.

A circuit can be established only between, or through, interfaces of the same type. Depending on the particular technology being used for each interface, different circuit names can be used, e.g., SDH circuit, optical trail, light-path, etc. In the context of GMPLS, all these circuits are referenced by a common name: LSP.

The concept of nested LSP (LSP within LSP), already available in the traditional MPLS, facilitates building a forwarding hierarchy, i.e., a hierarchy of LSPs. This hierarchy of LSPs can occur on the same interface, or between different interfaces.

For example, a hierarchy can be built if an interface is capable of multiplexing several LSPs from the same technology (layer), e.g., a lower order SDH LSP nested in a higher order SDH LSP. Several levels of signal (LSP) nesting are defined in the SDH multiplexing hierarchy. The nesting can also occur between interface types. At the top of the hierarchy (see Figure 2.4) are FSC interfaces, followed by LSC interfaces, followed by TDM interfaces, followed by L2SC, and followed by PSC interfaces. This way, an LSP that starts and ends on a PSC interface can be nested (together with other LSPs) into an LSP that starts and ends on a L2SC interface. This LSP, in turn, can be nested (together with other LSPs) into an LSP that starts and ends on a TDM interface. In turn, this LSP can be nested (together with other LSPs) into an LSP that starts and ends on a LSC interface, which in turn can be nested (together with other LSPs) into an LSP that starts and ends on a FSC interface.
GMPLS introduces the TE link and data link concept. Basically, a TE link is an association of various data link. In this sense, a fibre could be seen as a TE link which transports various data links that, in fact, are the DWDM wavelengths transmitted over the fibre.

### 2.3.1 SUMMARY OF GMPLS PROTOCOLS

The evolution of MPLS into GMPLS has extended its signalling (RSVP-TE, Resource ReServation Protocol with Traffic Extensions, and CR-LDP, Constrained-based Routing Label Distribution Protocol) and its protocols (OSPF-TE, Open Shortest Path First with Traffic Extensions, and IS-IS-TE, Intermediate System to Intermediate System with Traffic Extensions). A new protocol, Link Management Protocol (LMP), has been introduced to manage and maintain the health of the control and data planes between two neighbouring nodes. A summary of the protocols and its extensions for GMPLS is explained below [6]:

- OSPF-TE, IS-IS-TE; Routing protocols for the auto-discovery of network topology, advertise resource availability (e.g., bandwidth or protection type). The major enhancements are as follows:
  - Advertising of link-protection type.
  - Implementing derived links (forwarding adjacency) for improved scalability.
  - Accepting and advertising links with no IP address - link ID.
  - Incoming and outgoing interface ID.
  - Route discovery for back-up that is different from the primary path (shared-risk link group).
**RSVP-TE, CR-LDP;** Signalling protocols for the establishment of traffic engineered LSPs. The major enhancements are as follows:

- Label exchange to include non-packet networks (generalized labels).
- Establishment of bidirectional LSPs.
- Signalling for the establishment of a back-up path (protection Information).
- Expediting label assignment via suggested label.
- Waveband switching support - set of contiguous wavelengths switched together.

**LMP;** Neighbour discovery protocol. Its performance is composed of four functionalities

- *Control Channel Management:* Established by negotiating link parameters (e.g., frequency in sending keep-alive messages) and ensuring the health of a link (hello protocol).
- *Link Connectivity Verification:* Ensures the physical connectivity of the link between the neighbouring nodes.
- *Link Property Correlation:* Identification of the link properties of the adjacent nodes (e.g., protection mechanism).
- *Fault Isolation:* Isolates a single or multiple faults in the optical domain.

The protocol stack is shown in Figure 2.5:
Chapter 3

PHYSICAL LAYER DEGRADATIONS IN ALL-OPTICAL NETWORKS

Despite the numerous advantages that DWDM-AONs offer, they are not exempt from certain limitations in capacity and quality of the signal. This chapter will focus in the limitations that are present in the physical layer of such networks.

In these networks, PLIs incurred by non-ideal optical transmission medium accumulate along an optical path, and determine the feasibility or transmission quality of the light-paths. If the received signal quality is not within the receiver sensitivity threshold, the receiver may not be able to correctly detect the optical signal, causing the light-path (and the corresponding reserved resources) to be useless.

Some PLIs are unique to transparent networks (such as the filter cascading and crosstalk, etc.) and others may be present in all kinds of optical networks (such as dispersion). The impairments incurred by non-ideal physical layer elements accumulate along the path as the optical signal progresses toward the destination. The overall effect of individual PLIs determines the feasibility of an optical path; the received Bit-Error Rate (BER) at the destination node might become unacceptably high. The BER is the number of received binary bits that have been altered due to noise and interference, divided by the total number of transferred bits during a studied time interval; the maximum nominal value of BER for an optical transmission to be feasible is $10^{-9}$. The lack of OEO conversion in transparent networks makes it necessary to consider impairment accumulation along the path.

The limitations are imposed mainly by:

- *Amplifier Spontaneous Emission* (ASE) noise generated by the *Erbium Doped Fibre Amplifiers* (EDFAs) placed within the optical network. This factor accumulates along the fibre link and can be present regardless if there is signal or not.
• Shifts in the central frequency of the optical channels due multiple reasons such fluctuations on ambient temperature or the aging of optical components.

• *Polarization Mode Dispersion* (PMD) of the signal that causes distortions and fadings.

• Nonlinearities of the optical fibres that lead to interactions between powers of multiple channels.

• Imperfections of the components used in the network.

• The effects of other signals over the desired one (crosstalk).

### 3.1 OVERVIEW OF PHYSICAL LAYER IMPAIRMENTS

In this section are described the main PLIs and their effects [7] [8]. The PLIs can be divided in linear and nonlinear.

#### 3.1.1 LINEAR IMPAIRMENTS

Linear impairments are intensity-independent and static in nature.

#### 3.1.1.1 POWER LOSS

Power loss can be defined as the optical loss that is accumulated from source to destination along fibre-links and is normally made up of intrinsic fibre losses and extrinsic bending losses. Intrinsic fibre losses are due to attenuation, absorption, reflections, refractions, Rayleigh scattering, optical component insertion losses, etc. The extrinsic losses are due to micro and macro bending losses.

Additional losses occur due to the combined effects of dispersion resulting from *Inter-Symbol Interference* (ISI), mode-partition noise, and laser chirp.
3.1.1.2 CHROMATIC DISPERSION (CD)

The degradation of an optical signal caused by the various spectral components travelling at their own different velocities is called dispersion. CD causes an optical pulse to broaden such that it spreads into the time slots of the other pulses. The concept to consider when talking about CD should be optical phase. It is important to mention the optical phase before any explanations of CD because of their mathematical relationship: CD is the second derivative of optical phase with respect to the optical frequency.

CD consists of both material dispersion and waveguide dispersion, as illustrated in Figure 3.1:

![Figure 3.1 – Dispersion in a standard single-mode fibre](image)

Both of these phenomena occur because all optical signals have a finite spectral width, and different spectral components will propagate at different speeds along the length of the fibre. One cause of this velocity difference is that the index of refraction of the fibre core is different for different wavelengths. This is called material dispersion and it is the dominant source of chromatic dispersion in single-mode fibres. Another cause of dispersion is that the cross sectional distribution of light within the fibre also changes for different wavelengths. Shorter wavelengths are more completely confined to the fibre core, while a larger portion of the optical power at longer wavelengths propagates in the cladding. Since the index of the core is greater than the index of the cladding, this difference in spatial distribution causes a change in propagation velocity.

CD can cause bit errors in digital communications or distortion and a higher noise floor in analog communications and can pose a serious issue in high bit-rate systems if it is not measured accurately and some form of dispersion compensation is not employed.
3.1.1.3 POLARIZATION MODE DISPERSION (PMD)

Along a fibre-span, the fibre can have some irregularities, such as being non-circular or contain impurities, leading to present obstacles to an optical pulse along its path. These obstacles cause different polarizations of the optical signal to travel with different group velocities resulting in pulse spread in the frequency domain, known as PMD. The difference in propagation time between polarizations is called *Differential Group Delay* (DGD) (see Figure 3.2) and is the unit used to describe PMD.

![Figure 3.2 – DGD in a non-ideal fibre](image-url)

PMD becomes a major limiting factor for WDM systems designed for longer distances at higher bit-rates. Because PMD effects are random and time-dependent, the most reliable and efficient PMD compensation technology is the use of adaptive optics to realign and correct the pulses of dispersed optical bits.

3.1.1.4 POLARIZATION DEPENDENT LOSS (PDL)

Due to irregularities in the fibre, the two polarization components along the two axes of a circular fibre suffer different rates of loss, thereby degrading signal quality in an uncontrolled and unpredictable manner and introducing fluctuations in *Optical Signal to Noise Ratio* (OSNR).

PDL mainly occurs in passive optical components. The most common passive optical components that exhibit PDL include couplers, isolators, multiplexers/demultiplexers, and photodetectors.
3.1.1.5 AMPLIFIER SPONTANEOUS EMISSION (ASE) NOISE

The primary source of additive noise in optically amplified systems is due to the ASE noise produced by the optical amplifiers used as intermediate repeaters and preamplifiers at the receiver-end. ASE noise is emitted by the amplifier in both forward and reverse directions, but only the forward ASE noise is a direct concern to system performance since that noise will co-propagate with the signal to the receiver where it degrades system performance.

Excess of ASE noise is an unwanted effect in lasers, since it dissipates some of the laser’s power. In optical amplifiers, ASE noise limits the achievable gain of the amplifier and increases its noise level. The ASE noise is very broadband and needs to be carefully analyzed to evaluate its degrading effect on system performance.

3.1.1.6 CROSSTALK (XT)

XT arises due to incomplete isolation of DWDM channels provoking the effect of signal power leakage from other DWDM channels on the desired channel. XT depends on the ratio of the optical powers of two channels and can be either incoherent (i.e., heterowavelength or out-of-band) or coherent (i.e., homowavelength or in-band). The main difference between incoherent and coherent XT is that coherent XT’s phase is correlated with the signal considered while in the incoherent case the phase is not correlated with the signal considered [9].

Figure 3.3 shows a common XT pattern inside an OXC:

Figure 3.3 – XT inside an N-port and M-wavelengths OXC
The computation of XT becomes quite complicated as the number of XT elements which the signal passes through increases, and should be considered in the design of DWDM networks. XT effects can be mitigated by the use of intelligent wavelength assignment techniques.

### 3.1.1.7 FILTER CONCATENATION (FC) AND AMPLIFIER TILT EFFECTS

The narrowing of spectral width of the signal as it traverses through a set of filters along a path is called FC effect. The penalty induced by FC depends on the route, the modulation type used, and the number of network elements the signal traverses before it reaches the destination.

Because of the spectral non-uniformity of amplifiers gain, as a result, different wavelengths of a DWDM signal are amplified non-uniformly, thus causing lambda-dependent effects known as amplifier tilt effects. The problem becomes quite severe in long-haul DWDM networks, where the signal undergoes amplification over a chain of in-line amplifiers. Tilt effects can be reduced with the application of gain-flattening techniques using optical filters that have gain profiles that are the inverse of the gain profiles of the amplifiers.

### 3.1.2 NON-LINEAR IMPAIRMENTS

Non-linear impairments are intensity-dependent and dynamic in nature.

#### 3.1.2.1 SELF-PHASE MODULATION (SPM)

The non-linear phase modulation of an optical pulse caused by its own intensity in an optical medium is called SPM. An ultra-short optical pulse, when travelling in a medium, will induce a time varying refractive index of the medium, i.e., the higher intensity portions of an optical pulse encounter a higher refractive index of the fibre compared with the lower intensity portions. This variation in refractive index will produce a phase shift in the pulse, leading to a change of the pulse's frequency spectrum.

As the effect is proportional to the transmitted signal power, the SPM effect is more pronounced in systems with high transmitted power. SPM also depends on the input pulse shape.
Figure 3.4 shows how a Gaussian pulse (top curve) propagating through a non-linear medium undergoes through a self-frequency shift (bottom curve) due to SPM. The front of the pulse is shifted to lower frequencies and the back to higher frequencies. In the centre of the pulse the frequency shift is approximately linear.

![Figure 3.4 – SPM of a Gaussian pulse](image)

### 3.1.2.2 CROSS-PHASE MODULATION (XPM)

XPM refers to the non-linear phase modulation of an optical pulse caused by intensity fluctuations of other optical pulses. The result of XPM may be asymmetric spectral broadening and distortion of the pulse shape.

XPM hinders the system performance through the same mechanism as SPM. XPM damages the system performance even more than SPM and influences it severely when the number of channels is large. The XPM-induced phase shift can occur only when two pulses overlap in time. Due to this overlap, the intensity-dependent phase shift and consequent chirping is enhanced, leading to enhanced pulse broadening. The effects of XPM can be reduced by increasing the wavelength spacing between individual channels.
3.1.2.3 FOUR WAVE MIXING (FWM)

FWM is an intermodulation distortion in optical systems, similar to the third-order intercept point in electrical systems: if three optical signals with carrier frequencies $\omega_1$, $\omega_2$ and $\omega_3$ co-propagate inside a fibre simultaneously, due the third order non-linear susceptibility in optical links, a fourth signal with frequency $\omega_4$ is generated, being $\omega_4$ related to the other frequencies by $\omega_4 = \omega_1 \pm \omega_2 \pm \omega_3$.

FWM is also present when only three components interact, generating the so-called Degenerate FWM, showing identical properties as in case of four interacting waves.

The FWM effect is independent of the bit-rate and is critically dependent on the channel spacing and fibre dispersion. Decreasing the channel spacing increases the four-wave mixing effect. FWM has severe effects in a DWDM system, which uses dispersion-shifted fibre. If there is some dispersion in the fibre, then the effect of FWM is reduced. This is why non-zero dispersion-shifted fibres are normally used in DWDM systems. Another way to reduce FWM effect is to employ unequal channel spacing in such a way that the generated signals do not interfere with the original signals.

3.1.2.4 STIMULATED BRILLOUIN SCATTERING (SBS)

Brillouin scattering occurs when light in a medium interacts with time dependent optical density variations, changing its energy and path. These density variations may be due to acoustic modes, such as phonons, magnetic modes, such as magnons, or temperature gradients. As described in classical physics, when the medium is compressed, its index of refraction changes and the light's path necessarily bends.

For intense beams (e.g. laser light) travelling in a medium such as an optical fibre, the variations in the electric field of the beam itself may produce acoustic vibrations in the medium via electrostriction. The beam may undergo Brillouin scattering from these vibrations, usually in opposite direction to the incoming beam, a phenomenon known as SBS.

In SBS, the scattering process is stimulated by photons with a wavelength higher than the wavelength of the incident signal. SBS is recognized as the most dominant fibre non-linear scattering effect.
SBS sets an upper limit on the amount of optical power that can be launched into an optical fibre. When input optical power exceeds the SBS threshold, a significant amount of the transmitted light is redirected back to the transmitter leading to saturation of optical power in the receiver, and introducing noise that degrades the BER performance. The SBS threshold depends on the line-width of the optical source, with narrow line-width sources having considerably lower SBS thresholds.

### 3.1.2.5 STIMULATED RAMAN SCATTERING (SRS)

Raman scattering is the inelastic scattering of a photon. When light is scattered from an atom or molecule, most photons are elastically scattered, such that the scattered photons have the same energy and wavelength as the incident photons. However, a small fraction of the scattered light is scattered inelastically by excitation, with the scattered photons having a frequency different from, and usually lower than, the frequency of the incident photons.

SRS, which is a combination of Raman scattering and stimulated emission, causes optical signal power from lower wavelength optical channels to be transferred to the higher wavelength optical channels. This can modify the power distribution among the DWDM channels, reducing the OSNR of the lower wavelength channels and introducing XT on the higher wavelength channels.

SRS occurs at significantly higher optical powers than SBS, with threshold powers in the order of watts for SRS compared to the milliwatts for SBS and, unlike SBS, SRS scatters in both forward and reverse directions.

By inserting filters appropriately into the transmission link, it is possible to effectively suppress the SRS power flow from the WDM channels to lower frequency noise. Furthermore, usage of a high-pass filter can enhance the SRS threshold in an optical fibre.
As mentioned in chapter 1, actual optical networks, such as ASON, only use topological and data flow information to perform the decisions to establish an LSP. This way of routing doesn’t take into account the physical implementation of the LSP so it can lead to improper routing decisions when the establishment of an LSP is requested. For example, it may be a good decision to establish an LSP from point A to B through point C if only topological/data flow information is being used, but when establishing physically the LSP this decision leads to poor results because the links have excessive attenuation in the wavelength used.

In order to avoid that, a new way of routing, taking into account the physical characteristics of the network, has to be developed. The DICONET project, an European project whose participants include 13 institutions and companies from 8 different countries (Research and Education Laboratory in Information Technologies, Research Academic Computer Technology Institute, from Greece; Institut TELECOM, Alcatel-Lucent France, JCP-Consult, from France; Interdisciplinair Instituut voor Breedband Technologie, from Belgium; University of Essex, from United Kingdom; ADVA AG Optical Networking, Huawei Technologies Deutschland GmbH, Deutsche Telekom AG, from Germany; Center of REsearch And Telecommunication Experimentations for NETworked communities, from Italy; Universitat Politècnica de Catalunya, from Spain; and ECI Telecom, from Israel) is targeting a novel approach to optical networking providing a disruptive solution for the development of the core network of the future, taking into account the physical structure of optical networks to perform the right decisions when the establishment of an LSP is requested.

The techniques and algorithms developed within the DICONET project are being validated using simulations and experiments, mainly using a small-scale test-bed mixing real and emulated all-optical nodes. In particular, simulations are carried on over two realistic topologies for which all necessary physical parameters are known: the “Deutsche Telekom” topology, a country-sized (diameter: 800 km) network of 14 nodes and 23 bidirectional links, and the “GEANT-2” topology, a continental-sized (diameter: 7000 km) network of 34 nodes and 52
bidirectional links [10]. Although the Deutsche Telekom topology can be used to simulate fully transparent networks, the GEANT-2 topology is too large to be fully transparent, and hence is used to validate algorithms specific to semi-transparent scenarios such as regenerator placement.

The components of the DICONET network can be seen in Figure 4.1:

![DICONET components diagram]

Figure 4.1 – DICONET components

Attached to the physical layer there are various monitors used to see the status of the network and the evolution of its performance and physical impairments. The information collected through these monitors is used by failure localization algorithms that work in conjunction with the network planning tool to find the most suitable route to establish an LSP taking into account the actual status of the network, physically and from a topological/data flow perspective. The results obtained through these two tools are collected by the control/management plane to make the final decisions.

### 4.1 TRANSPARENT OPTICAL NETWORK CHALLENGES

Optical transparency has an impact on network design, either by putting some limits on the size of DWDM transparent domains in order to neglect physical impact on Quality of Transmission (QoT), or by introducing physical considerations in the network planning process. The realization of dynamic and fully automated transparent optical core networks is an important task that is required in order to provide cost reduction and performance benefits. This goal has not yet been achieved in commercial exploitation due to: a) limited system reach and overall transparent optical network performance and b) difficulties related to the fault localization and isolation in transparent optical networks.
In transparent optical networks, as the signal propagates in a transparent way it experiences the impact of a variety of quality degrading phenomena that are introduced by different types of signal distortions, as reviewed in chapter 3. These impairments accumulate along the path and limit the system reach and the overall network performance. There are distortions of almost “deterministic” type related only to the pulse stream of a single channel, such as SPM, Group Velocity Dispersion (GVD) or the optical filtering. The other category includes degradations having a statistical nature such as ASE noise, WDM nonlinearities (FWM and XPM), PMD and XT.

In a transparent optical network, the impact of failures also propagates through the network and therefore cannot be easily localized and isolated. The huge amount of information transported in optical networks, makes rapid fault localization and isolation a crucial requirement for providing guaranteed quality of service and bounded unavailability times. The identification and location of failures in transparent optical networks is complex due to three factors: a) fault propagation, b) lack of digital information and c) large processing effort. The placement of monitoring equipment to reduce the number of redundant alarms and to lower the capital expenses, and the design of fast localization algorithms are among challenges of fault localization in transparent optical networks.

4.2 THE DICONET SOLUTION

As mentioned in [10] and [11], The vision of the DICONET project is that intelligence in core optical networks should not be limited to the functionalities that are positioned in the management and control plane of the network, but should be extended to the data plane on the optical layer.

The key innovation of DICONET is the development of a dynamic network planning tool residing in the core network nodes that incorporates real-time assessments of optical layer performance into routing algorithms and is integrated into a unified control plane. Several components should be considered to reach the desired functionality, which are discussed in the following points.
### 4.2.1 PHYSICAL LAYER MODELING AND MONITORING

To successfully develop the routing algorithms, physical impairments should be carefully identified and modelled. Physical layer impairments may be classified as linear and non-linear, as explained in chapter 3. Linear impairments are independent of the signal power and affect each of the optical channels individually while nonlinear effects scale with optical power levels and produce interdependencies of channels.

The important linear impairments that should be modelled and monitored are ASE noise, CD, XT, FC, and PMD. ASE noise degrades the OSNR. When uncompensated, CD limits the maximum transmission reach and channel bit-rate. XT is the general term given to the phenomenon by which signals from adjacent wavelengths leak and interfere with the signal in the actual wavelength channel. FC is produced by signal propagation through multiple DWDM filters between source and destination and results mainly in the narrowing down the overall filter pass-band. Finally, PMD manifests itself in a difference of propagation velocities between orthogonal polarizations, resulting in a broadening of the signal pulses.

There are two categories of nonlinear effects. The first arises due to the interaction of light waves with phonons in the silica medium. The two main effects in this category are SBS and SRS. The second set of nonlinear effects arises due to the dependence of the refractive index on the intensity of the applied electric field, which in turn is proportional to the square of the field amplitude. The effects in this category are SPM, XPM and FWM.

For routing algorithms it is very important to be able to accurately predict the performance of the propagating channel considering all the impairments that can degrade the signal quality along the propagation.

### 4.2.2 IMPAIRMENT AWARE LIGHT-PATH ROUTING

In transparent networks, data is transmitted over light-paths, the combination of a route and a wavelength. Because all-optical wavelength conversion is still experimental, once a signal is launched over a channel, it has to remain on the same channel (wavelength) from end-to-end, or from electrical regenerator to electrical regenerator in the case of semi-transparent networks. Such constraint, unique to all-optical networks, is called “wavelength continuity constraint”, also referred as *Routing and Wavelength Assignment* (RWA) problem.
In most RWA proposals the optical layer is considered as a perfect medium and therefore all outcomes of the RWA algorithms are considered valid and feasible even though the performance might be unacceptable. The incorporation of physical impairments in transparent optical network planning problems has recently received some attention from the research communities. We can classify impairment aware algorithms into two main categories: a) those that consider separately the RWA problem and the effects of impairments and b) those that solve the RWA problem including impairment constraints in the problem formulation. The DICONET project plans to examine the feasibility and applicability of algorithms belonging to the second case that consider jointly the RWA problem and the impairment constraints. The objective of the corresponding joint optimization problem would be not only to serve the connection requests using the available wavelengths, but also to minimize the total accumulated signal degradation on the selected light-paths.

4.2.3 FAILURE LOCALIZATION

By design, OXCs do not decode signals that traverse them and component failure can only be detected in an end-to-end, as opposed to local, fashion. This makes failure localization difficult in transparent networks. At the same time, failure recovery and localization is very important in networks where a single link can carry dozens of wavelengths modulated at 10-40Gbps each.

Single failure detection and recovery is a well-known topic in transparent optical networks, but the multiple failure case was far less studied. This problem was shown to be NP-complete and heuristics are needed. In the DICONET framework an algorithm that solves the multiple failures location problem in transparent optical networks is proposed where the failures are more damaging and affect longer distances. The proposed solution also covers the non-ideal scenario, where lost and/or false alarms may exist. Although the NP-complete complexity of the problem, the proposed algorithm keeps most of its complexity in a pre-computational phase. This algorithm locates the failures based on received alarms and the failure propagation properties, which differ with the type of failure and the kind of device that are in the network. Another algorithm has been proposed to correlate multiple security failures locally at any node and to discover their tracks through the network. To identify the origin and nature of the detected performance degradation, the algorithm requires up-to-date connection and monitoring information of any established light-path, on the input and output side of each node in the network.
This algorithm mainly runs a localization procedure, which will be initiated at the downstream node that first detects serious performance degradation at an arbitrary light-path on its output side. Once the origins of the detected failures have been localized, the network management system can then make accurate decisions to achieve finer grain recovery switching actions.

As can be seen, failure detection relies on the knowledge of the network's physical layer state, which is addressed in the following section through control plane mechanisms.

**4.2.4 CONTROL PLANE**

Monitors, nodes, network management system communicate through a so-called “control plane”, a set of protocols which makes interactions between all elements in the network. The control plane is ultimately responsible for light-path establishment, tearing down, rerouting, failure detection, dissemination of hardware monitoring information, and traffic engineering in general. Several protocols well-adapted to circuit-switching architectures at the wavelength granularity already exist; however, none of them incorporates physical layer characteristics. For this reason, the DICONET project proposes to extend well-known protocols to include optical layer characteristics in control planes. DICONET will evaluate and implement two distinct GMPLS-based control planes: a centralized control plane and a distributed control plane, both relying on the building blocks depicted in Figure 4.2:

![Figure 4.2 – Overview of control plane’s architecture](image)

The Impairment Aware-RWA (IA-RWA) module makes RWA decisions based on QTool computations. The QTool computes Q (quality) factors based on information contained in two databases: the Traffic Engineering Database (TED), which contains resource availability information, and the Physical Parameters Database (PPD), which contains impairment-related information.
In the centralized approach, all computations are done by a dedicated process called *Path Computing Element* (PCE) on a dedicated server. The PCE groups the IA-RWA and QTool blocks as well as the hardware acceleration module. The control plane learns monitoring information via extensions of the routing protocol (OSPF-TE) or with *Simple Network Management Protocol* (SNMP) queries. Standard RSVP-TE is used for the signalling.

In the distributed approach, an instance of the control plane runs on every node. A link-state routing protocol, OSPF-TE, can be used to disseminate the monitor information to all nodes. By design, information in the TED or PPD can be outdated as OSPF-TE takes a positive time to converge. Signalling is done by a modified version of RSVP-TE which is able to account for impairments in real time, as a light-path is being established. This is done to counter any stalled or outdated information in the local TED/PPD.

The DICONET control plane uses extended GMPLS to facilitate IA-RWA and fault localization which makes the software stack even more complex and computationally intensive compared to standard GMPLS implementations. Examples of these functions include impairment-aware forwarding and path selection and fault localization/detection, and in particular online physical layer impairment processing like the QTool. Therefore, to improve performance of the control plane, a hardware implementation of some of computationally intensive control protocol procedures can be envisioned. The main objective is to overcome the complexity of the control plane stack by implementing only time critical procedures of the DICONET control protocols in *Field Programmable Gate Arrays* (FPGAs) with embedded network processor in the form of a control protocol hardware accelerator. The hardware can potentially perform control protocol procedures up to 1000 times faster than the equivalent software based approach.

### 4.2.5 NETWORK PLANNING TOOL

The key innovation of DICONET is the development of a dynamic network planning tool residing in the core network nodes that incorporates real-time measurements of optical layer performance into IA-RWA algorithms and is integrated into a unified control plane. This tool will serve as an integrated framework that considers both physical layer parameters and networking aspects and will optimize automated connection provisioning in transparent optical networks.
The network planning tool has two operational modes: a) off-line mode and b) on-line (or real-time) mode. The off-line mode is selected in the planning phase of a network. In this phase a full map of network traffic and network conditions will be fed into the tool in order to produce the planning outcomes. The gained results can be disseminated to the network management system, controlled by an operator. For on-line use of the network planning tool an online traffic engineering solution is required utilizing an interface between the control plane and the management plane so that network situation could be evaluated in real time and its results could be periodically disseminated into the network.

4.3 ROLE INSIDE THE DICONET PROJECT

As mentioned before, this final project is included within the DICONET project. The purpose is to develop an NMS usable for the DICONET network. Looking back at chapter 2, especially the part referring to ASON architecture, the role of the developed NMS is clear: it has to provide a global view of the network in order to fully perform any management operation, check the state of any resource in the network and communicate with the control plane whenever a connection establishment or deletion is necessary.

The final goal is to provide a graphical tool capable of doing all the functionalities desirable in an NMS, all with a neat and intuitive environment, so that any person with sufficient knowledge about the topic and little training can easily understand it and use it.

Key points during the development of the NMS, aside the development of the graphical tool itself, are the communication with the GMPLS-enabled control plane (i.e., the NMI-A interface) and the obtaining and depiction of the values PP of the different network elements, the main distinctive point of the DICONET project. The following chapter will focus on the design and development process of the whole NMS tool.
As mentioned in chapter 1, the objective of the present final project is the development of a graphical interface for the NMS of the DICONET project. This interface will allow performing management operations, by a network administrator, such as establishing an LSP, check the status of the signal quality, add/delete interfaces or edit optical nodes’ configuration.

The graphical view will display the transport plane of the network, so the represented nodes will be OXCs, and will only show the core part of the network, so no client TE links will be represented. Indeed, the only information regarding the client side of the network that will be shown is the one corresponding to client-port information and client-PP information and will be through windows and menus, not in a graphical way (the information is not represented into the network map, it is represented through text).

5.1 APPLICATION REQUIREMENTS

Before starting to explain the development of the whole application, a key point in order to plan an efficient working method is to identify and analyze the requirements needed by the application. First of all, a list of the parts and agents involved in the project is shown below:

- **Transport Plane**: it is the physical plane were the optical signal travels through. It is composed of various OXCs, TE links, various electrical equipments (switches, routers...), ports (interfaces of the electrical equipments), and transponders (interfaces of the optical equipments). This plane will be the one represented on the graphical interface.

- **Control Plane**: connected to the transport plane in the way explained in chapter 3. For the present project, an emulated control plane will be used to develop the application and perform all the necessary tests.
Chapter 5 – Implementation of the NMS in the DICONET project

- **Management plane:** composed by the graphical interface. It shows a general view of the network and provides all the necessary tools to perform any management task.

### 5.1.1 APPLICATION FUNCTIONALITIES

The application is divided mainly in three parts: a database, a server-side and a client-side. The database and the server-side compose the server part of the application and the client-side composes the client part of the application. The role of each part is described as follows:

- The database stores the information associated to the components of the network (OXC, TE links, ports, data links, LSPs...) providing the necessarily persistence of this information for a proper operation of the whole application. This information will be accessed/edited only when necessary through the server-side in a transparent way so the user does not have to know when a database operation is being done. Moreover, the user can not access directly to the database; any kind of access to the database will be done by the server-side, reporting the result of the access to the client-side.

- The server-side is composed of various managers that receive any incoming request from the client-side, treat them and report the result to the client-side. Depending on the requested operation, an update of the database may be necessary. Also, the server-side is the responsible to talk with the control plane when the establishment/tearing-down of an LSP is requested.

- The client-side is composed by two parts: 1) a middleware part, that stores locally the information requested from the database in order to reduce the number of requests to it (only when strictly necessary a request to the database will be performed), stores the information associated to the position and status of the graphical elements and stores the information associated to the PP of the equipments; 2) a graphical front-end, that will be the user-side view of the application, composed by menus, windows and other necessary widgets to provide a full functional and understanding graphical interface.
A scheme of the described architecture is shown below (Figure 5.1):

In Figure 5.1 it is shown that the technology used for implementing the server-side is the **Enterprise Java Beans** (EJBs), an architecture specially designed for distributed client/server applications in Java programming language; for implementing the client-side, the **Standard Widget Toolkit** (SWT) utility from the Eclipse **Integrated Development Environment** (IDE) will be used; and, although it is not shown, for implementing the **DataBase** (DB) MySQL, a form of **Structured Query Language** (SQL), developed by Sun Microsystems, will be used. As mentioned before, an emulated control plane will be used to perform all the necessary tests along the whole developing process.

### 5.1.2 SPECIFIC REQUIREMENTS

#### 5.1.2.1 FUNCTIONAL REQUIREMENTS

The functional requirements define what and how operations are executed within the application: services that offers and the interaction with the user. A full compendium of the functional requirements is shown below.
FUNCTIONALITY (FU)

- Add/Delete OXC in the network map

FU_CF-1. Add/Delete OXC

The NMS network map will allow updating the network by adding/deleting an OXC.

- Ports

FU_CF-2. Configurable ports

The ports parameters can be modified through the NMS.

- Add/Delete TE link in the network map

FU_CF-3. Add/Delete TE link

The NMS network map will allow updating the network by adding/deleting a TE link.

- Topological structure

FU_CT-1. Physical parameters displaying

The elements of the network will allow showing their associated physical information, such as fibre type, dispersion, power, etc.

FU_CT-2. Modification of the names

The name of any network element will be modifiable without any restriction.

- Transmission rates

FU_TR-1. Rates

The NMS will allow managing LSPs of the following transmission rates:
• 1, 10 GbE.

• STM-1, 4, 16, 64, 256.

The Synchronous Transport Module level-1 [12] (STM-1) is the SDH ITU-T fibre optic network transmission standard. It has a bit rate of 155.52 Mbps. The other levels (4, 16, 64 and 256) are multiples of this basic transmission unit.

- LSP characteristics

<table>
<thead>
<tr>
<th>FU_TR-2. LSP parameters</th>
</tr>
</thead>
</table>

An LSP is characterized by the following parameters, which will be defined during the moment of its creation and will be monitored by the NMS:

• **Rate**: will show the transmission rate associated to the LSP.

• **Source, destination**: will show the source and the destination of an LSP.

• **LSP identifier**: a unique identifier, composed by a maximum of 16 characters, which will identify the LSP among all the others. It is provided by the control plane.

- LSP service states

<table>
<thead>
<tr>
<th>FU_TR-3. States definition</th>
</tr>
</thead>
</table>

• **Defined**: It is the first service state. This state only exists inside the NMS, where the end points are being reserved. No orders have been sent towards the control plane. In this state, the LSP is already planned (all the characteristics have been decided) but it doesn’t have a route assigned.

• **Implemented**: in this state, the LSP has been implemented into the optical network and a route has been assigned. The necessary resources will be reserved within the NMS.
- *Transition between LSP service states*

**FU_TR-4. Defined → Implemented**

In this transition, the NMS sends a request to the OCC responsible of the source OXC, with all the necessary information. After having established the LSP, the NMS will mark the used resources as reserved or allocated, making it impossible to use them for establishing another LSP.

**FU_TR-5. Defined, Implemented → Erased**

All the resources marked as reserved or allocated will be released.

**GRAPHICAL FUNCTION (FG)**

- *General functionality*

**FG_GE-1. Management functions**

All the functionalities described previously will be available through the graphical tool of the management system.

**FG_GE-2. Network map**

A map of the managed network will be shown through the graphical tool. It will be possible to perform management operations by selecting the elements in the map.

**FG_GE-3. Topological elements’ symbols**

Standard symbols will be used to represent the topological elements.

**FG_GE-4. Topological elements’ names**

The name of a resource, assigned during the creation of the network, will be displayed as a label next to the symbol that represents it on the map.
Chapter 5 – *Implementation of the NMS in the DICONET project*  

- **Windows**

  **FG_VE-1. Window refresh**

  All the windows will refresh automatically in a way that the last possible data will be available in the graphical tool.

  **FG_VE-2. Active window**

  The actual active window will be shown in first plane.

  **FG_VE-3. Window’s name**

  In all windows, into the upper side, it will appear the window’s name, so the user can be aware of the operation that is doing.

- **Events**

  **FG_AL-1. Event windows**

  The graphical tool will display through windows any kind of possible event, so the user can be aware of it.

- **Menus and toolbars**

  **FG_BM-1. Access to the functionality**

  All the functionality of the NMS will be available through toolbars and menus.

  **FG_BM-2. Unavailable functionalities**

  When a functionality is not available at a given moment, it will not appear in the menus and toolbars.
- Error messages

FG-ME-1. Messages to the user

The NMS will show to the user error, warning, etc. messages when it is not possible to perform an operation, an error has occurred, etc.

FG-ME-2. Clear messages

The generated messages will be clear, concise and self-explanatory.

5.1.2.2 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements are characteristics or properties that define how the system works.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>It must be a reasonable value. It is a critical point.</td>
</tr>
<tr>
<td>Security</td>
<td>Protection towards unwanted accesses.</td>
</tr>
<tr>
<td>Environment requirements</td>
<td>What is needed to run the application.</td>
</tr>
</tbody>
</table>

A description of the non-functional requirements is shown below:

- **Response time**: As every client/server application, the NMS must provide all the results within a reasonable time-span, being it not unnecessarily long. It is directly related with how the application is being developed.

- **Security**: To avoid unwanted accesses or data modifications, a good security policy is needed. It will depend on the system policy where the application will run, not on the application itself, so the point is not taken in consideration.

- **Environment requirements**: related to the equipment and software needed to run the application. For both client and server, a computer with Java installed and connection to the network will be needed. More specific requirements will be discussed in the corresponding sections.
5.2 IMPLEMENTATION OF THE NMS

This section will describe the whole process and all the parts involved during the implementation of the NMS. Although described before and displayed through Figure 5.1, a more detailed representation of the logical architecture of the NMS is shown below (Figure 5.2):

![Diagram showing the logical architecture of the NMS]

It can be seen that the server is composed by various EJB managers, as described before, and a module called trap manager which is the responsible to treat any incoming trap from the control plane via SNMP. The meaning of this action and how is done will be explained later. Also, the parts that compose the client are a middleware module and a SWT application module and work as explained above. The numbers represent the connection ports used to establish a dialog between all the parts.

The working and design details of each part will be explained during the following sections.
5.3 THE DATABASE

The main purpose of the DB is to store information related to the components of the network so it can be used by the application at any time. The reason of using a dedicated DB and not loading the information directly into the system memory is because this information has to persist over the time: the information life-cycle must be longer than the life-cycle of an application session, so it can be (and must be) used for future sessions. In this sense, the DB gives persistence to the information used by previous sessions and all the modifications done during the current session, because, at the end, this information represents the status of the network and all its components, physical and logical, so it has to be saved in a permanent way.

Already mentioned in 5.1.1, the technology used for implementing the DB is MySQL, a form of SQL. SQL [13] is a database computer language designed for managing data in Relational Database Management Systems (RDBMSs), and originally based upon relational algebra. Its scope includes data query and update, schema creation and modification, and data access control. The SQL language is sub-divided into several language elements, including:

- **Clauses**: constituent components of statements and queries.

- **Expressions**: can produce either scalar values or tables consisting of columns and rows of data.

- **Predicates**: specify conditions that can be evaluated and are used to limit the effects of statements and queries, or to change program flow.

- **Statements**: can have a persistent effect on schemas and data, or may control transactions, program flow, connections, sessions, or diagnostics.

- **Queries**: most common operations in SQL, used to retrieve data based on specific criteria using statements.

Figure 5.3 shows an schema of several elements that compose a single statement:
SQL commands can be divided into two main sublanguages. The *Data Definition Language* (DDL) contains the commands used to create and destroy databases and database objects. After the database structure is defined with DDL, database administrators and users can utilize the *Data Manipulation Language* (DML) to insert, retrieve and modify the data contained within it using SQL queries.

Among all the SQL-based languages, MySQL is the one that has been used. MySQL has the following distinctive characteristics [14]:

- Multiple storage engines, allowing the user to choose the one that is most effective for each table in the application:
  - Native storage engines.
  - Partner-developed storage engines.
  - Community-developed storage engines.
  - Custom storage engines.

- Commit grouping, gathering multiple transactions from multiple connections together to increase the number of commits per second.

Among all the storage engines that MySQL offers, the one used is InnoDB, a partner-developed storage engine whose owner is Oracle Corporation, distributed under both proprietary license and *General Public License* (GPL). InnoDB is transaction-safe and has commit, rollback, and crash-recovery capabilities to protect user data. Its row-level locking (without escalation to coarser granularity locks) capability increases multi-user concurrency and performance making InnoDB specially recommended for large DB sites which require high performance and processing large data volumes.
A very important aspect that motivates the choice of InnoDB and not another storage engine is that it supports Foreign Key (FK) referential-integrity constraints to maintain data integrity. The concept of FK, along with the concept of Primary Key (PK), is an important issue when dealing with the DB, so a detailed explanation is discussed below:

- The PK of a table within a DB is a special field inside the table that uniquely identifies an entry among all the others. Normally, it is an integer and has the property of auto-increment, so every time a new entry is inserted onto the table, the new PK for the entry will be the last PK used plus one. A PK must uniquely identify all possible rows that exist in a table and not only the currently existing rows and can never be the NULL value. Aside of the importance of identifying an entry from another, PKs are interesting when making an SQL query, because its nature makes them very suitable for being used within the predicate of a query.

- A FK is a referential constraint between two tables. The FK identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the PK or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the referenced table (except potentially NULL). This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

A table may have multiple foreign keys, and each foreign key can have a different referenced table. Each foreign key is enforced independently by the database system. Therefore, cascading relationships between tables can be established using foreign keys.

Because the RDBMS enforces referential constraints, it must ensure data integrity if rows in a referenced table are to be deleted (or updated). If dependent rows in referencing tables still exist, those references have to be considered.

Among 5 different referential actions that may take place in such occurrences, the Restrict referential action is the one that has been used. The Restrict referential action works as follows:
• A value cannot be updated when a row exists in a foreign key table that references the value in the referenced table.

• Similarly, a row cannot be deleted as long as there is a reference to it from a foreign key table.

By the usage of PKs and FKS, a map of the network can be inferred from the DB by the application, making it easier to represent and manage all its components.
5.3.1 STRUCTURE OF THE DATABASE

A diagram representing all the tables used inside the DB, along with its fields and its relationships, is shown in Figure 5.4.

The lines and numbers represent the relations between the entries of the tables: 1 represents the relation one-to, * represents the relation many-to, and 0..4 represents the relation up to four-to. The discontinuous line is used to represent the fact that some entries inside the same table associate between them to make an entry of another table.
The meaning of each table and its fields is explained below:

- **t_OCC**: represents the OCCs of the network.
  - id_occ: PK of the table.
  - ip_occ: represents the IP associated to the OCC. It must be a unique value among all the OCCs.
  - location: the physical place where the OCC is, for example, Berlin.
  - ccport: port of the CC of the OCC. It is used when trying to establish/tear-down an LSP.

- **t_OXC**: represents the OXCs of the network.
  - id_occ: FK referring to the PK of t_OCC. Every OXC in the network is controlled by a unique OCC.
  - ip_occ: represents the IP associated to the OXC. It must be a unique value among all the OXCs.
  - oxc_cfgport: configuration port of the OXC. It is used to establish/tear-down an LSP.
  - oxc_trapport: trap port of the OXC. It is used to receive/send traps from/to the OXC via SNMP.
  - location: the physical place where the OXC. It has the same role as the field with the same name in the t_OCC.
• **t_Port_LSC**: represents the port associated to the OXCs. Each port is a physical interface within an OXC responsible to modulate/demodulate the optical signal and to receive/transmit the signal. An OXC has a set of client-ports, used for the incoming/outgoing client data, and as much network-ports as connections towards other OXCs within the network it has (e.g. an OXC of degree three will have three network-ports associated to it).

  - id_port: PK of the table.
  - id_oxc: FK referring to the PK of t_OXC. Each port is associated to a unique OXC, but a single OXC can have multiple ports associated to it.
  - maxrbw: represents the maximum transmission bit-rate of the port.
  - minrbw: represents the minimum transmission bit-rate of the port.
  - availbw: represents the available bandwidth that the port has at a given moment. By default, and if no LSPs has been established, it will be equal to the maximum.
  - swtype: a coding number that represents the switching type used by the port. The switching type can be PSC-Level 1, PSC-Level 2, PSC-Level 3, PSC-Level 4, L2SC, TDM, LSC or FSC.
  - enctype: a coding number that represents the encoding type used by the port. The encoding type can be Packet, Ethernet, *Plesiochronous Digital Hierarchy* (PDH), SDH/*Synchronous Optical Network* (SONET), Digital Wrapper, Lambda, Fibre or Fibre-Channel.
  - alarms: the number of alarms associated to the port. By default is equal to 0. An alarm occurs every time a failure is detected inside the port and is reported to the NMS via SNMP. In the actual version of the NMS, this field is not used as a consequence of the version of the control plane used that doesn’t implement any kind of monitor to detect failures.
  - numWavelengths: the number of lambdas that the port is able to use.
Chapter 5 – *Implementation of the NMS in the DICONET project*

- bidir: binary value to indicate if the port is bidirectional or unidirectional.

- client: binary value to indicate if the port is a client-port or a network-port.

- portID: represents the local identifier within the OXC that is associated to. This identifier must be unique for a single OXC, but multiple ports belonging to various OXCs can have the same identifier.

- **t_TELink**: represents the TE links of the network. Two network-ports associate between them to make a network-TE link and only a client-port constitutes a client-TE link. A port can only be associated to a unique TE link.

  - id_telink: PK of the table.

  - id_port1: FK referring to the PK of t_Port_LSC. For a network-TE link, represents one of the ports the TE link is connected to, and for a client-TE link represents the only port the TE link is connected to.

  - id_port2: FK referring to the PK of t_Port_LSC. For a network-TE link, represents the other port the TE link is connected to, and for a client-TE link this field takes the value NULL.

  - bidir: indicates if the TE link is bidirectional or unidirectional.

  - client: indicates if the TE link is a client-TE link or a network-TE link.

  - maxrbw: represents the maximum transmission bit-rate transmitted over the TE link.

  - minrbw: represents the minimum transmission bit-rate transmitted over the TE link.

  - swtype: a coding number that represents the switching type used by the TE link.

  - enctype: a coding number that represents the encoding type used by the TE link.

  - numWavelengths: the number of lambdas transmitted over the TE link.
- **t_ConnectionPoint_LSC**: represents the connection points associated to each port of the network. A connection point is the logical representation of a wavelength inside a particular port. Each wavelength has two connection points that represent it: an input connection point and an output connection point.
  
  - id_cp: PK of the table.
  
  - id_port: FK referring to the PK of t_Port_LSC. Each connection point is associated to a unique port.
  
  - status: indicates if the connection point is allocated or unallocated.
  
  - type: indicates if the connection point is an input or an output connection point.
  
  - wavelength: represents the wavelength that the connection point belongs to inside a particular port.

- **t_DataLink_LSC**: represents the data links of the network. Each data link is contained inside a unique TE link, either client or network. A data link is associated to two connection points, representing the used wavelength and both source and destination, in the case of a network-data link, and only to a connection point if it is a client-data link.
  
  - id_dl: PK of the table.
  
  - id_cp1: FK referring to the PK of t_ConnectionPoint_LSC. For a network-data link, represents one of the connection points the data link is associated to, and for a client-data link represents the only connection point the data link is associated to.
  
  - id_cp2: FK referring to the PK of t_ConnectionPoint_LSC. For a network-data link, represents the other connection points the data link is associated to, and for a client-data link this field takes the value NULL.
  
  - id_telink: FK referring to the PK of t_TELink. Represents the TE link the data link is contained in.
Chapter 5 – Implementation of the NMS in the DICONET project

- **id_lsp**: FK referring to the PK of t_LSP. Represents the LSP the data link belongs to. A data link can only belong to a unique LSP. In the case that the data link doesn’t belong to any LSP (the data link is unallocated), the field takes the value NULL.

- **localIf**: represents the portID of the source port associated to the TE link that belongs to.

- **remoteIf**: represents the portID of the destination port associated to the TE link that belongs to. For a client-data link, this field is a blank space character.

- **status**: indicates if the data link is allocated or unallocated.

- **type**: indicates if the data link is an input or an output data link. For a data link, the type is referred from the source node.

- **wavelength**: represents the wavelength that the data link uses inside a particular TE link.

- **t_SubNetworkConnection**: represents the sub-networks connections established in the network. A sub-network connection is the logical representation of the cross-connection performed inside an OXC when an LSP is established. It is associated to up to four connection points which represent the ports and the wavelength used to establish the cross-connection, and to a unique LSP. If no LSPs have been established, then there aren’t any sub-network connections.

  - **id_snc**: PK of the table.

  - **id_cpi1**: FK referring to the PK of t_ConnectionPoint_LSC. Represents one of the input connection point used to create the cross-connection.

  - **id_cpo1**: FK referring to the PK of t_ConnectionPoint_LSC. Represents one of the output connection point used to create the cross-connection.

  - **id_cpi2**: FK referring to the PK of t_ConnectionPoint_LSC. Represents the other input connection point used to create the cross-connection.
o id_cpo1: FK referring to the PK of t_ConnectionPoint_LSC. Represents the other output connection point used to create the cross-connection.

o id_lsp: FK referring to the PK of t_LSP. Represents the LSP the sub-network connection belongs to.

o bidir: indicates if the cross-connection is bidirectional or unidirectional.

o usedBW: the bandwidth used for establishing the sub-network connection.

- **t_LSP**: represents the LSPs established within the network. An LSP is composed by multiple data links, multiple sub-network connections, a unique source OXC and a unique destination OXC. Regardless of having a unique source and destination, an OXC can be associated to multiple LSPs, either being source or destination of those LSPs.

  o id_lsp: PK of the table.

  o oxc_or: FK referring to the PK of t_OXC. It represents the source of the LSP.

  o oxc_dest: FK referring to the PK of t_OXC. It represents the destination of the LSP.

  o lspID: an identifier, as explained in 5.1.2.1, which uniquely represents an LSP among all the others. It is provided by the control plane.

  o bitrate: the transmission bit-rate of the LSP.

### 5.3.2 DATABASE OPERATIONS

Mainly, only four operations will be performed onto the DB once has been created. As mentioned before, the server-side of the application will be the part responsible of doing all the requests to the DB and reporting the result, if any, to the client-side. The operations and their flows are explained below:

- **Consultation operation**: executed when is necessary to retrieve some information from the DB, using the SELECT SQL query (see Figure 5.5). This operation may take place as a result of an internal process of the server or as a client request.
• **Update operation**: executed when an entry of the DB needs to be updated with new information, using the UPDATE SQL query (see Figure 5.6). This operation may take place as a result of an internal process of the server or as a client request.

• **Insert operation**: executed when a new entry has to be inserted into the DB, using the INSERT SQL query (see Figure 5.7). This operation may take place as a result of an internal process of the server or as a client request.
• *Delete operation*: executed when a DB entry need to be removed from it, using the DELETE SQL query (see Figure 5.8). This operation may take place as a result of an internal process of the server or as a client request.

![Figure 5.8 – Delete operation flow](image)

### 5.4 THE SERVER

The NMS server-side has to be capable to serve the requested operations coming from the client-side and establish a communication with the control plane when needed. The main motivation behind the decision of making the NMS a client/server application is to separate the graphical tool and user operation (i.e. the client-side) from the application engine (i.e. the server-side) in such a way the part in communication with the control plane can be close to the optical network equipment and monitors, while any authorized person may use the graphical tool to perform management tasks from any terminal with the right software and a connection to the network shared with the server. So the creation of a distributed application with two parts, one acting as client and the other as server, fits well with the mentioned philosophy.

For developing both client and server parts, the Eclipse IDE (see Figure 5.9) has been used along with Java programming language. The particular options of the development (EJBs and SWT) will be discussed later. Eclipse [15] is a programming platform specially designed for developing applications with a graphical front-end, but also suitable for developing any kind of applications. Originally it was developed by IBM but nowadays is being developed by the Eclipse Foundation, an independent non-profit organization whose goal is to foment an open-source community and a whole set of complementary products, capabilities and services.

The Eclipse IDE uses modules to provide full functionality, opposite to other monolithic environments where the functionalities are all included regardless if the user needs them or not. This module mechanism results in a very light platform for software components. Apart from
allowing Eclipse to extend itself by using other programming languages such C/C++ or Python, it permits working with text processing languages such LaTeX, network applications such Telnet and DBs management systems. The Software Development Kit (SDK) of Eclipse provides full support to Java programming language, which is the one used for developing the NMS.

Focusing on the server-side implementation, it has been decided to deploy it into a specialized application server. Deploying the server-side into a specialized server and not as a standard application with a Main routine makes the application more efficient and more tolerant to errors, among a key point that is the creation, instantiation and management of the Beans that compose the server-side of the NMS. The whole point will be discussed later. Eclipse allows deploying EJBs applications into the server chosen by the user under the same conditions as the ones in the moment of deploying the application in the real server.

JBoss is the server chosen for deploying the application. JBoss [16], developed by JBoss Inc., is an open-source Java Enterprise Edition (JEE)-based application server. Because it is Java-based, the JBoss application server operates cross-platform being usable on any operating system that Java supports. Also, it is the first open-source application server, prepared for producing and certifying JEE Ver1.4 applications, to come into the market, offering a high-performance platform for applications.
The prominent features of JBoss are the following:

- Clustering.
- Standard-compliant.
- Middleware services for any Java object.
- Incrustable, service-architecture oriented.
- Load Balancing.
- Failover.
- Java Naming and Directory Interface (JNDI) capable.

Another tool that has been used for developing the server-side is XDoclet. XDoclet [17], developed by XDoclet Team, is an open-source code generation library which enables attribute-oriented programming for Java via insertion of special Javadoc tags. It comes with a library of predefined tags, which simplify coding for various technologies. By using XDoclet is possible to generate automatically the code associated to the interfaces of the EJBs that compose the server.

Because this code has always the same structure and its programming is quite monotonous and repetitive, it is a good idea to use XDoclet for generating it in order to accelerate the developing process and reduce human errors. Moreover, XDoclet permits generating automatically the deployment descriptor, an Extensible Markup Language (XML) file that has all the properties and relationships of the EJBs programmed so JBoss can create, instantiate and manage correctly all of them.

### 5.4.1 STRUCTURE OF THE SERVER

Functionally, the server is composed by the modules shown in Figure 5.2, but its logical structure is quite different and more complex than that. Like most applications programmed using Java, the server is structured in packages, each one having multiple classes within them.
that may be used by other classes in other packages. The server is composed by two packages
which are the following:

- **db package**: it contains the class responsible of creating and destroying the connections
towards the DB.

- **domain package**: it contains the classes perform the server functionalities. It is
composed of various sub-packages:
  
  o **objects package**: contains a set of classes that represent the elements (OXC,
    OCC,) managed by the NMS.

  o **dbobjects package**: contains a set of classes that represent the tables within the
    DB. All the operations into the DB are done using these classes.

  o **objectmanager package**: contains a set of classes that manage the
corresponding objects and tables in order to satisfy any incoming request from
the client. In a sense, it is the core of the server. Also, it contains the methods to
establish a dialog with the control plane. A special class within the package,
called ParserManager, is used to load an initial topology, when requested by the
client, by parsing the corresponding XML configuration files. This package is
the one that has been programmed using the EJBs architecture.

  o **traps package**: contains the class that listens the incoming traps from the
control plane via SNMP, treats them and displays the result through the
standard output of the system.

  o **utils package**: contains various useful classes to perform some tasks by other
classes. Examples are the classes UDPmessage and UDPtransport needed for
establishing a dialog with the control plane.
A schema showing the whole structure and the relations among all the classes is displayed in Figure 5.10:

![Figure 5.10 – Structure of the server](image)

It can be seen in Figure 5.10 that the managers are integrated by three classes: a Bean class, a Home class and a Remote class. Their meaning and how they work are explained in the following point.
5.4.2 ENTERPRISE JAVA BEANS ARCHITECTURE

Any incoming request from the client will be treated by the corresponding manager reporting the result, if any, back to the client. These managers, as it has been said before, are implemented using the EJBs architecture from Java. EJBs architecture [18] is a managed, server-side component architecture for modular construction of enterprise applications. Its specification details how server applications provide objects to the client from its side, which are, precisely, the EJBs. The specification details the following aspects:

- Remote communication using *Common Object Request Broker Architecture* (CORBA).
- Transactions.
- Concurrency control.
- Event flow using *Java Messaging Service* (JMS).
- JNDI services.
- Security.
- Placement of components in a server application.

The specification of EJBs defines the roles played by the container of the EJBs and the EJBs themselves, aside of putting the EJBs into the container. It provides a standard model of distributed components from the server side. The goal of EJBs is to provide a model to the programmer that permits him to forget about generic problems of an application (concurrency, transactions, persistence, security, etc.) and focus on the development of the application’s logic itself. Being based in components, the EJBs are flexible and, above all, reusable.
5.4.2.1 OPERATION OF AN ENTERPRISE JAVA BEAN

EJBs are arranged in an EJB container within the application server. The specification describes how an EJB interacts with its container and how the client’s code interacts with the combination of container and EJB.

Each EJB must provide a Java implementation class and two Java interfaces. The EJB container creates instances of the Java implementation class to facilitate implementation of EJB. Java interfaces are used by the EJB client code. The two interfaces, known as home interface and remote interface, specify the signatures of the EJB remote methods. The remote methods are divided in two groups:

- Methods that are not tied to a specific instance, for example those used to create an EJB instance or to find an existing EJB entity. These methods are declared in the home interface.

- Methods linked to a specific instance. They are located in the remote interface.

Since it is just Java interfaces and not concrete classes, the EJB container generates classes for those interfaces that act as a proxy on the client. The client invokes a method on the generated proxies who in turn places the method arguments in a message and sends the message to the EJB server. The server will call a corresponding method to an instance of the Java implementation class to handle the remote method call.

As said before, any EJB must provide an implementation class and two interfaces:

- **Home interface**: allows the client code of an EJB to manipulate certain methods which are not associated to any particular instance.

- **Remote interface**: specifies the methods of public instances responsible for performing the operations. Faced with a client query, the container returns a serialized stub object that implements the remote interface. The stub knows how to pass *Remote Procedure Calls* (RPCs) to the server.
• **Implementation class**: is provided by the developer and facilitates the business logic and maintenance of the data of the object interfaces. Implements all the methods specified within the remote interface and, maybe, some methods specified within the home interface.

Figure 5.11 shows the relationship between an EJB, all the components associated to it and the client:

![Figure 5.11 – Architecture of an EJB](image)

The client searches into a JNDI service (in the NMS case, provided by the JBoss server) the location of the home interface of the EJB. The home interface is the responsible of creating and maintaining an instance of the implementation class and the remote interface, in collaboration with the home interface, executes the methods implemented into the EJB. The remote interface is the client view of any EJB, because the implementation class is never accessed directly by the client. Finally, the implementation class is the one that does all the operations with the DB.
There are three types of EJBs:

- **Entity EJBs**: its aim is to encapsulate the server-side objects that store the data. The entity EJBs have the fundamental characteristic of persistence:
  
  - *Container Managed Persistence* (CMP): the container is responsible for storing and retrieving the object’s data by mapping or linkage of the columns of a table in a DB with the attributes of the object.
  
  - *Bean Managed Persistence* (BMP): the object itself is the responsible, using a DB or other mechanisms, for storing and retrieving the data that refers to. For this reason, the implementation of all persistence methods has to be done by the programmer.

- **Session EJBs**: manage the flow of information on the server. Generally they serve customers as a front for the services provided by other components available on the server. There are two types of session EJBs:
  
  - *Stateful*: stateful session EJBs are distributed objects with a state. The state is not persistent but the access is limited only to a client.
  
  - *Stateless*: stateless session EJBs are distributed objects without a state associated to them. For this reason, they can be accessed concurrently by multiple clients. There is no guarantee that the value of the variables of the instance are maintained between calls of the same method.

- **Message-driven EJBs**: they are the only EJBs with an asynchronous behaviour. Using JMS, they subscribe themselves to a particular topic or queue and are activated when a message is received by the topic or queue. They do not require any kind of installation from the client.

The solution adopted for the manager classes has been to make them stateful session EJBs and relegate all the operations related to the DB to the other classes used by them as it has been described in 5.4.1.
5.4.3 SIMPLE API FOR XML

Among all the managers implemented using the EJBs architecture, there is a special case within them called ParserManager. This class, using the classes related to it as shown in Figure 5.10, is the responsible to load an initial topology from a set of XML files to the DB if a request of this kind comes from the client.

There are multiple options to perform a parsing of an XML file and the one adopted is using Simple API for XML (SAX). SAX [19] is a serial access parser Application Programming Interface (API) for XML.

SAX provides a mechanism for reading data from an XML document. A parser which implements SAX functions as a stream parser, with an event-driven API. The user defines a number of call-back methods that will be called when events occur during parsing. Using these methods, the programmer can specify any desired action when a SAX event occurs. The SAX events include:

- XML text nodes.
- XML element nodes.
- XML processing Instructions.
- XML comments.

Events are fired when each of these XML features are encountered, and again when the end of them is encountered. XML attributes are provided as part of the data passed to element events. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

There is no formal specification for SAX. The Java implementation of SAX is considered to be normative, and implementations in other languages attempt to follow the rules laid down in that implementation, adjusting for the differences in language where necessary.
Although it is not very suitable for parsing XML documents where a whole document view is needed, SAX has some interesting benefits that make it the solution adopted for parsing the configuration files already cited.

The quantity of memory that a SAX parser needs is typically much smaller than other XML parsers. The memory footprint of a SAX parser is based only on the maximum depth of the XML file (the maximum depth of the XML tree) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the parsed tree itself.

Because of the event-driven nature of SAX, processing documents can often be faster than other parsers. Also, using SAX, streamed reading from disk and processing XML documents larger than main memory is possible in contrast of other parsers where this action may be impossible.

**5.4.4 SNMP TRAP LISTENER**

The server has a dedicated class, without relations with other classes, which its purpose is listening to any incoming trap that may come from the control plane, identify the type of trap, treat them, get their value and reporting the result through the standard output of the system.

Before explaining in more detail how this module works, an explanation of what is SNMP and its structure is discussed in the following point.

**5.4.4.1 SNMP**

SNMP [20] is a User Datagram Protocol (UDP)-based network protocol. It is used mostly in NMSs to monitor network-attached devices for conditions that warrant administrative attention. SNMP is a component of the IP protocol suite (Figure 5.12) as defined by the IETF. It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects.

SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications.
Multiple versions of the SNMP protocol have been released over the time. The first one, SNMPv1, was the initial implementation of the SNMP protocol. It operated over several popular protocols such as UDP or IP so it was seen as an easy way to implement management protocol, but it was severely criticized for its poor security. SNMPv2 revised version 1 and included improvements in the areas of performance, security, confidentiality, and manager-to-manager communications, but it still lacked privacy and authentication. SNMPv3, to the date the last version that has appeared, solve these problems and also provides access control.

5.4.4.2 SNMP ARCHITECTURE

In typical SNMP use, one or more administrative computers have the task of monitoring or managing a group of hosts or devices on a computer network. Each managed system (also called slave) executes, at all times, a software component called an agent which reports information via SNMP to the managing systems (also called masters).

Essentially, SNMP agents expose management data on the managed systems (slaves) as variables. But the protocol also permits active management tasks, such as modifying and applying a new configuration. Configuration and control operations are used only when changes are needed to the network infrastructure. The monitoring operations are usually performed on a regular basis.
Typically, SNMP uses UDP ports 161 for the agent and 162 for the manager. The manager may send requests from any available source port to port 161 in the agent (destination port). The agent response will be sent back to the source port. The manager typically receives notifications on port 162. The agent may generate notifications from any available port.

An SNMP-managed network consists of three key components:

- Managed devices (slaves).
- Agents (software which runs on a slave device).
- NMS (software which runs on a master device).

A managed device is a network node that implements an SNMP interface that allows unidirectional (read-only) or bidirectional access to node-specific information. Managed devices exchange node-specific information with the NMSs. Sometimes called network elements, the managed devices can be any type of device, such as routers, access servers, switches, bridges, printers, etc.

An agent is a network-management software module that resides on a managed device. An agent has local knowledge of management information and translates that information to or from an SNMP specific form.

The NMS executes applications that monitor and control managed devices. NMS provides the bulk of the processing and memory resources required for network management. One or more NMSs may exist on any managed network.

SNMP itself does not define which information (which variables) a managed system should offer. Rather, SNMP uses an extensible design, where the available information is defined by Management Information Bases (MIBs). MIBs describe the structure of the management data of a device subsystem; they use a hierarchical namespace containing Object Identifiers (OIDs). Each OID identifies a variable that can be read or set via SNMP. An OID uniquely identifies a managed object in the MIB hierarchy.

The MIB hierarchy can be depicted as a tree (see Figure 5.13) with a nameless root, the levels of which are assigned by different organizations. The top-level MIB OIDs belong to different standards organizations, while lower-level object IDs are allocated by associated organizations.
A managed object (sometimes called a MIB object, an object, or a MIB) is one of any number of specific characteristics of a managed device. Managed objects are made up of one or more object instances (identified by their OIDs), which are essentially variables.

A managed object has these unique identities:

- The object name. For example, and following the tree of Figure 5.13, iso.identified-organization.dod.internet.private.enterprise.cisco.temporaryvariables.AppleTalk.atInput.
  or

- The equivalent object descriptor. For example, 1.3.6.1.4.1.9.3.3.1.

SNMP must account for and adjust to incompatibilities between managed devices. Different computers use different data-representation techniques, which can compromise the ability of SNMP to exchange information between managed devices.
5.4.4.3 SNMP OPERATIONS

In the previous point it has been discussed how SNMP organizes information, but it has been left out how actually works about gathering management information. This point takes a more in depth look how operates.

The Protocol Data Unit (PDU) is the message format that managers and agents use to send and receive information. There is a standard PDU format for each of the following SNMP operations:

- **Get**: the get request is initiated by the NMS, which sends the request to the agent. The agent receives the request and processes it to best of its ability. Some devices that are under heavy load, such as routers, may not be able to respond to the request and will have to drop it. If the agent is successful in gathering the requested information, it sends a get-response back to the NMS, where it is processed.

- **Get-next**: the get-next operation lets the NMS issue a sequence of commands to retrieve a group of values from a MIB. In other words, for each MIB object the NMS wants to retrieve, a separate get-next request and get-response are generated. The get-next command traverses a sub-tree in lexicographic order. Since an OID is a sequence of integers, it's easy for an agent to start at the root of its MIB object tree and work its way down until it finds the OID it is looking for. When the NMS receives a response from the agent for the get-next command it just issued, it issues another get-next command. It keeps doing this until the agent returns an error, signifying that the end of the MIB has been reached and there are no more objects left to get.

- **Get-bulk (SNMPv2 and SNMPv3)**: SNMPv2 defines the get-bulk operation, which allows a management application to retrieve a large section of a table at once. The standard get operation can attempt to retrieve more than one MIB object at once, but message sizes are limited by the agent's capabilities. If the agent can't return all the requested responses, it returns an error message with no data. The get-bulk operation, on the other hand, tells the agent to send as much of the response back as it can. This means that incomplete responses are possible.

- **Set**: The set command is used to change the value of a managed object or to create a new row in a table. Objects that are defined in the MIB as read-write or write-only can
be altered or created using this command. It is possible for an NMS to set more than one object at a time.

- **Get-response**: the get-response is sent by the SNMP agent and it contains the requested instance by the NMS.

- **Trap**: A trap is a way for an agent to tell the NMS that an error or failure has happened. The trap originates from the agent and is sent to the trap destination, as configured within the agent itself. The trap destination is typically the IP address of the NMS. No acknowledgment is sent from the NMS to the agent, so the agent has no way of knowing if the trap makes it to the NMS. Since SNMP uses UDP, and since traps are designed to report problems with a network, traps are especially prone to getting lost and not making it to their destinations. However, the fact that traps can get lost doesn't make them any less useful; in a well-planned environment, they are an integral part of network management.

- **Notification (SNMPv2 and SNMPv3)**: In an effort to standardize the PDU format of SNMPv1 traps, SNMPv2 defines a Notification-Type. The PDU format for Notification-Type is identical to that for get and set.

- **Inform (SNMPv2 and SNMPv3)**: SNMPv2 provides an inform mechanism, which allows for manager-to-manager communication. This operation can be useful when the need arises for more than one NMS in the network. When an inform is sent from one NMS to another, the receiver sends a response to the sender acknowledging receipt of the event. This behaviour is similar to that of the get and set requests.

- **Report (SNMPv2 and SNMPv3)**: The report operation was defined in the draft version SNMPv2 but never implemented. It is now part of the SNMPv3 specification and is defined for internal SNMP communication.
5.4.4.4 FUNCTIONING OF THE SNMP TRAP LISTENER

Among all the operations defined previously, the trap operation is the only one that will be listened and treated by the module. Any time a trap is generated in the control plane, the NMS will listen for it into the port 162. When the NMS receives a trap, it needs to know how to interpret it; that is, it needs to know what the trap means and how to interpret the information it carries. A trap is first identified by its generic trap number. There are seven generic trap numbers (0-6) in which generic trap 6 is a special catch-all category for “enterprise-specific” traps, which are traps defined by vendors or users that fall outside of the six generic trap categories. Enterprise-specific traps are further identified by an enterprise ID (i.e., an OID somewhere in the enterprises branch of the MIB tree) and a specific trap number chosen by the enterprise that defined the trap. Thus, the OID of an enterprise-specific trap is enterprise-id.specific-trap-number.

A trap is usually packed with information. This information is in the form of MIB objects and their values; these object-value pairs are known as variable bindings. For the generic traps 0 through 5, knowledge of what the trap contains is generally built into the NMS software or trap receiver. The variable bindings contained by an enterprise-specific trap are determined by whoever defined the trap. For example, if an OXC port in a port rack fails, the rack’s agent may send a trap to the NMS informing it of the failure. The trap will most likely be an enterprise-specific trap defined by the rack’s manufacturer; the trap’s contents are up to the manufacturer, but it will probably contain enough information to let the NMS determine exactly what failed.

Using the Java API referenced at [22], it is possible to build a class that listens for any incoming trap from the control plane, looks at its variable bindings, identifies the type of the trap and extracts the value in order to display it through the standard output of the system, in conjunction with an informative message telling that a trap has been received and its type.

The implemented listener detects automatically which version of SNMP it is being used when a trap is received. This is an important issue because the way a trap has to be treated depends on the version of the SNMP it uses. After having received a trap and identified its version, the listener extracts from the variable bindings the OID of the trap and compares it with a list of predefined trap types: if it matches with any of them, the listener extracts the value and proceeds as explained; if it doesn’t match with any of them, it doesn’t perform any further operation. In the actual version of the emulated control plane that has been used to test and develop the NMS,
a trap is sent only when a cross-connection has been performed to establish an LSP, so the number of checks that the listener has to perform for knowing the trap’s type is reduced to one.

5.4.5 COMUNICATION WITH THE CONTROL PLANE

The NMS has to establish a communication with the control plane any time an LSP-establishment or LSP-teardown request comes from the client. The responsible of setting this communication, sending the appropriate message, listening to the response and reporting back the result to the client is the class LSPManager within the sub-package objectmanager.

This class has two dedicated methods, called requestLSP and teardownLSP, which perform this communication. Whenever any of these two operations is requested from the client, it calls one of the mentioned methods using the remote interface associated to the class as explained in 5.4.2.1, providing them with the necessary parameters to perform the operation.

The communication between the LSPManager and the control plane is done using UDP messages. When any of the mentioned methods is called, the LSPManager opens an UDP port towards a special IP within the control plane called the manager node. The purpose of the manager is to agglutinate any incoming message to the control plane and coordinate the operations of the other nodes when a request for establishing or tearing-down an LSP is received.

Once the operation is performed, the manager answers back to the server reporting if the operation succeeds or not and the corresponding result. The manager listens to requests on port 10000 has shown in Figure 5.2.

After receiving an answer, and if the operation succeeds, the LSPManager updates the DB with new information regarding the actual state of the network and returns to the client the result of the operation, notifying that there hasn’t been any problem. In the other case, if the operation has not been properly performed, the LSPManager reports only the failure to the client (no modification of the DB is done).

The messages exchanged between the LSPManager and the control plane and the flow of the operations are described in the following points.
5.4.5.1 LSP-ESTABLISHMENT REQUEST

The UDP message that is sent to the control plane has the format shown in Figure 5.14:

The remoteHostAddr and remotePort contain the IP address and the port of the CC of the source OCC for the requested LSP; the length field contains the length of the data field; and the sendingSocket field contains the port used for the server to send the message, where the manager node has to send the replay.

Inside the data field there is again a field that contains the length of the data; an operationCode field that contains an integer representing the operation requested (for the establish-LSP request, the field is 1); two pairs of IPs and ports representing the IP and the port (physical interface) of both source and destination OXCs of the LSP; a classofService field that contains an integer representing the class of service (best effort, unprotected, restorable,...) requested when establishing the LSP (for the actual version of the NMS, only unprotected class of service is contemplated); and a BW field that contains the requested bit-rate for the LSP.
The exchange of messages between the NMS and the control plane is illustrated in Figure 5.15:

![Figure 5.15 – LSP establishment](image)

First, the NMS sends the previous explained packet with the request of establishing the LSP. If the request can be served and the LSP can be established, the control plane answers back with an Acknowledgment packet that contains the ID assigned to the LSP; otherwise, if the request can not be served, a Negative Acknowledgment packet is send back to the NMS telling that the requested LSP couldn’t be established. After sending the Acknowledgment packet, the control plane sends another packet that contains the details of the route: the ID of the LSP; the wavelength used; and a variable number of pairs IP/port representing the IPs and ports (physical interfaces) of the OXCs that form the route.

When the LSP has been established and the server has the details of the route, these details are returned back the client.
5.4.5.2 LSP-TEARDOWN REQUEST

The UDP message that is sent to the control plane has the format shown in Figure 5.16:

```
<table>
<thead>
<tr>
<th>remoteHostAddr</th>
<th>remotePort</th>
<th>Data</th>
<th>length</th>
<th>sendingSocket</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes</td>
<td>2 bytes</td>
<td>8 bytes</td>
<td>2 bytes</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>
```

Figure 5.16 – LSP-teardown request message format

The meaning of the fields is the same as in the case of the LSP-establishment message packet. The only difference resides in the data field: again, the length field contains the length of the data and the operationCode field contains an integer representing the operation requested (for the teardown-LSP request, the field is 5); the tunneldt field contains an integer that represents the ID of the LSP that has to be torn-down.

The exchange of messages between the NMS and the control plane is illustrated in Figure 5.17:

```
NMS                   Control Plane
```

Figure 5.17 – LSP tearing-down
First, the NMS sends the previous explained packet with the request of tearing-down the LSP. If the request can be served and the LSP is tore-down, the control plane answers back with an Acknowledgment packet that contains the ID of the torn-down LSP; otherwise, if the request can not be served, a Negative Acknowledgment packet is send back to the NMS telling that the requested LSP couldn’t be torn-down.

When the LSP has been torn-down and the server has the certainty that the operation succeeded, it returns an acknowledgment to the client.

**5.5 THE CLIENT**

The NMS client-side has to be capable to provide all the graphical tools to act as a proper front-end for the user, allowing him to perform any supported management task in a graphical way, being both the interface and the operations intuitive, self-explanatory and neat. Moreover, the client has to have a proper interface to establish a communication with the server-side of the NMS whenever it is needed.

As explained in 5.4, the Eclipse IDE, along with Java programming language, has been used to develop the client-side. Because the nature of the graphical environment that is composed of multiple widgets (objects) that are dynamically created and destroyed, the object-oriented philosophy of Java fits very well to the needs of the client-side since objects can be created and destroyed at user’s will. Moreover, during the phase of requirements analysis, it was decided that SWT was going to be the tool used to develop the graphical part of the client and, because SWT is developed using Java, it was a natural act to use Java to develop also the client.

Another important reason for using Eclipse along with Java is that Eclipse, as mentioned at 5.4, is specially designed for applications that need a graphical front-end, improving their performance.

The structure of the client is discussed in the following point.
5.5.1 STRUCTURE OF THE CLIENT

As shown in Figure 5.2, the client is composed by two modules: a SWT module and a middleware module. And, like the server, it is structured in packages, each one having multiple classes within them that may be used by other classes in other packages. The client is composed by the following packages:

- **middleware package**: contains all the classes needed for communicating the graphical front-end with the server. It is composed of multiple sub-packages:
  
  - **objectmanager package**: has the same purpose as the package with the same name in the server.
  
  - **objectphpmanager package**: similar to the other managers but in this case, their purpose is to manage the associated information to the PP of the elements represented in the graphical view. This information resides only in the client-side of the NMS, so the server is unaware of it.
  
  - **phpobjects package**: contains the objects that represent the PP. These objects are used by the phpmanagers. It also contains a sub-package associated to it:
    
    - **components package**: representation of each element that compose the phpobjects. They are associated to the physical devices that have PP information such as a fibre or an amplifier.
  
  - **utils package**: contains useful classes that allow the loading of the PP from a set of XML files.

- **presentation package**: it contains all the classes developed using SWT. They allow to build up the graphical front-end and all its functionalities. Also it contains a set of Portable Network Graphics (PNG) files that contain all the images used for icons and logos in the graphical tool.

Aside from these packages, the client also has a Main class where the execution of the tool starts. It is also in this class where the instances to the Beans into the server are obtained.
Another important point is that the client has to know where to find the associated classes to the server components that needs to use. Aside of using the JNDI service offered by JBoss that has been already explained, it is needed to add the associated packages to the server to the building path of the client’s Java Archive (JAR) file, so when building the JAR file, all the necessary resources will be included in it.

A schema showing the whole structure and the relations among all the classes is displayed in Figure 5.18. Because the high complexity of the relations between the presentation classes and the other packages and the large number of classes the package contains, the details of the presentation package are not shown:

![Figure 5.18 – Structure of the client](image-url)
5.5.2 PRESENTATION MODULE

As said before, the presentation module contains all the necessary classes to build up the graphical front-end. This is the only module that the user is aware of it because the other modules (either in the client-side or the server-side) remain invisible to him. It establishes a communication with the server-side of the NMS through the middleware module by calling the proper methods of the classes that compose the middleware package.

Although it is not feasible to represent all the classes that compose the presentation module and the relationships between them in a diagram, due the large number of classes and the complexity of the relationships, there are four very important classes that need some further explanation:

- **GlobalContainer class**: this class is the starting point of the graphical part of the client. Also it is the one that gives continuity to the graphical front-end: as long as the class is not disposed, the graphical tool will continue to operate. It gives a substrate for the other classes, which, in fact, are placed above it. Mainly it is a shell, resized to fill the computer screen, with a canvas put above it. A canvas is a special widget that allows to draw graphics into it and place other widgets above it. The following three classes are compositions of multiple widgets that are placed in the GlobalContainer class.

- **ToolMenu class**: this class is, in fact, a toolbar that contains the necessary options and menus to perform management operations over the represented network. Through some of the options that contains, it is possible to load an initial topology from a set of XML files, as explained before, and represent that topology.

- **PaintingZone class**: this class is a big canvas where the graphical representation of the network is drawn. It allows to move the components and, using the proper options on the ToolMenu class, add/erase/resize the represented components. It is also where the established LSPs are shown by lighting up with special colours the representation of the TE links that compose an LSP. In addition, specific management operations in particular components of the network can be done through that class by selecting the desired component.

- **InformationTabs class**: is composed of multiple tabs where all the information regarding the components of the network, such as OXCs, TE links, etc. and the
information associated to the PP. The information is dynamically shown/removed depending on the actions taken with other classes.

5.5.2.1 SWT

As mentioned many times before, SWT has been used to develop the classes related to the graphical part of the client-side of the NMS. SWT [23] [24] is a graphical widget toolkit for use with the Java platform. It was originally developed by IBM and is now maintained by the Eclipse Foundation in tandem with the Eclipse IDE. SWT is written in Java. To display Graphical User Interface (GUI) elements, the SWT implementation accesses the native GUI libraries of the operating system using Java Native Interface (JNI) in a manner that is similar to those programs written using operating system-specific APIs. Programs that call SWT are portable, but the implementation of the toolkit, despite the fact that it is written in Java, is unique for each platform.

The roots of SWT go back to work that Object Technology International (OTI) did in the 1990s when it created the multiplatform, portable, native widget interfaces for Smalltalk (originally for OTI Smalltalk, which became IBM Smalltalk in 1993). IBM Smalltalk's Common Widget layer provided fast, native access to multiple platform widget sets while still providing a common API without suffering the “lowest common denominator” problem typical of other portable GUI toolkits. IBM was developing their VisualAge development IDE that was written in Smalltalk. They decided to open-source the project, which led to the development of Eclipse, intended to compete against other IDEs such as Microsoft Visual Studio. Eclipse is written in Java, and IBM developers, deciding that they needed a toolkit that had “native look and feel” and “native performance”, created SWT.

SWT is a wrapper around native code objects, such as Motif objects etc. Because of this, SWT widgets are often referred to as “heavyweight”, evoking images of a light Java wrapper around a “heavy” native object. In cases where native platform GUI libraries do not support the functionality required for SWT, SWT implements its own GUI code in Java.. In essence, SWT is a compromise between low level performance and high level ease of use. SWT is a relatively simpler toolkit than other widget toolkits, with less extraneous functionality for the average developer.

Although SWT does not implement the popular Model-View-Controller architecture used in many high level GUI toolkits, it does provide a platform-independent, higher-level Model-
View-Controller abstraction on top of SWT. SWT was designed to be a high performance GUI toolkit; faster, more responsive and lighter on system resource usage than other toolkits.

SWT widgets have the same “look and feel” as native widgets because they often are the same native widgets. Since SWT is simply a wrapper around native GUI code, it does not require large amounts of updates when native code is changed, providing that operating system vendors are careful not to break clients of their API when the operating systems are updated.

SWT aims for deep platform integration, the Eclipse reference to SWT’s use of native widgets. This deep integration can be useful in a number of ways, for example enabling SWT to wrap ActiveX objects on Microsoft Windows.

Due to the use of native code, SWT classes do not allow for easy inheritance for all widget classes, which some people consider can hurt extensibility. This can make customizing existing widgets difficult, needing extra work to make the new widget work on every platform.

SWT widgets, unlike almost any other Java toolkit, require manual object unallocation, as opposed to the standard Java practice of automatic garbage collection. SWT objects must be explicitly unallocated using the .dispose() function, which is analogous to the C language’s free. If this is not done, memory leaks or other unintended behaviour may result. The need for manual object unallocation when using SWT is largely due to SWT’s use of native objects. As these objects are not tracked by the Java virtual machine, it is unable to ascertain whether or not these native objects are in use, and thus unable to garbage collect them at an appropriate time.

SWT must be ported to every new GUI library that needs supporting. Unlike other Java-based toolkits, SWT is not available on every Java-supported platform since SWT is not part of the Java release. Since SWT uses a different native library for each platform, SWT developers may be exposed to platform specific bugs.

SWT exposes developers to more low level details than other toolkits. This is because SWT is technically just a layer over native library provided GUI functionality - that said, exposing the programmer to native GUI code seems to be part of the design intent of SWT: Its goal is not to provide a rich user-interface design framework but rather the thinnest possible user-interface API that can be implemented uniformly on the largest possible set of platforms while still providing sufficient functionality to build rich GUI applications.
Despite that SWT has some drawbacks, as explained along this whole point, it is really easy to use and understand, and any person with sufficient knowledge about Java and little patience can program any simple widget with very few code, as it shows the example in Figure 5.19:

```java
public class HelloSWT {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("My First SWT GUI");
        shell.setSize(200, 100);

        Text helloText = new Text(shell, SWT.CENTER);
        helloText.setText("Hello SWT!");
        helloText.setBounds(47, 20, 100, 20);

        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

Figure 5.19 – Simple widget programming example

Nerveless it is said that for a more sophisticated behaviour widget, the programmer has to expend more time and use more code to reach the goal.

### 5.5.3 MIDDLEWARE MODULE

The middleware module is the link between the graphical part of the client and the server-side. It works in a similar way than the managers of the server-side: Java classes that manage representations of the objects that compose the network. In fact, the managers used for the middleware module are just a mere copy of the managers used in the server but with two essential differences:
• The middleware managers are not programmed using the EJBs architecture, they are simply plain Java classes.

• Aside from some methods used only inside the client, the middleware managers develop their methods by making the correspondent call to the proper manager into the server.

The idea behind making a middleware module to communicate the graphical front-end with the server and not do this communication directly is to avoid excessive petitions to the server, because, at the core, EJB works like Remote Method Invocation (RMI) which tends to slow down its performance when a high number of remote calls is done.

To reduce the number of petitions to the server, the middleware module has a local copy of the information contained into the server, so any petition from the graphical tool can be satisfied through this local copy. Any modification of this information is first reflected into the middleware and the updated into the server, so the middleware will always have the most recent information. Only in some cases, mainly when an LSP is established or torn-down, the middleware will have to make a petition to the server to update its local information, because the information generated by these operations is only seen in the server. By following this methodology, the speed and performance of the whole NMS is enhanced notoriously.

The communication between the middleware and the server is done as explained in 5.4.2.1 where the working method of an EJB was explained: a middleware manager uses a remote interface, obtained through the JNDI service and the home service, to make a remote call to a method of a manager in the server.

Figure 5.20 shows a schema of the communication involving the presentation, middleware and server parts:
Also, in some of the middleware managers, information regarding the position, size and colour of the elements represented is stored. It makes sense to store this information into the local memory of the client because it is information that has to be accessed many times to ensure that the network is drawn correctly, so storing it into the server will only slow down the performance of the NMS. Also, this information is only valid for one session, so storing it permanently into the server is not the correct option.

5.5.4 PP REPRESENTATION

Inside the middleware package, there are some special managers that its purpose is to manage the information related to the PP of the devices (fibres, amplifiers, transmitters…) of the elements of the network. This is done in the same way as it is done for loading an initial topology: parsing a set of XML files.

There are two XML files for each one of the nodes represented: one containing information regarding the TE links and the OXC, and the other containing information regarding the ports. The parsing of the files is done in the same manner than the topology files, using SAX, but the information parsed is not stored permanently into the DB. In fact, the information is stored into the local memory of the client.

So, every time a user starts a new session, there is no PP information associated to the network, regardless if a topology has been already loaded. If the user wants to see the associated PP information, he has to load that information from the corresponding XML files. In the actual version of the NMS, as a consequence of the version of the control plane used that does not implement any kind of monitor, the PP information does not change over the time, it remains static. The PP information, as other information associated to the network, is displayed in one of the tabs contained into the graphical-front end, but in contrast of the other tabs that are exclusively dedicated to shown information associated to particular elements, the PP tab is used to represent all the PP information, so its structure changes dynamically depending on the element chosen for this kind of petition.
Once the NMS was fully implemented and the communication with the control plane was set up properly, its functionalities were tested over an experimental ASON/GMPLS network test-bed.

These tests were intended to:

- Test that the “loading topology” operation was done correctly.
- Verify that all the graphical functionalities were properly integrated and none of them presented any erroneous behaviour.
- Check that the communication with the control plane worked as expected.

In addition, the proper deployment of the server-side of the NMS onto the JBoss server was tested, although that for performing all the other tests, a monolithic (non-distributed) version of the NMS was used. This version works exactly as the client/server version but all the functionalities are concentrated into one unique block, so the application works in local. The only difference is that the managers are not longer implemented using EJB; they are simply plain Java classes. But, regardless of this difference, the overall behaviour of the NMS is the same in both cases.

The reason of using a monolithic version of the NMS is that for the process of testing/debugging the application, every time a modification has to be done into the source code, the JBoss server has to be stopped, synchronize it again with the new code and start it from the beginning. This process is quite tedious and takes some time, so, in order to perform more agile tests, the already mentioned version of the NMS has been used. Another reason is the fact that the distribution of both client-side and server-side into different hosts and the communication between those hosts has not been implemented in the actual version of the NMS, so all the tests have been performed in local.
6.1 TEST NETWORK CHARACTERISTICS

The test network is based in the Deutsche Telekom Network. This network consists of 14 nodes and 23 links with an average node degree of 3.29 [25]. The topology of this network is depicted in Figure 6.1:

![Deutsche Telekom Network topology](image)

Figure 6.1 – Deutsche Telekom Network topology

Other characteristics of the network are shown in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes:</td>
<td>14</td>
</tr>
<tr>
<td>Number of links:</td>
<td>23</td>
</tr>
<tr>
<td>Node degree:</td>
<td>3.29 (min. 2, max. 6)</td>
</tr>
<tr>
<td>Link length (km):</td>
<td>186 km (min. 37, max. 353)</td>
</tr>
<tr>
<td>Span length (km):</td>
<td>25 km (Long Aggr.), (min. 0.21, max. 65)</td>
</tr>
<tr>
<td>Number of Span:</td>
<td>7.5 (Long Aggr.), (min. 1, max. 17)</td>
</tr>
<tr>
<td>Path length (km):</td>
<td>410 km (min. 37, max. 874)</td>
</tr>
<tr>
<td>Hop count:</td>
<td>2.35 (min. 1, max. 5)</td>
</tr>
</tbody>
</table>
6.2 DEPLOYMENT OF THE SERVER

As mentioned before, the distributed version of the NMS will not be used for performing all the tests. Nevertheless, a test to analyze whether the server has been correctly deployed has been also performed. The compiled classes of the server, along with the source code and other resources such as libraries are packed into a JAR file; this operation is also performed with the client.

With this JAR file and the deployment descriptor associated to it, the JBoss server is able to run the server code. The first step for executing this operation is to synchronize the JAR file with the JBoss server, that is, to bond the file with the server. This is performed using the publish command in the JBoss server. Figure 6.2 shows the result of this action:

![Figure 6.2 – JBoss publish command](image)

It can be seen that the operation succeeds, resulting in a correct bonding of the server JAR file with the JBoss server. Once the file has been correctly synchronized, the JBoss server can be started. During the starting operation, and with the help of the deployment descriptor, the JBoss server will associate each of the EJB classes with its interfaces, so that it can give a proper JNDI service to the user. A fragment of the log file generated when starting the JBoss is shown below:

```
12:17:12,417 INFO  [ServerImpl] Starting JBoss (Microcontainer)...
12:17:12,419 INFO  [ServerImpl] Release ID: JBoss [Morpheus] 5.0.0.CR2 (build: SVNTag=JBoss_5_0_0_CRS date=200809171139)
12:17:12,423 INFO  [ServerImpl] Home URL: file:/home/apages/App/jboss-5.0.0.CR2
12:17:12,424 INFO  [ServerImpl] Library URL: file:/home/apages/App/jboss-5.0.0.CR2/lib/
12:17:12,425 INFO  [ServerImpl] Patch URL: null
12:17:12,425 INFO  [ServerImpl] Server Name: default
```
12:17:12,426 INFO [ServerImpl] Server Temp Dir: /home/apages/App/jboss-5.0.0.CR2/server/default/tmp
12:17:12,427 INFO [ServerImpl] Root Deployment Filename: jboss-service.xml
12:17:13,277 INFO [ServerImpl] Starting Microcontainer, bootstrapURL=file:/home/apages/App/jboss-5.0.0.CR2/server/default/conf/bootstrap.xml

12:17:52,815 INFO [EjbDeployer] installing bean: ejb/#ConnectionPoint,uid33476402
12:17:52,816 INFO [EjbDeployer] with dependencies:
12:17:52,816 INFO [EjbDeployer] and supplies:
12:17:52,817 INFO [EjbDeployer] jndi:ConnectionPoint
12:17:52,817 INFO [EjbDeployer] jndi:local/ConnectionPoint@29150942
12:17:52,817 INFO [EjbDeployer] installing bean: ejb/#DataLink,uid23497899
12:17:52,817 INFO [EjbDeployer] with dependencies:
12:17:52,817 INFO [EjbDeployer] and supplies:
12:17:52,817 INFO [EjbDeployer] jndi:local/DataLink@11622891
12:17:52,817 INFO [EjbDeployer] jndi:DataLink
12:17:52,818 INFO [EjbDeployer] installing bean: ejb/#TELink,uid27084246
12:17:52,818 INFO [EjbDeployer] with dependencies:
12:17:52,818 INFO [EjbDeployer] and supplies:
12:17:52,818 INFO [EjbDeployer] jndi:TELink
12:17:52,818 INFO [EjbDeployer] installing bean: ejb/#SNC,uid5503211
12:17:52,818 INFO [EjbDeployer] with dependencies:
12:17:52,819 INFO [EjbDeployer] and supplies:
12:17:52,819 INFO [EjbDeployer] jndi:SNC
12:17:52,819 INFO [EjbDeployer] jndi:local/SNC@2199864
12:17:52,819 INFO [EjbDeployer] installing bean: ejb/#Nodes,uid16461919
12:17:52,819 INFO [EjbDeployer] with dependencies:
12:17:52,819 INFO [EjbDeployer] and supplies:
12:17:52,819 INFO [EjbDeployer] jndi:Nodes
12:17:52,820 INFO [EjbDeployer] jndi:local/Nodes@29686996
12:17:52,820 INFO [EjbDeployer] installing bean: ejb/#Ports,uid19187487
12:17:52,820 INFO [EjbDeployer] with dependencies:
12:17:52,820 INFO [EjbDeployer] and supplies:
12:17:52,820 INFO [EjbDeployer] jndi:Ports
12:17:52,820 INFO [EjbDeployer] installing bean: ejb/#LSP,uid26003689
12:17:53,053 INFO [EjbModule] Deploying ConnectionPoint
12:17:53,162 INFO [EjbModule] Deploying DataLink
12:17:53,179 INFO [EjbModule] Deploying TELink
12:17:53,195 INFO [EjbModule] Deploying SNC
Looking at the log it can be seen that all the EJBs are correctly deployed, installed and associated with their interfaces. From this point, and using the JNDI service, the client can obtain an instance to any of the EJBs that compose the server and start to operate as explained in chapter 5.
6.2 TESTING THE GRAPHICAL FRONT-END

Before detailing the tests done and the collected results, a general perspective of the graphical front-end is shown in figure 6.3:

![NMS graphical interface](image)

**Figure 6.3 – NMS general perspective**

In Figure 6.3, the three important parts that compose the graphical tool (ToolMenu, PaintingZone and InformationTabs) can be easily identified, being the ToolMenu the upper part in the screen, the PaintingZone in the middle and the InformationTabs in the lower part.

6.2.1 LOAD A TOPOLOGY

A very important operation to be included into the functionalities of the NMS is the possibility to load a predefined topology from a set of XML files using the graphical tool. It helps to speed up the process of passing the desired topology into the DB, because doing it by hand using SQL queries for every entry of every table can be a very long and complex operation, leading to an incorrect loading of the desired network.
To perform this operation, into the toolbar the user has to select the option **Database**, and then **Load Topology From XML**, it will appear a form as the one shown in Figure 6.4:

![Load Topology form](image)

**Figure 6.4 – Load Topology form**

The form allows the user browsing through the file system and selecting the desired XMLs, one for each OCC and one for each OXC. It is important to follow the note that appears at the bottom of the form to ensure a proper loading of all the elements. This restriction happens because there is some dependency between the information of the XML files, so they must be loaded in a specific order.

Once the topology has been correctly loaded into the DB, the user can decided to draw it into the graphical tool. To execute this operation, he has to select **Database→Draw Network From DB**. Once the network has been drawn and after redistributing the nodes from their default position, the representation of the network is like the one shown in Figure 6.5:
From this moment, the user can start to perform management operations like editing the elements of the network or establish an LSP. He can also add/delete elements by selecting the proper options into the **Nodes** or **TELinks** menus. These options also allow drawing a topology from scratch, if no one has been loaded from a set of XML files, by adding any desired component and then editing it.

The loading operation, like any operation that can be performed, will warn the user with an error/warning message if it has not been executed correctly or some field in one of the forms has an incorrect value or format.

The user can resize the represented network by choosing the option **View→Zoom** and selecting the desired zoom from the list. This is particularly useful when working with large networks.
6.3.2 EDIT/REPRESENT A COMPONENT’S INFORMATION

By selecting one of the represented elements, either node or TE link, the user can edit its properties using the correct option in the popup menu that appears. For example, if the user wants to edit the information associated with one of the nodes, by selecting the desired node with the right button of the mouse and choosing the Edit option, it will appear a form that allows editing the information associated to the node (Figure 6.6):

![Edit Node Properties form](image)

The information introduced is stored automatically into the DB after pressing the Save button. If any field of the form is incorrect, it will detect it and display the corresponding warning message as said before.

This operation can also be performed to edit the information associated to a TE link or the ports of an OXC.

Also, by selecting an element, it is possible to display its associated information into the corresponding tab of the tab folder in the lower part. For example, if the user wants to know the
information associated to the data links of a particular TE link, he can perform this by selecting the option DataLinks into the popup menu. The result of this action is shown in Figure 6.7:

![Data link information](image)

Figure 6.7 – Data link information

The data links displayed in the tab correspond to the highlighted TE link. It is also possible to display the information associated to an OXC, the ports of a particular OXC or a TE link.

By selecting the option View→Return Default View it is possible to deselect any highlighted element and clean the corresponding tab from the tab folder.

### 6.2.3 LOAD/REPRESENT PP INFORMATION

As mentioned during chapter 5, the NMS offers the possibility to load information regarding the PP of the components of the network and represent it into the graphical front-end. The loading process is similar to the one used for loading a topology. By selecting the option Database→Load Ph Parameters From XML, the following form will appear (Figure 6.8):
In Figure 6.8, it can be seen that the form looks exactly the same as the one used for loading a topology. As before, two XML files have to be loaded for each node and some restrictions are imposed to ensure a proper loading of the information. This time the restriction comes from the fact that the information is loaded directly into the client’s memory and not in the DB, so a particular order must be followed when loading the XML files to bond the information with its corresponding element correctly.

After loading the topology, the user can display the PP information associated to an element by choosing the appropriate option in the popup menu. The user can choose to display the information associated to an OXC, a port (either client or network) of an OXC, a client TE link associated to an OXC or a network TE link. For example, if the user wants to display the PP information of a network TE link, he can perform this by selecting the TE link and choosing the option **TELink Ph Params**. The result of the operation is shown in the Figure 6.9:
Each cell in the tab represents one of the components that compose the TE link and it contains the PP information associated to them. The structure of the cells changes dynamically depending on the element chosen and the component represented as well as the number of cells.

If there is no PP information associated to the chosen element, a warning message will appear informing the fact to the user so he can decide to load it.

If there is no PP information associated to the chosen element, a warning message will appear informing the fact to the user so he can decide to load it.

The operation of loading the PP information can only be executed after having drawn a network.

**6.3.4 ESTABLISH/TEAR-DOWN AN LSP**

This two operations are the ones that give sense to the NMS, because the finality of the NMS, aside from representing a network and allowing to edit and display the information associated to it, is the possibility of managing the soft-permanent connections (LSP) of the network by providing them on demand following the requisites established by the demander. The NMS executes these operations by establishing a dialog with the control plane, responsible for the creation/deletion of the LSP within the network. As mentioned before, an emulated control
plane has been used for testing these functionalities, so before testing them, the control plane must be started up. Figure 6.10 shows a fragment of the initialization log of the control plane:

![Control plane initialization log](image)

After every component of the control plane as been properly started up, a representation of the wavelength used inside every TE link of every node is shown. These representations are showed in the Views entries of the log, one for each wavelength used, displaying the status of each wavelength for each network TE link associated to the node. It can be seen that after starting up the control plane, all the wavelength are marked as free, as one would expect.

To establish an LSP the user has to select the option **LSPs → Establish LSP**. A form will appear, asking about the desired characteristics for the LSP being them source of the LSP, destination of the LSP, bit-rate, and both source and destination client port. This form is shown in Figure 6.11:
Figure 6.11 – Establish LSP form

To test this operation, two LSP has been established: one from Berlin to Leipzig and another one from Leipzig to Nurnberg. The control plane will give each one a different wavelength because the chosen routes have a TE link in common. To distinguish LSP with different wavelength, the NMS high lightens each LSP with a colour depending on the wavelength used, so the user can easily identify which wavelength is being used in every LSP.

The form automatically detects if a client port is already in use for one of the ends of the LSP so it will only allow entering ports that are not in use. In the case that any information entered is erroneous or the operation can not be satisfied, a warning message will appear. Also, if the user tries to establish or tear-down an LSP and the control plane is not running, the NMS will detect it and inform the user.

The result after having established the mentioned LSPs can be seen in Figure 6.12:
The high lightened LSP is the one that goes from Leipzig to Nurnberg. The user can decide which LSP wants to see by selecting the right option in the LSPs menu. Also, he can decide to clean any represented LSP by selecting the option View→Return Default.

The following figures show how the status of the data links of one of the TE links involved into the establishment of the LSPs have changed from unallocated to allocated, both in the control plane (Figure 6.13) and in the NMS (Figure 6.14).
Figure 6.13 – Allocated data links (control plane perspective)

Figure 6.14 – Allocated data links (NMS perspective)
The tearing-down operation is also possible by selecting the option **LSPs→TearDown LSP**. A list of all the established LSP will appear and the user can choose which one wants to tear-down (see Figure 6.15).

![Figure 6.15 – TearDown LSP form](image)

After choosing the LSP that the user wants to tear-down, and if the operation is successful, the status of the corresponding data links will return to unallocated, both in the control plane and the NMS. On the other hand, if the operation is not successful, a message will appear informing the user.
CONCLUSIONS AND FUTURE WORK

The finality of this final project has been the development of a NMS for the DICONET project. It has been achieved by defining the set of objectives it had to accomplish and then studying each one to find the best option possible. The three involved parts (DB, server-side and client-side) have been properly implemented after contemplating multiple design options and choosing the best suited for each part so they can be integrated as a whole application working in harmony.

The implementation of the DB has involved the study of how databases are structured and how to perform operations in them to manipulate the data stored. In addition to, a specially designed schema has been constructed to contain the tables of the DB.

The implementation of both client and server has involved studying how client/server applications work and finding the best way to provide a fast and reliable application. Moreover, a more in depth study for each one of the technologies used (SWT and EJBs) has been necessary to make good use of all the capabilities both technologies can offer so they can be properly integrated in the NMS.

A documenting about the scope and goals of the DICONET project and how ASON/GMPLS networks work (the control plane is based in GMPLS) was strictly necessary to fully understand the demands and needs of the NMS and how they could be satisfied. After having understood them, it was possible to start thinking about the objectives of the project.

Chapter 6 (Use Cases) has reflected, in a very generic way, how is possible to manage a network using the designed NMS by representing and editing the elements that compose the network. The feasibility of the connection between the NMS and the DICONET control plane has also been proven by establishing some LSP and then tearing them down. To conclude, the overall performance of the NMS has been tested successfully through all the tests explained in chapter 6.
Conclusions and Future Work

The end of this final project opens some future development line regarding the NMS: The first one should be the deployment of the NMS as a true client/server application as it was intended from the beginning, not deploying it as the monolithic application used to perform the tests; some modifications and improvements onto the source code may be needed to achieve this goal. The second one should be the integration of the trap listener into the graphical front-end so the user can be aware of any kind of problem involving the network equipment by looking at the representation of the network. The third one could be the permanent storage of the PP information into the DB so it may be available any time and update that information dynamically depending on the network status. For both of these possible development lines, an upgraded version of the control plane that uses monitors to track the network status will be necessary. The fourth one could involve refining the graphical-front end by adding more options or make the actual ones more accessible and intuitive from the user point of view. The nature of these modifications can be though in the moment of developing an upgraded version of the NMS.
AON – All-Optical Network
APS – Automatic Protection Switching
API – Application Programming Interface
ASE – Amplifier Spontaneous Emission
ASON – Automatic Switched Optical Network
ATM – Asynchronous Transfer Mode
BER – Bit-Error Rate
BLSR – Bidirectional Line-Switched Ring
BMP – Bean Managed Persistence
CallC – Call Controller
CC – Connection Controller
CCI – Connection Controller Interface
CD – Chromatic Dispersion
CMP – Container Managed Persistence
CORBA – Common Object Request Broker Architecture
CR-LDP – Constrained-based Routing Label Distribution Protocol
DB – DataBase
DDL – Data Definition Language
DGD – Differential Group Delay
DICONET – Dynamic Impairment Constraint Networking for Transparent Mesh Optical Networks
DML – Data Manipulation Language
DWDM – Dense Wavelength Division Multiplexing
EDFA – Erbium Doped Fibre Amplifier
EJB – Enterprise Java Bean
E-NNI – External Network to Network Interface
FC – Filter Concatenation
FK – Foreign Key
FPGA – Field Programmable Gate Array
FSC – Fibre Switch Capable
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWM</td>
<td>Four Wave Mixing</td>
</tr>
<tr>
<td>GMPLS</td>
<td>Generalized Multiprotocol Label Switching</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GVD</td>
<td>Group Velocity Dispersion</td>
</tr>
<tr>
<td>IA-RWA</td>
<td>Impairment Aware-RWA</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technologies</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>I-NNI</td>
<td>Internal Network to Network Interface</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISI</td>
<td>Inter-Symbol Interference</td>
</tr>
<tr>
<td>IS-IS-TE</td>
<td>Intermediate System to Intermediate System with Traffic Extensions</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>JEE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Messaging Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface</td>
</tr>
<tr>
<td>L2SC</td>
<td>Layer-2 Switch Capable</td>
</tr>
<tr>
<td>LMP</td>
<td>Link Management Protocol</td>
</tr>
<tr>
<td>LRM</td>
<td>Link Resource Manager</td>
</tr>
<tr>
<td>LSC</td>
<td>Lambda Switch Capable</td>
</tr>
<tr>
<td>LSP</td>
<td>Label Switched Path</td>
</tr>
<tr>
<td>LSR</td>
<td>Label Switching Router</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium Access Control</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multiprotocol Label Switching</td>
</tr>
<tr>
<td>NMI-A</td>
<td>Network Management Interface for ASON Control Plane</td>
</tr>
<tr>
<td>NMI-T</td>
<td>Network Management Interface for the Transport Plane</td>
</tr>
<tr>
<td>NMS</td>
<td>Network Management System</td>
</tr>
<tr>
<td>OADM</td>
<td>Optical Add and Drop Multiplexer</td>
</tr>
<tr>
<td>OCC</td>
<td>Optical Connection Controller</td>
</tr>
<tr>
<td>OEO</td>
<td>Optical-Electrical-Optical</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OSNR</td>
<td>Optical Signal to Noise Ratio</td>
</tr>
<tr>
<td>OSPF-TE</td>
<td>Open Shortest Path First with Traffic Extensions</td>
</tr>
</tbody>
</table>
Abbreviations 113

OTI – Object Technology International
OTN – Optical Transport Network
OXC – Optical Cross-Connect
PC – Protocol Controller
PCE – Path Computing Element
PDH – Plesiochronous Digital Hierarchy
PDL – Polarization Dependent Loss
PDU – Protocol Data Unit
PI – Physical Interface
PK – Primary Key
PLI – Physical Layer Impairment
PMD – Polarization Mode Dispersion
PNG – Portable Network Graphics
PP – Physical Parameters
PPD – Physical Parameters Database
PSC – Packet Switch Capable
PXC – Photonic Cross-Connect
QoS – Quality of Service
QoT – Quality of Transmission
RC – Routing Controller
RDBMS – Relational Database Management System
RMI – Remote Method Invocation
ROADM – Reconfigurable OADM
RPC – Remote Procedure Call
RSVP-TE – Resource ReServation Protocol with Traffic Extensions
RWA – Routing and Wavelength Assignment
SAX – Simple API for XML
SBS – Stimulated Brillouin Scattering
SC – Switched Connection
SDH – Synchronous Digital Hierarchy
SDK – Software Development Kit
SNMP – Network Management Protocol
SONET – Synchronous Optical Network
SPC – Soft-Permanent Connection
SPM – Self-Phase Modulation
SQL – Structured Query Language
SRS – Stimulated Raman Scattering
STM – Synchronous Transport Module
SWT – Standard Widget Toolkit
TDM – Time Division Multiplexing
TED – Traffic Engineering Database
UDP – User Datagram Protocol
UNI – User Network Interface
UPSR – Unidirectional Path-Switched Ring
VCI – Virtual Channel Identifier
VoIP – Voice over IP
VPI – Virtual Path Identifier
VPN – Virtual Private Network
WDM – Wavelength Division Multiplexing
XML – Extensible Markup Language
XPM – Cross-Phase Modulation
XT – Crosstalk
REFERENCES


[17] XDoclet Team, “XDoclet User Documentation”.


[22] Java SNMP Stack API, http://snmp.westhawk.co.uk


