MASTER THESIS

TITLE: Design and Implementation of a control plane for a L2 virtualized network

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Roberto Machado Calvo

DIRECTOR: Cristina Cervelló i Pastor

DATE: July 14 th 2009

Title: Design and Implementation of a control plane for a L2 virtualized network

Author: Roberto Machado Calvo

Director: Cristina Cervelló Pastor

Date: July, 14th 2009

Overview

This Master Thesis explains the implementation and design of a Layer 2 resources management tool, able to control the resources of a network and configure the devices across it in order to offer a user an end-to-end resource.

The main objective is to implement an own tool in order to fulfil the needs of control plane focused in a network similar to the existing for FEDERICA project, but with the previous knowledge of the existing control tools for the purpose of resource management.

With this purpose, the Thesis includes a previous analysis of the requirements of a control plane (or its tool implementation), that will be used as a base to study other existing tools and to design ours.

A study of the state of the art concerning the resource management tools already working; comparing them with the requirements has been done and included in the Annexes of this thesis. None of these tools fulfil all the requirements, so the decision was to design a specific tool for FEDERICA.

Also the resources and devices to be managed have been studied and are explained in the corresponding part of this Master Thesis. A possible real scenario has been described, and a solution is proposed in order to implement end-to-end Q-in-Q in the real FEDERICA scenario, taking into account different actors as the switches and the Virtual machines servers.

Finally, the tool that will be able to manage the resources across the network is introduced and explained, from its design to the implementation, from its basic structure to the different extension modules, such as a graphical user interface and services to interact with the application. The Web services and user interface explanation is described in the annexes due to its extension. The design process has been done taking into account the previous requirements trying to fulfil many of them.

INDEX

INTRODUCTION

The socialization of the data networks, the expansion of their use, and the new applications and traffic types over them, are making that current networks are reaching their maximum capacity.

New business models over the networks, new paradigms and new architectures will succeed the current Internet architecture. Currently, a large number of researchers are working on these topics focused on making a step forward into the networks of the future.

In this context, FEDERICA project was started to offer these researchers the possibility to test new protocols in a working network and interconnect their test frames with real scenarios, other researchers' networks or even the current Internet.

FEDERICA offers different isolated slices to the researchers. These slices are based on the virtualization of a set of available resources, and the correct management of these resources will ensure the success and avoid the missconfiguration of the slices.

Is in this scenario where a management tool acquires importance. We have developed an embryonic tool to manage and control the layer 2 resources in the network in centralized way. Resources can be created, deleted, monitored and configured from this tool, which will be a part of a further complete framework for the FEDERICA environment.

The project has been structured as it follows. An introduction to FEDERICA project has been done in order to locate the reader in the limits of the project. A previous study of the already existing tools has been done and included in the annexes. Also an analysis of the requirements for the project has been done and it is described in the chapter 2. The devices to be controlled have been studied under the optics of the project needs, such as VLAN configuration, etc. Moreover, in a vision of a further integration with a complete management environment, a Q-in-Q deployment proposal has been also included, in order to offer a possible configuration of Q-in-Q in a scenario where virtual machines servers are present. Finally this knowledge has been translated into the tool. A full chapter explains how the tool is designed and how it has been implemented.

Concerning environmental studies this master thesis purpose management tools for a virtualization environment. The controlled switches interfaces are virtualized in order to be shared between different users traffic. This kind of process, and virtualization in general is a good way to improve the energy efficiency, because the performance of each one of the devices is increased, avoiding a useless waste of the resources and even reducing the number of devices needed in a network deployment.

1. FEDERICA PROJECT

1.1 Introduction

This project is framed in the context of the FEDERICA European project.
FEDERICA is a two-and-a-half-year EC 7th Framework project to implement a versatile experimental network infrastructure for trialling new networking technologies. This infrastructure is intended to be agnostic as to the type of protocols, services and applications that may be trialled, whilst allowing disruptive experiments to be undertaken. The aim is to develop mechanisms that will allow such experiments to be run on *slices* of the FEDERICA physical infrastructure without adversely affecting other slices.

1.2 Project summary

The main goal of the FEDERICA project is to expand the services provided by the National Research and Education network to support research experiments and projects on new Internet architectures and protocols. FEDERICA will create a versatile, scalable, European wide and "technology agnostic" network infrastructure, open to disruptive experiments, in parallel, but independent from production networks. The infrastructure will benefit from long-distance circuits and support contributions from partners, exploiting the existing NREN and GÉANT2 networks services therein.

FEDERICA adopts the paradigm that "the infrastructure is the network". The traditional model of core and edges is replaced by a quasi-homogeneous network, where each node is defined according to its functions, rather than through its position and the logical functions and global topology can be easily changed. The infrastructure will feature the novel introduction of computing resource as network elements, capable of virtualising services, from routers to measurement or end nodes.

The FEDERICA infrastructure will be built in phases, starting with the advanced tools and research results available from the GÉANT2 project and partners. Its initial phase, currently operational, offers the possibility to create virtual networks, or "slices", in which experiments can run in parallel.

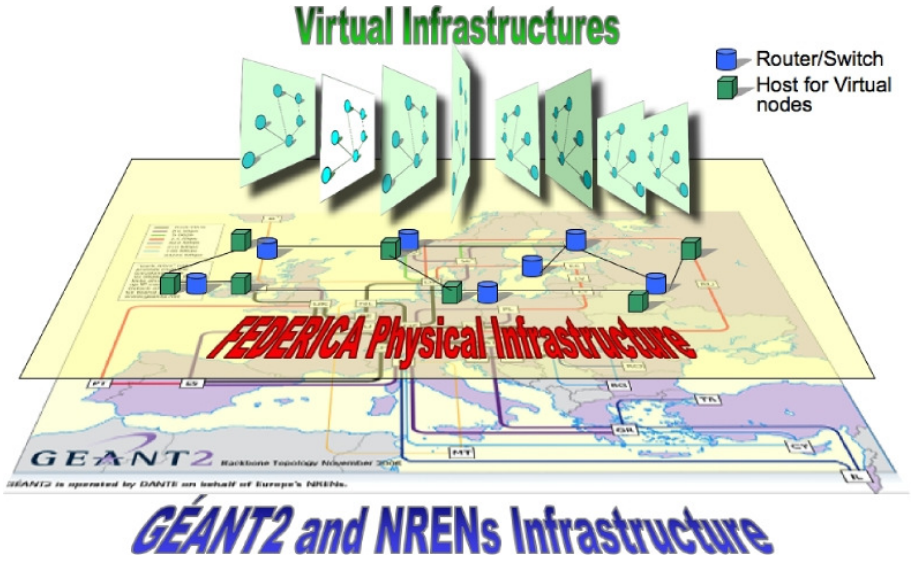

Figure 1-1. Infrastructures layers

The virtualization will be realised initially using VLANs and Layer 2 transport MPLS capabilities to offer Ethernet based virtual circuits and attached computing resources. The infrastructure will be connected to the research production networks and thus be reachable from all locations. It is will be possible for other infrastructures to connect to FEDERICA to enhance its capabilities or to perform joined experiments.

The FEDERICA infrastructure may grow according to users/ needs and available resources. It would be possible with relative ease to extend it to specific user premises using virtual circuit technologies with Layer 2 emulation.

The FEDERICA project will actively support and facilitate technical discussion amongst specialists, and disseminate knowledge and NREN experience related to the requirements of the users.

To develop the FEDERICA potential, internal research activities and experiments have been planned by the partners, in particular to:

- Understand and produce initial solutions on management and control of distributed, parallel, virtual networks which may communicate between them and with the general Internet.
- Develop experience and a model for managing and using virtual infrastructures as a combination of networks and systems. Currently, this type of management and control is in its infancy: pro-active tools based on systems are needed for monitoring and services.
- Enable the graceful implementation of new inter-domain service layer.

1.3 FEDERICA objectives

The FEDERICA project has the following main objectives, divided into 3 areas:

1. Support

- Support research experiments on new Internet architectures and protocols
    - Create a versatile, scalable, European wide "technology agnostic" infrastructure, separated from the production networks, but with the possibility to interoperate
    - Open to host researchers' hardware and applications.
    - Adopt the paradigm that "the infrastructure ***is*** the network". The traditional model of core and edges is replaced by a quasi-homogeneous network, where each node is defined according to its functions, rather than through its position. The logical topology and node functions can be easily changed. The infrastructure can be adapted to whatever the user wants it to be.
    - Simultaneous use by more than one research group.
- Make infrastructure facilities available for Call 1 "Network of the Future" and Call 2 "FIRE" projects.
- Exploit the excellent networking facilities that are available in Europe; especially the existing NREN and GEANT networks and available technologies and services therein.

2. Networking

- Facilitate technical discussion amongst specialist, in particular when based on experimental results and disseminate knowledge and NREN experience of meeting users' requirements.
- Provide preliminary information and results for the next generation of the NREN networks, and link with GEANT2 and as a precursor experimental phase for GEANT3
- Contribute with real test case and results to standardization bodies, e.g. IETF, ITU-T, OIF, IPsphere.

3. Research

- Understand and produce initial solutions on management and control of distributed, parallel, virtual networks which may communicate between them and with the general Internet.
- Develop experience and a model for managing and using virtual infrastructures as a combination of networks and systems. Currently, this type of management and control is in its infancy: pro-active tools based on systems are needed for monitoring and services.
- The infrastructure will be open to any "application" type (application can be a test)
- Use, test and advance the tools and services being developed by GN2, NRENs and partners, in particular for end-to-end services
- Enable the graceful implementation of new inter-domain service layer.

1.4 FEDERICA basic infrastructure

The infrastructure will be based on a set of dedicated physical network resources (wavelengths, circuits, nodes) and a set of virtual connections created from the NREN and GEANT2 production networks either as physical circuits or as logical circuits.

Each Point of Presence (PoP) of the infrastructure may contain computing and networking resources in the form of physical network hardware (switches, routers, agnostic switches) and computing elements (high end commodity servers with virtualization software). Each PoP can host users' hardware and/or set-up a connection to an infrastructure external to FEDERICA.

A computing element is a node capable of hosting virtual computing resources, which can be e.g. endnodes, virtual routers or measurement boxes. The users can than prepare the virtual node at his premise and just require that the FEDERICA infrastructure runs one or more instances of it.

The infrastructure blurs the distinction between core and border nodes, placing the emphasis on the services offered. FEDERICA users will access a service, rather than a specific node and the access to FEDERICA can be through any of its point of presence. It is not needed that all PoPs have the same configuration, on the contrary, diversity in network technologies and computing resources may offer a richer infrastructure.

The infrastructure will be controlled and managed by the consortium as one single domain, connected to the global Internet, both IPv4 and IPv6, with its own address space and control mechanisms.

This pan-European domain may physically be extended to reach other similar infrastructure or specific user labs.

The infrastructure will be capable of creating virtual networks or "slices" through technologies like MPLS or using lower layers up to a physical partition. Each virtual infrastructure can be a separate domain, with its own architecture, routing schemes, end-to-end services. FEDERICA will host concurrent slices and let them operate independently or connected to each other, according to users' requests. FEDERICA will develop a tool-bench which will leverage existing results in virtualization and network control and extend them to allow a fine-grained multi-domain management and monitoring.

The performance of a network resource and of the global infrastructure is important, but it is not considered crucial. The most important characteristics are its flexibility and potential for accepting and making conceptual changes (customisable or programmable). It is expected that most of the active elements will be based on Open Source operating systems and software running on virtual machines (through e.g. VMWare, Xen).

The use of Open Source in the project and the development of Open Source solutions is considered fundamental.

The network status will be permanently monitored and controlled. Performance data will be used for further analysis and studies of the behaviour of particular network in a virtual slice. According to such evaluations the network will be tuned to offer the most adequate transport characteristics and conditions for users' experiments.

FEDERICA is open to support research in sensors, wireless, mobility, autonomic communications, ad-hoc networks areas. It will provide the interconnecting "glue" (based on knowledge of the aggregated traffic), but does not plan to implement directly such networks types. It is also important to underline that FEDERICA is capable of hosting the testing of application of any type. The infrastructure is in fact built to avoid any form of filtering or preference for any architectural principles at the application layer.

1.5 Network topology

FEDERICA's network infrastructure is the basis for providing project virtualization services to researchers and end users. The infrastructure is based on 13 PoPs hosted by different NRENs.

The FEDERICA PoPs are interconnected by dedicated 1 Gb/s channels, which are provisioned via DANTE's GÉANT+ service. The design phase of the physical infrastructure addressed, the emphasis was placed on the reliability, resiliency and load balancing of the network, based on the GÉANT2 infrastructure and resources available at project partners locations.

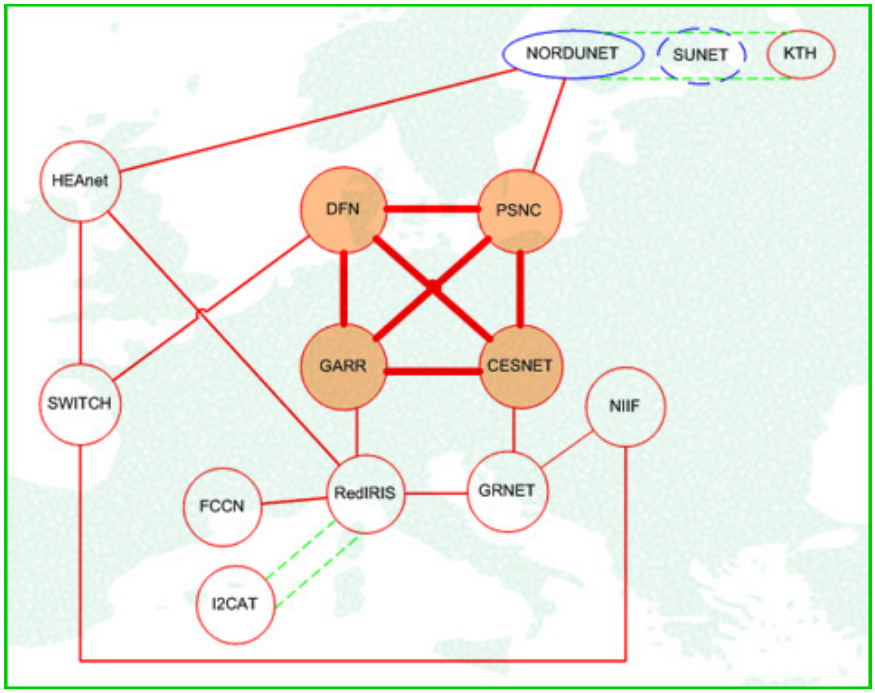

Figure 1-2. Federica network topology

1.6 FEDERICA architecture

FEDERICA is able to provide on slicing principles, and using dedicated lines for connecting Physical Nodes (PN). Within boundaries of each PN multiple instances of Virtual Nodes (VN) may be created, including virtual machines based on various operating systems or emulating L2/L3 network equipment.
Each virtual machine acts independently, and can run different applications at the same time. Connections between VNs are made via Virtual Paths (VP) created over the physical networking infrastructure, using techniques such as tunnelling.

All VNs dedicated to the same purpose, along with the VPs connecting them together, represent one independent virtual infrastructure called a slice. A slice contains all distributed networking resources (routers, switches, etc.) and also end point virtual machines. All of these resources are seen by the end users as physical infrastructure and the virtualization layer is hidden from them. There is no theoretical limit to number of slices that can be created, and there is no limit on number of applications (of any type) running inside each slice.

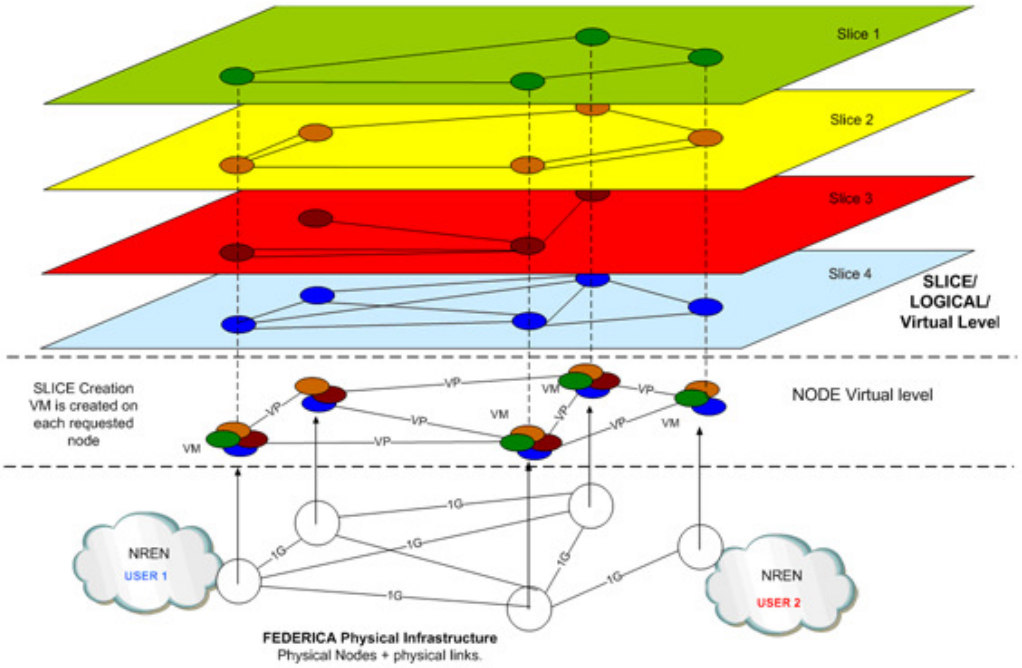

Figure 1-3. FEDERICA multidimensional architecture

The FEDERICA architecture consists of three parts:

- At the lowest level, there is a fixed physical infrastructure connecting Physical Nodes (PN) (Juniper MX/EX boxes and V-nodes) and to users outside the FEDERICA Network.
- Above the physical infrastructure is a virtual layer where VNs are created and connected with VPs according to user requirements.
- The third layer includes slices provided to users, which are seen as physical infrastructure.

2. NETWORK REQUIREMENTS

2.1 Introduction

In this section some basic network requirements are going to be listed. This list will be useful in order to compare the different exiting tools under the same focus.

This is the list of the main requirements, and will be checked for each one of the studied tools:

- Virtualization requirements
- Operational requirements
- Control plane requirements
    - Layer 2 requirements
- End user requirements
- Monitoring requirements
- User-friendship requirements

2.2 Virtualization requirements

These requirements are referred to the partition property of a physical resource into different logical resources.

The main requirements are:

- Representation and control of physical resources.
- Resource partitioning (slice):
    - Switch partitioning or slicing.

The resources to be managed in a layer 2 test bed are going to be a series of switches. A first approach to this switch slicing is the configuration of VLANs.

2.3 Operational requirements

The operational requirements refer to the general characteristics of the infrastructure and the tool bench. These requirements are:

- Isolation between different slices.
- Inter-slice communication capability.
- Possibility to apply changes to the network manually.
- Possibility to modify the slice after the original request.
- Repeatability of the experiments during the lifetime of an experiment.

2.4 Control plane requirements

The control plane of the tool would interact with the different devices along the network, trying to offer end-to-end connectivity.

- Multiple devices configuration.
- Homogeneous end-to-end network connection.
- Allow specified layer 2 requirements.

These layer 2 requirements are the following:

- Managing VLANs.
  - Create/Delete VLANs.
  - Enable/Disable VLANs.
  - Displaying VLANs.
- Adding static members to VLANs.
  - VLAN – port assignment.
- Configuring VLAN trunks.
- Q-in-Q VLAN tagging.
- Configuring Private VLANs.
- Configuring VLAN behavior for interfaces (GVRP).
- Configuring Protocol-Based VLANs.

2.5 End user requirements

The end user requirements are the following:

- Sub-allocate a set of virtual resources to the end user with the control associated to its user profile (full or limited control) during a limited period of time.
- Choose a specific topology under the user requirements.

2.6 Monitoring requirements

The monitoring requirements are the following:

- Visualization of the current configuration of the infrastructure.
- Get notifications from the devices.

2.7 User-friendship requirements

These are the requirements:

- Software mechanism to represent the various physical or logical devices.
- Visualization of the configuration.
- Easy customization of the request.

3. JUNOS software in EX-3200 switch

3.1 Basic concepts

3.1.1 JUNOS architecture

JUNOS architecture is based in a two plane system. Every configuration that is introduced to the switch, via CLI, Netconf, etc., is stored managed by a control plane. This control plane controls the forwarding plane, where the packets are managed and forwarded in an autonomous way from the control plane.

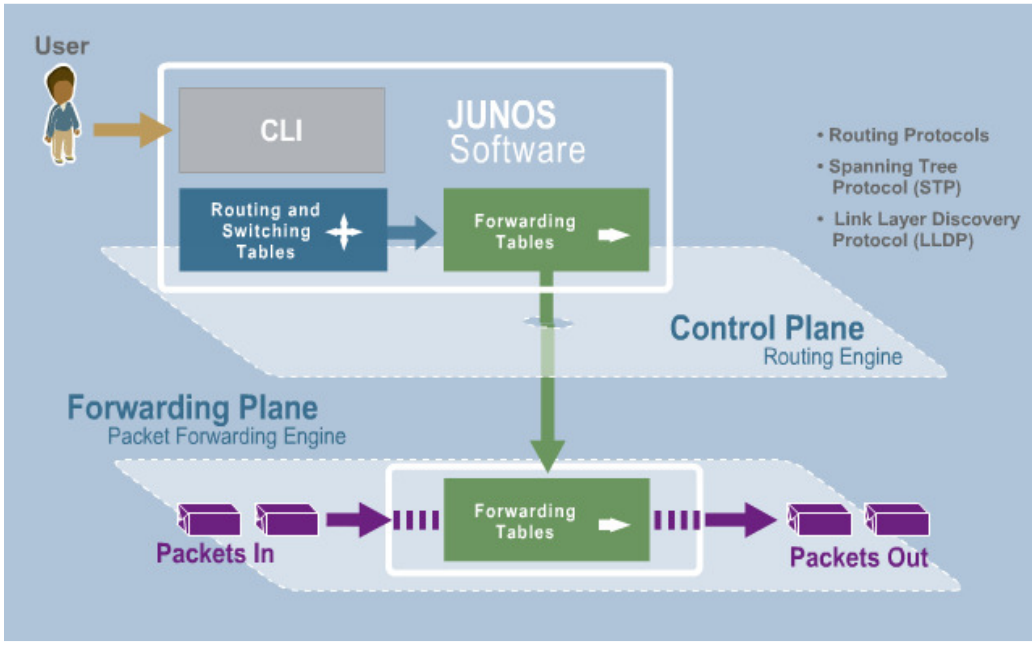

Figure 3-1. JUNOS Architecture.

One of the main advantages of this structure is the independence in the performance of each one of the layers. If the control plane is busy, or it is being updated for instance, the forwarding plane works independently from it, so it takes no effect to this forwarding plane.

3.1.2 Managing the switch

JUNOS software offers several ways to manage the devices:

- Console port
- Telnet
- SSH
- J-Web
- SNMP
- JUNOScript API
- NETCONF API

- NetScreen Security Manager

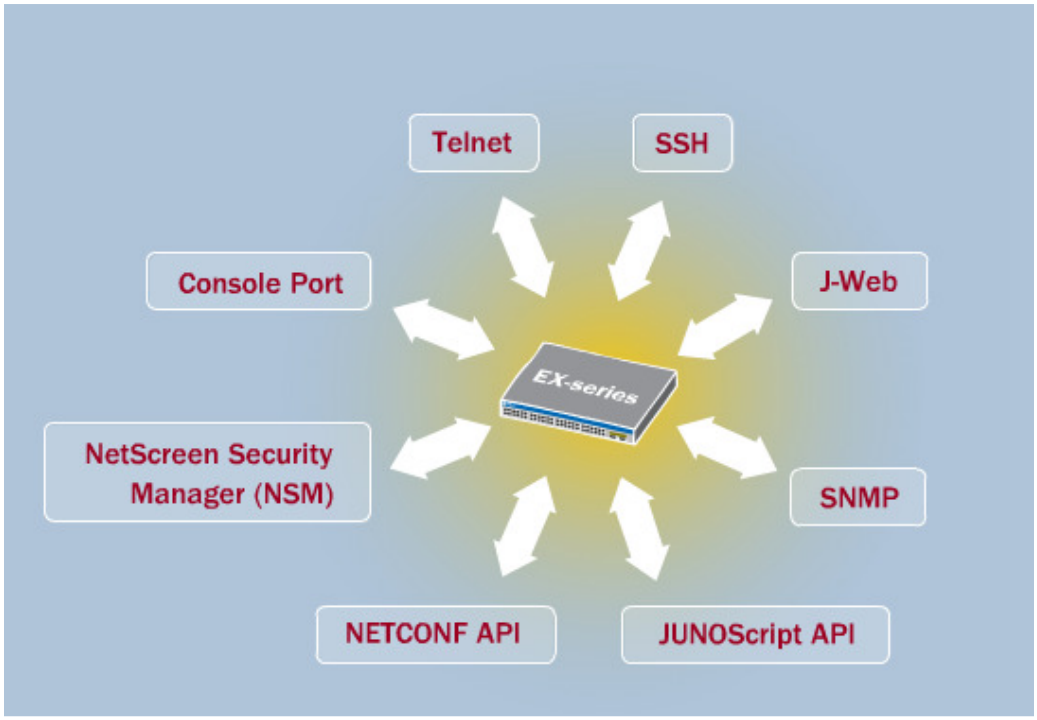

Figure 3-2. Switch configuration options

In Console Port, Telnet and SSH modes, the switch is controlled through the CLI (Command line interface). The J-Web system is a GUI (graphical user interface).

Due to the characteristics of our project, we have focused in two of them: CLI and NETCONF API.

3.1.3 Command Line Interface

The CLI is Juniper specific command shell that runs on top of a UNIX-based operative system kernel and also offering some UNIX tools.

The CLI has two modes, operational and configuration mode. The operational mode is used to monitor and troubleshoot switch hardware and software and network connectivity.

In the configuration mode, all properties of the JUNOS software are defined: Interfaces, VLANs, user access, etc. must be defined working in this mode. To enter in the configuration mode, the command `configure` must be introduced.

```
user@switch> configure
```

An important fact to take into account is that every change done in the configuration mode is done to the candidate configuration file. The candidate configuration allows changing the configuration of the switch, without doing any operational change to the running operating configuration (called active configuration).

To activate the candidate configuration, the commit command must be introduced. The system will detect some bad configurations and it will not commit the changes until a correct configuration is done.

```
user@switch> commit
```

3.1.4 Netconf

Netconf is an API based in XML (Extensible markup language). Client applications use this API to exchange information with the Netconf server running on the switch. The API provides a way to display, edit and commit configuration statements. The operations defined in the API are equivalent to the configuration commands in the command line interface (CLI).

The Junos XML API represents the JUNOS configuration statements and operational mode commands in XML. The operations of the NETCONF API respond to the JUNOS XML API tags.

Working with Netconf and Junos XML APIs offers some advantages. One of them is that both are programmatic interfaces, and they fully document all options for every supported JUNOS operational request and all elements in every configuration statement.

Other advantage is the simplicity to understand the content and structure of the XML-tagged data sets, due to the combination of the meaningful tag names and the structural rules in a DTD (Document type definition).

Other of the main advantages of Netconf is that it is able to get the status of the switch, and parse it to the program in a easier way than CLI would do.

Language description

To map commands to JUNOS XML Tag Elements the JUNOS API defines tag-element equivalents for many of those commands in CLI operational mode.

Many CLI commands have options that identify the object that the command affects or reports about, distinguishing the object from other objects of the same type. Moreover, many commands include options that have a fixed form which

specify the amount of detail to include in the output. JUNOS XML API usually maps such an option to an empty tag whose name is the option name.

Resources request

The request message sent to the devices is the following:

```
<rpc>
     <get-config>
           <source>
                <candidate/>
           </source>
           <filter>
                <configuration>
                     <interfaces></interfaces>
                     <vlans></vlans>
                </configuration>
           </filter>
     </get-config>
</rpc>
```

With the specification of a filter as is shown below, we limit the information asked to the switch to the elements included within the filer. In this case, only interfaces and VLANs information will be retrieved.

```
           <filter>
                <configuration>
                     <interfaces></interfaces>
                     <vlans></vlans>
                </configuration>
           </filter>
```

Interfaces description

Once the switch sends the configuration information, we can obtain the resources description from the tag hierarchy as it is explained below.
The interfaces description is done as shown in the following code:

```
<interfaces>
        <interface>
          <name>ge-0/0/0</name>
               <unit>
                  <name>0</name>
                  <description></description>
                  <family>
                     <ethernet-switching>
                        <port-mode>trunk</port-mode>
                        <vlan-id>120</vlan-id>
                     </ethernet-switching>
                  </family>
               </unit>
        </interface>
</interfaces>
```

All the described interfaces are encapsulated by the `<interfaces></interfaces>` tags. Inside these tags, each interface is described, surrounded by the `<interface></interface>` tags. Inside these tags the interface properties are described by the following tags.

```
<name></name>
```

Inside each interface, units are described, specified among `<unit></unit>` tags. The properties of each unit are described by the following tabs:

```
<name></name>
<description></description>
<family></family>
```

Inside family tabs, more family-specific properties can be described.
For ethernet-switching family:

```
<ethernet-switching>
        <port-mode>trunk</port-mode>
        <vlan-id>120</vlan-id>
</ethernet-switching>
```

where `port-mode` can be configured as access or trunk.

For inet family, there are some configurable properties such as the address or the MTU.

```
<inet>
        <address></address>
        <mtu></mtu>
</inet>
```

VLANs description

VLANs are described by the JUNOS software as is shown below:

```
<vlans>
        <vlan>
             <name>VLANA</name>
             <description>testing</description>
             <vlan-id>234</vlan-id>
             <interface>
                 <name>ge-0/0/0.0</name>
             </interface>
             <interface>
                 <name>ge-0/0/19.0</name>
             </interface>
        </vlan>
</vlans>
```

VLAN properties are specified among the `<vlan></vlan>` tags. Each one of them is described inside the corresponding labels. The more common are:

```
<name> </name>
<description> </description>
<vlan-id></vlan-id>
<interface></interface>
```

CLI – Netconf example

The second one is a comparison among the ASCII and XML-tagged versions of output information from the switch. The formatted ASCII follows:

```
Physical interface: fxp0, Enabled, Physical link is Up
Interface index: 4, SNMP ifIndex: 3
```

And this is the XML-tagged version:

```
<interface>
      <name>fxp0</name>
      <admin-status>enabled</admin-status>
      <operational-status>up</operational-status>
      <index>4</index>
      <snmp-index>3</snmp-index>
</interface>
```

As can be seen in this comparison, the structure is much clearer in the XML-tagged format. In addition, any provided information is surrounded by a XML tag with the meaning of the field.

3.2 Interfaces

There are two types of interfaces in the EX-3200 switch: network and special interfaces.

Figure 3-3. Juniper EX-3200 switch

3.2.1 Network interfaces

Network interfaces connect to the network and carry network traffic. The following types of network interfaces are supported:

- LAN access interfaces: The EX-3200 switch provide 24 network ports that can be used to connect laptops, computers, servers, etc. to the network. This access mode is the port default configuration.
- Trunk interfaces: An access switch connected to a distribution switch must be explicitly configured as a trunk port. Also the distribution switch interface connected to the access switch must be configured as trunk.
- Power over Ethernet (PoE) interfaces: This kind of ports are able to supply power to the devices connected, such as VoIP phones, wireless access points, video cameras, etc. at the same time that are being connected to the network.
- Aggregated Ethernet interfaces: Different Ethernet interfaces can be grouped at the physical layer to form a single link layer interface. This is useful to balance traffic and increase uplink bandwidth.

3.2.2 Special interfaces

Special interfaces give a different use to access the switch. These are the special interfaces provided by the switch:

- Management interface: This interface provides an out-of-band method for connecting to the switch. To be able to use me0 as a management port, its logical port me0.0 must be configured with a valid IP address. Then, this interface can be connected using SSH and telnet. Also SNMP (Single network Management Protocol) can use this interface to gather statistics from the switch.
- Console port: This is a serial port, labelled console, with the purpose of connecting tty-type terminals to the switch using standard PC-type tty cables. It has not any physical address or IP address associated.
- Loopback: This software-only interface provides a stable and consistent interface and IP address on the switch.

3.2.3 General information

To understand and configure the different interfaces, it is useful to know the naming convention used for this switch. Interfaces in JUNOS software are specified as follows:

```
type-fpc / pic /port
```

where:

- type: EX-series interfaces use the following media types:

- ge – Gigabit Ethernet interface
- xe – 10 Gigabit Ethernet interface
- fe – Fast Ethernet interface

- fpc: Ex-series use the following convention for the FPC portion of interface names:
  - On EX 3200 FPC number portion is always 0.

- pic: The physical Interface Card (PIC) follows this convention:
  - On EX-3200 the PIC number is 0 for all built-in interfaces (interfaces that are not on an uplink module).
  - On uplink modules, the PIC number is 1.

- port: EX-series interfaces use the following convention for port numbers:
  - Built-in network ports are numbered from left to right. If the model have to row of ports, the ports on the top row start with 0 followed by the remaining even-numbered ports, and the ports on the bottom row start with 1 followed by the remaining odd-numbered ports.

The logical unit part of an interface can be a number from 0 through 16384, and it is separated of the port name by a period. When aggregated Ethernet interfaces are configured, logical interfaces are configured, and called bundle or LAG. Each LAG is able to include up to eight Ethernet interfaces

3.3 Virtual LANs

In this section we are going to explain how VLANs work in EX-3200 switch and how to manage them.

3.3.1 How VLANs work

In the EX-3200 switch Virtual LANs (VLANs) are implemented by the protocol IEEE 802.1Q. This protocol tags the Ethernet frames, using a frame-internal field for tagging.

802.1Q adds a 32-bit field between the source MAC Address and the EtherType/Length fields of the frame.

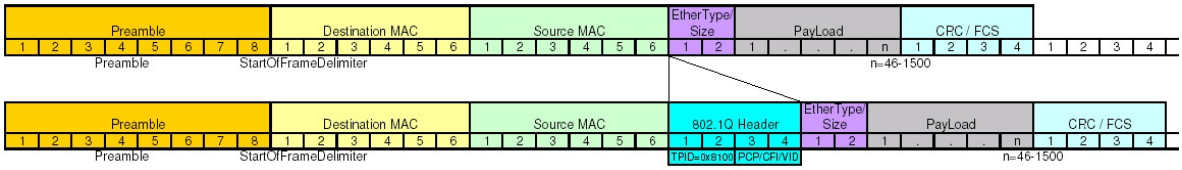

Figure 3-4. VLAN tagging.

As a result, switched traffic from a VLAN will not be seen by any other different VLAN member.

3.3.2 How to configure VLANs in the switch

The first step to configure a VLAN in the switch is to create a VLAN, and assign a VLAN ID to this VLAN. Afterwards, this VLAN must be added to each one of the ports, which will be members of it.

In this example two ports are configured and added to the created vlan Federica-1. The physical interfaces are ge-0/0/1 and ge-0/0/2 and we have configured the logical interface 0 (unit 0) of each one of them.

If this configuration is going to be done sequentially through CLI, this should be the process:

```
set vlans federica1 vlan-id 100
set interfaces ge-0/0/1 unit 0 description “LAN port”
set interfaces ge-0/0/1 unit 0 family ethernet-switching vlan members
federica1
set interfaces ge-0/0/2 unit 0 description “Router port”
set interfaces ge-0/0/2 unit 0 family ethernet-switching vlan members
federica1
```

Similar instructions can be sent to delete a VLAN, or to disassociate a VLAN from a port. To configure a port to trunk mode the following instructions must be introduced:

```
set interfaces ge-0/0/1 unit 0 ethernet-switching port-mode trunk
set interfaces ge-0/0/1 unit 0 ethernet-switching vlan members
federica1
```

A similar configuration can be done through Netconf. First of all, the VLAN is created, and then, ports are configured and assigned.

```
<configuration>
   <vlans>
      <vlan>
```

```
            <name>federica1</name>
            <tag>10</tag>
            <description>Federica vlan</description>
        </vlan>
    </vlans>
        <interfaces>
            <interface>
                <name>ge-0/0/1</name>
                    <unit>
                        <name>0</name>
                        <desciption>Unit0</desciption>
                    <family>
                            <ethernet-switching>
                                <port-mode>trunk</port-mode>
                            </ethernet-switching>
                        </family>
                        <vlan>
                            <name>federica1</name>
                        </vlan>
                    </unit>
            </interface>
            <interface>
                <name>ge-0/0/2</name>
                    <unit>
                        <name>0</name>
                        <desciption>Unit0</desciption>
                    <family>
                            <ethernet-switching>
                                <port-mode>trunk</port-mode>
                            </ethernet-switching>
                        </family>
                    </unit>
            </interface>
        </interfaces>
</configuration>
```

3.4 Q-in-Q

3.4.1 Introduction

Q-in-Q tunnelling allows to service providers on Ethernet access networks to extend a Layer 2 Ethernet connection between two customer sites. Q-in-Q is also used to segregate customer traffic into different VLANs. This is done by adding a second layer of 802.1q tags.

The utility of this service is made clear when different users have overlapping VLAN IDs. Then these client VLANs (C-VLANs) can be preceded by Service VLANs (S-VLANs) tags.

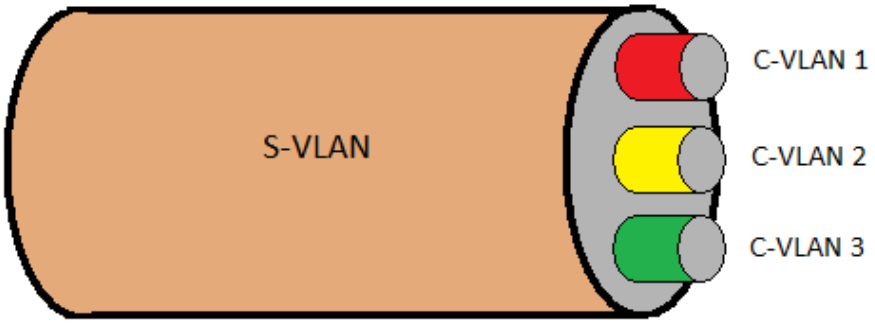

Figure 3-5. Q-in-Q encapsulation.

We have worked in how to deploy the Q-in-Q capabilities into a real scenario for this project. This (Virtual Machine Servers connected to switches). This study has been included in the following chapter and it is a key issue about how Q-in-Q service must be deployed in the network.

For the moment, the tool is not able to configure Q-in-Q, but how to implement and deploy it is well known. This will be one of the next steps to go further in the project.

3.4.2 How Q-in-Q works

A user or customer can configure a VLAN. This VLAN is tagged with a 802.1Q tag as usual. Once a packet travels from this C-VLAN to the service-provider VLAN (S-VLAN), an additional tag is added preceding the C-VLAN tag. This way, the customer 802.1Q tag remains, and it is transmitted transparently, going through the service provider network.

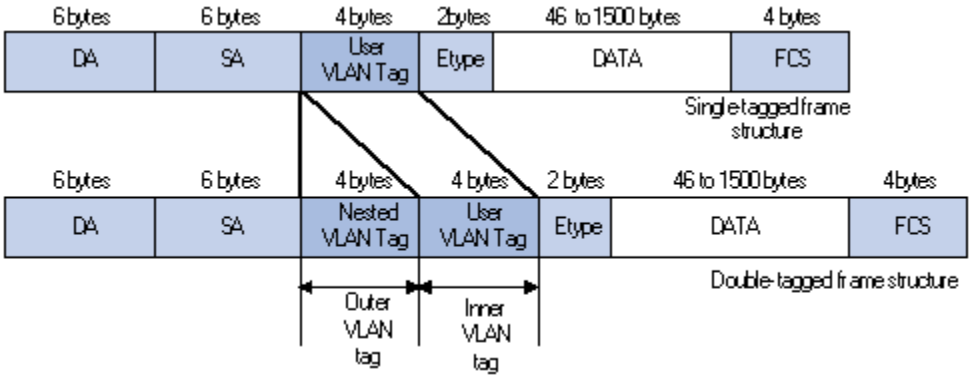

Figure 3-6. Q-in-Q tagging.

When Q-in-Q is enabled, trunk interfaces are assumed to be part of the service provider network, and the access interfaces will be facing the customers.

Multiple C-VLAN can be mapped to a S-VLAN (N:1). As the C-VLAN and S-VLAN are unique, a VLAN-ID can be found in S-VLAN and C-VLAN. For instance, we can have a C-VLAN 1234 and a S-VLAN 1234.

In the case of the EX-3200 switch, the S-VLAN is added on egress (in the access interfaces) for incoming packets. These access interfaces can receive both tagged and untagged frames. Then, when the packets leave the S-VLAN in downstream direction, the extra 802.1Q tag is removed, and the original customer tag remains in the packets.

In a Q-in-Q deployment, customer packets from downstream interfaces are transported without any changes to the source and destination MAC addresses. In large scale networks, the service provider switches might learn a huge number of MAC addresses. In order to avoid possible slow performance in the switches, disabling MAC address learning in these switches would be recommended.

There are some limitations of the implementation of Q-in-Q in the EX-3200 switch. The C-VLAN tag cannot be added on egress for incoming untagged packets or be removed in the downstream direction. Q-in-Q does not support IGMP snooping or most access port security features.

Customer policing or outgoing shaping and limiting cannot be applied per-VLAN in Q-in-Q.

3.4.3 How to configure Q-in-Q in the switch

To configure Q-in-Q in the switches, the necessary customer VLANs (C-VLANs) must have been previously created and configured.

The desired configuration is shown in the following table:

Interface	Description
ge-0/0/15.0	Tagged S-VLAN trunk port
ge-0/0/16.0	Untagged customer-facing access port
ge-0/0/17.0	Untagged customer-facing access port
ge-0/0/18.0	Tagged S-VLAN trunk port

Table 1. Possible interface configuration.

The commands to configure the Q-in-Q tunnelling are the following:

1. S-VLAN VLAN ID setting:

```
set vlans qinqvlan vlan-id 1234
```

2. Enable Q-in-Q tunnelling and customer VLAN ranges specification:

```
set vlans qinqvlan dot1q-tunneling customer-vlans 1-100
set vlans qinqvlan dot1q-tunneling customer-vlans 201-300
```

3. Port mode and VLAN information setting:

```
set interfaces ge-0/0/15 unit 0 family ethernet-switching port-mode trunk
set interfaces ge-0/0/15 unit 0 family ethernet-switching vlan members 1234
set interfaces ge-0/0/16 unit 0 family ethernet-switching port-mode access
set interfaces ge-0/0/16 unit 0 family ethernet-switching vlan members 1234
set interfaces ge-0/0/17 unit 0 family ethernet-switching port-mode access
set interfaces ge-0/0/17 unit 0 family ethernet-switching vlan members 1234
set interfaces ge-0/0/18 unit 0 family ethernet-switching port-mode trunk
set interfaces ge-0/0/18 unit 0 family ethernet-switching vlan members 1234
```

4. Q-in-Q Ethertype value setting:

```
set ethernet-switching-options dot1q-tunneling ether-type 0x9100
```

Now we can see the changes done:

```
user@switch> show configuration vlans qinqvlan
vlan-id 1234;
      dot1q-tunneling {
      customer-vlans [ 1-100 201-300 ];
}
```

To verify the correct configuration, the `show vlans` command must be used:

```
user@switch> show vlans qinqvlan extensive
VLAN: qinqvlan, Created at: Thu Sep 18 07:17:53 2008
802.1Q Tag: 1234, Internal index: 18, Admin State: Enabled, Origin:
Static
Dot1q Tunneling Status: Enabled
Customer VLAN ranges:
                     1-100
                     201-300
Protocol: Port Mode
Number of interfaces: Tagged 2 (Active = 0), Untagged 4 (Active = 0)
      ge-0/0/15.0, tagged, trunk
      ge-0/0/18.0, tagged, trunk
      ge-0/0/16.0, untagged, access
      ge-0/0/17.0, untagged, access
```

3.5 CoS

3.5.1 How CoS works

EX-3200 switches provide some mechanisms to implement CoS. These mechanisms are policers, classifiers, forwarding classes, tail drop profiles and schedulers.
For the moment any of these technologies has been implemented for the tool yet, but it is in the roadmap for further steps.

Policers
Policers limit traffic of a certain class to a specified bandwidth and burst size. The policers can be associated with input interfaces. Packets exceeding the policer limits can be discarded.

Classifiers
Packet classification associates incoming packets with a CoS servicing level. Classifiers associate packets with a forwarding class and loss priority, assigning packets to output queues.

Two general types of classifiers are supported:

- CoS value traffic classifiers: Based on the CoS value in the packet header.
- Multifield traffic classifiers: Based in multiple fields in the packet, as source and destination addresses.

Forwarding classes
Forwarding classes group the packets for transmission. Packets are assigned to output queues based on the forwarding classes. Forwarding, scheduling, and marking policies applied to the packets are affected by the forwarding classes as packets transit switch.

There are four main forwarding classes, although the switches are able to manage up to 16 in order to improve the granularity of the classification:

- Best effort
- Assured forwarding
- Expedited forwarding
- Network control

Tail drop profiles
Tail drop profile is a mechanism for congestion management. This mechanism allows starting dropping the incoming packets when the queue buffers get full (percentage of the queue is full).

The queue fullness defines the delay-buffer bandwidth, which provides packet buffer space to absorb burst traffic up to the specified duration of delay. Once the specified delay buffer becomes full, packets with 100 percent drop probability are dropped from the tail of the buffer (drop probability cannot be modified for this switch).

Schedulers
Schedulers are used to define the output queues properties. These properties are:

- Amount of interface bandwidth assigned to a queue.
- The size of the memory buffer allocated for storing packets.
- The priority of the queue.
- The drop profiles associated with the queue.

4. Q-in-Q implementation in Virtual Machines Servers

In a final deployment scenario, regardless the switches implement Q-in-Q, the user computers will be located in virtual machines, placed in several VM servers, which are able to tag just one level of VLANs. How to map this one level VLAN, into a Q-in-Q scenario, is an issue that must be solved.

In order to implement the layer 2 control tool (including Q-in-Q implementation) in this real scenario, different possibilities have been studied. This chapter tries to explain a possible solution to this problem. This solution is just a suggestion that tries to open a discussion about how to solve this issue.

First of all, a short introduction to the virtual machines servers used in FEDERICA Project VMWare ESXi Server 3 networking properties will be done.

4.1 Virtual Machines Server networking

FEDERICA will use virtual machines (VM) as endpoint. The virtual machines are able to load different operative system and can be created and deleted on real-time. Those virtual machines are allocated in Virtual Machines Servers (VMS). These servers are VMware ESX Server 3i, and are managed with VMware Infrastructure Client.

Each network card of a VMS can connect several Virtual machines through a Virtual switch.

A virtual switch (vSwitch) is an abstracted network device, that behaves as a physical Ethernet switch. It is able to detect the virtual machines that are logically connected to each of its virtual ports. It is also able to forward traffic between virtual machines.

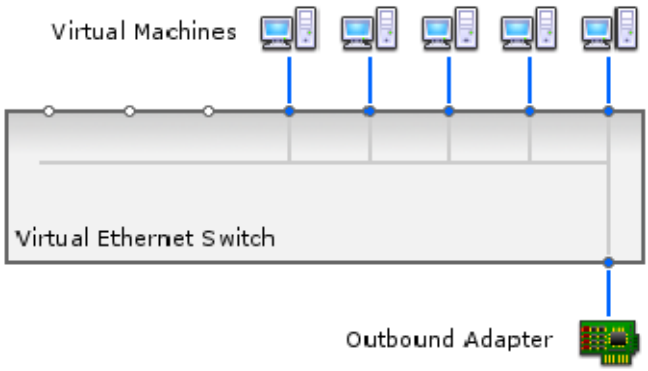

Figure 4-1. Virtual machines server

Virtual networks and physical networks can be interconnected by connecting a vSwitches and physical switches through Ethernet adapters. Connecting these different types of switch works in a similar way as physical switches are

connected between them, although vSwitches does not offer all the advanced functionalities that a physical switches use to do.

Port groups are another kind of abstraction working with Virtual Machines Servers. These port groups connect network services to the virtual switches, and define how a connection is made through the vSwitch. Port groups aggregate multiple ports under a common configuration, helping to connect virtual machines to labeled networks. A port group specifies port configuration options, such as bandwidth limitations and VLAN tagging policies for each member port.

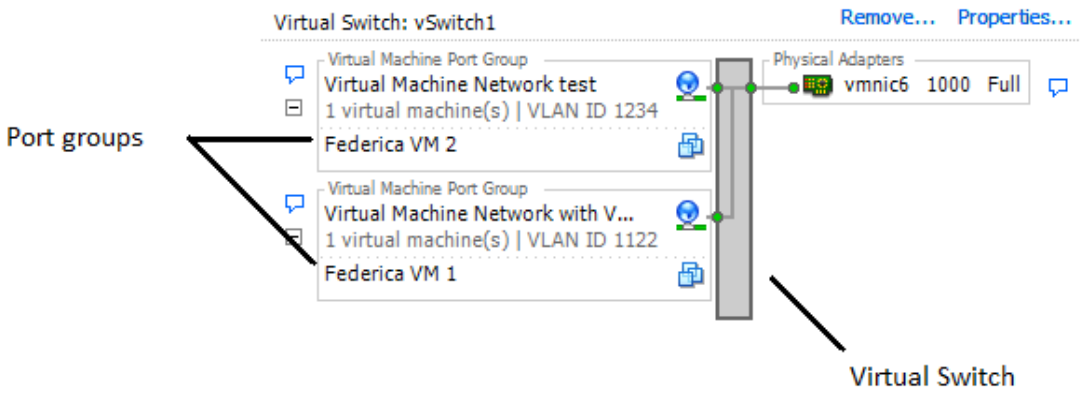

Figure 4-2. Virtual machines server networking structure

4.2 One level VLAN implementation

We have been looking for the way to implement Q-in-Q tagging via software (directly in Linux), with the purpose to be the VM who double-tags the frame.

For the moment, any solution has been found in this direction, and only one level tag can be implemented via software. This is the second possibility to do it, for the moment. The first one is to rely on the ESX Server vSwicht to tag the frames. Both possibilities are explained, and finally, a way to integrate one of them into a Q-in-Q scenario is purposed.

4.2.1 First option

If this procedure is implemented, the vSwitch inside the VMServer will be the one who will tag the S-VLAN from the VM. As a VLAN per port group can be configured, we are able to configure many S-VLANs tagging just one VM frames per S-VLAN.

This second test is presents two virtual machines, each one in a VMS. The virtual switches inside the VMS are configured to tag the packets as VLAN 1234. The interfaces ge-0/0/14 and ge-0/0/15 are members of the testVlan, with vlan-id 1234. Both interfaces must be configured as trunk interfaces.

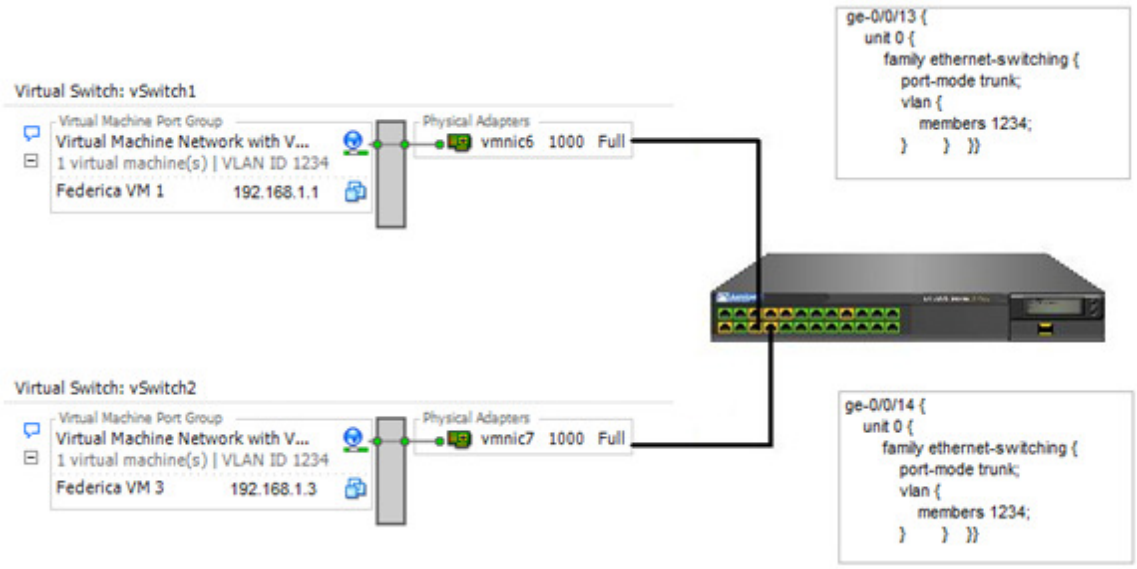

Figure 4-3. VLAN implementation in the Virtual machines server (1)

The behavior of the system is equal to a system with two physical switches connected to a third one, in a tree configuration. We have been able to ping from one VM to the other, even in separated physical cards.

4.2.2 Second option

We are going to set the vSwitches up not to tag any VLAN by themselves. The user VLANs will be tagged by the VMs operative system, instead of being tagged by the vSwitches. This will be performed by an Ubuntu Linux distribution able to manage VLANs. The physical switch does not receive any kind of traffic in this scenario.

The goal of this test is to get in touch with the VLAN tagging via software in a PC or VM.

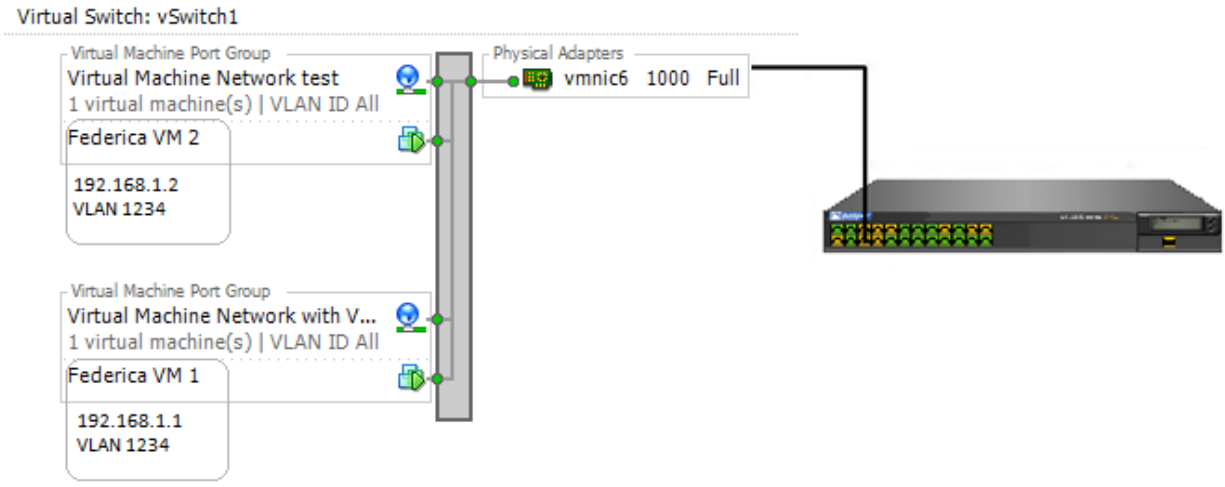

Figure 4-4.VLAN implementation in the Virtual machines server (2)

The VM machines are configured with an Ubuntu Linux 8.10 distribution. This distribution must have installed the vlan package.

Configuring 802.1q must be done as described:

```
modprobe 8021q
sh -c 'grep -q 8021q /etc/modules || echo 8021q >> /etc/modules'
nano /etc/network/interfaces

#Add the vlan interface
      auto eth0.1234
      iface eth0.1234 inet static
      address 192.168.99.1
      netmask 255.255.255.0
       vlan_raw_device eth0
```

If the VLAN tags are different, there will be no communication between the two VMs. If the vlan tags are the same, we are able to ping from one machine to other.

Unfortunately, implementing this kind of configuration, we have only been able to ping from one VM to other pending in the same vSwitch. If we try to do it from a VM machine connected to one port of the physical switch, to other VM connected to a different port, we have not been able to communicate them, even trying different configurations of the physical switch (interfaces set as trunk, or access, etc.).

4.3 Q-in-Q implementation proposal

As a possible solution to the limitation of the VM Server (which does not offer a Q-in-Q implementation), has been studied.

Keeping in mind the following scenario:

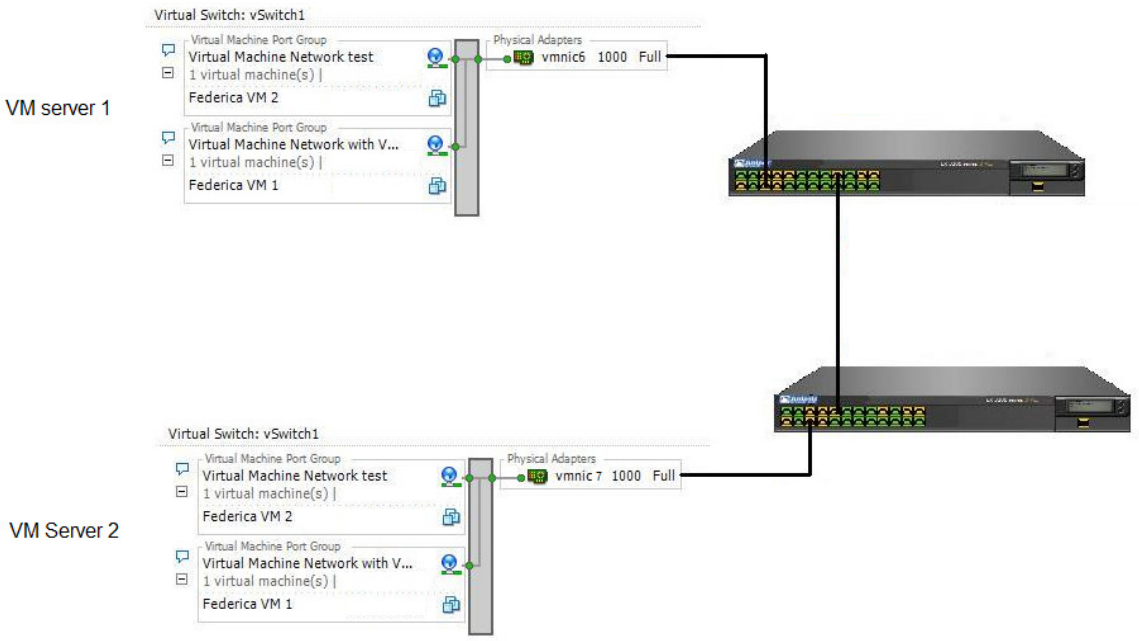

Figure 4-5.Scenario for the Q-in-Q implementation.

We want to configure a slice, consisting of two VMs, one of them placed in the VM Server 1, and the other one placed in the VM Server 2, and connected through the switches.

The first step would be the NOC to create the slice, by configuring the service VLAN, or slice VLAN, for the user 1 (S-VLAN 1). With this purpose, the NOC would have to follow these steps:

1. Create a port group in the VM Server 1, and set it up to tag with VLAN ID 1, and finally assign the VM1 to this port group. In practice, this would be an equivalent to have a VM tagging frames with the S-VLAN 1.
2. Do the equivalent in the VM Server 2, setting up also the S-VLAN 1.
3. Create a VLAN in the switches, with the VLAN ID set to 1.
4. The physical ports where the VM servers are connected must be members of the VLAN 1.
5. Set up the physical interfaces connecting both switches, as trunk, and also add them to the VLAN 1.

Right now we have the slice of the user 1 configured. From now, we are going to set up the user VLAN (C-VLAN). To do it, the user would have to configure:

6. In the physical switches, we must create a VLAN with the VLAN ID 11 (for instance). This will be the C-VLAN 11.
7. Add C-VLAN 11 to the trunk ports.
8. The next step would be to configure the switch to perform Q-in-Q, adding an additional tag (C-VLAN 11) to the incoming S-VLAN 1 packets, coming from the port where VM Server 1 is connected.

Steps from 6 to 8 are where the user VLAN is configured. It could be a guided process, done through a tool, where the user would have to specify the VLAN ID.

If we repeat the same steps to configure a C-VLAN for a different user (different slice) we would have the configuration shown in the figure 6, where the green frames are the slice-tagged frames, and the yellow headers are the C-VLAN headers.

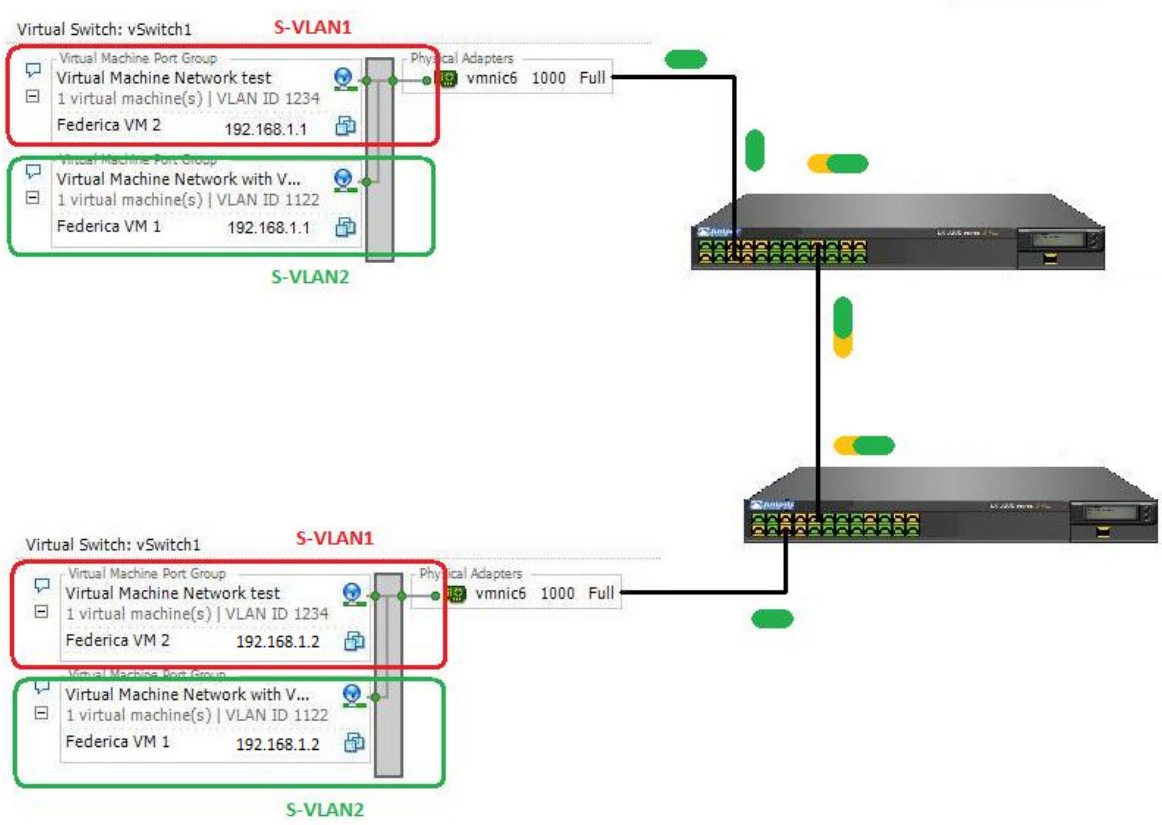

Figure 4-6. Q-in-Q implementation expected result

This strategy presents some weak points that must be taken into account if the deployment is done as we have explained:

- To map correctly S-VLAN <-----> C-VLAN, each VM Server (actually only each physical card) can only give service to a unique C-VLAN. If the configuration follows this restriction, the map S-VLAN – C-VLAN is univocal.
- We have not tested the behavior of the switches with the double-tagged frames.
- We can only have some suppositions about the scalability of this solution in a scenario with several VLANs.
- The user application must be able to configure the switches completely (access and port interfaces, Q-in-Q) transparently to the user (the user will ask for interfaces to be members of his C-VLAN, but the application will have to automatically configure Q-in-Q transparently).

5. Tool overview

5.1 Introduction

The tool is expected to control the layer 2 resources along a network, providing to the NOC admin an automated way of resource management and provisioning. The objective of this tool is to cover some the requirements related to Layer 2 presented in the chapter 2.

The tool will control remotely different devices in a network, being able to manage some properties of these devices from a centralized graphical user interface. The main properties to be controlled are the referred to layer 2 configuration parameters. So we will be able to configure interface properties as well as to create and manage VLANs of these several devices from a single interface. This tool has been designed to be used in the FEDERICA project, a scenario where a NOC will virtualize physical resources and will lease them to different users. This NOC will have to be able to manage and control these resources. The tool has been designed to help in this task. In this chapter the core of the application has been explained. The Web Services module and the Graphical Interface description can be found in the annexes of this thesis.

In a future, more parameters will be able to be managed from this tool, such as CoS parameters, Q-in-Q configuration. Moreover, the tool will be able to manage not only the current Juniper EX-3200 Ethernet Switches, but Juniper MX-480 Router as well.

5.2 Technologies used for the development and deployment

5.2.1 Java

Java is a programming language originally developed by Sun Microsystems and derives much of its syntax from C and C++. It has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture.

5.2.2 Globus Toolkit

The Globus Toolkit, currently at version 4, is an open source toolkit for building computing grids developed and provided by the Globus Alliance. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own

machine room while simultaneously preserving local control over who can use resources and when.

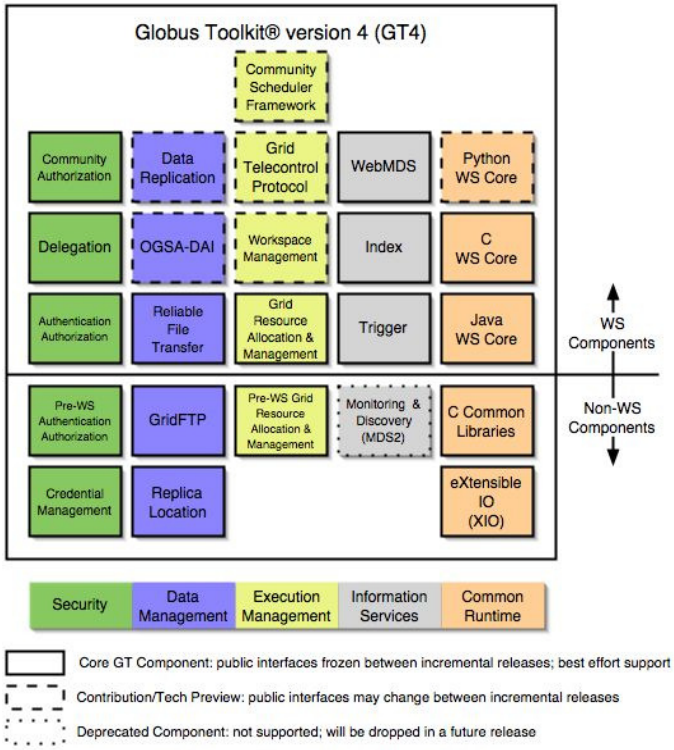

Figure 5-1. Globus toolkit modules

5.2.3 PostgreSQL

PostgreSQL is an open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX, and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored. It includes most SQL92 and SQL99 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others, and exceptional documentation.

5.2.4 IaaS Framework

The IaaS Framework is a set of resources, libraries and tools licensed under the Apache Software License version 2 that enable developers to quickly create new virtualization solutions based on the Framework programming model. The functionalities provided by these tools allow a developer to choose which web service stack will be used to expose the physical infrastructure as a service (supported SOAP engines include Axis2, CXF and Spring-WS), and provide a series of modules to plug-in capabilities like security, reservation management and data persistence to the infrastructure service. The Framework also provides libraries to speed up the development of drivers to communicate with the

physical devices, like protocol parsers (TL1, NetConf), transport handlers (TCP, SSL, SSH) and a driver architecture called the IaaS Engine.

The architecture of IaaS framework it is described in the next picture:

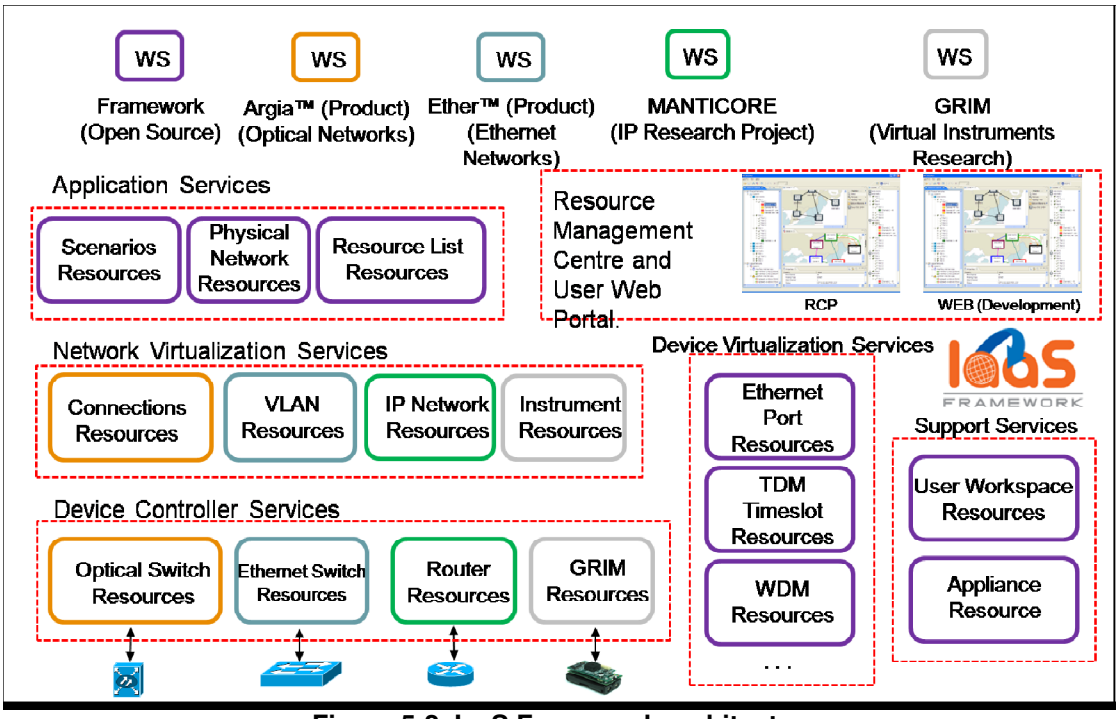

Figure 5-2. IaaS Framework architecture

5.2.5 Eclipse Rich Client Platform

Eclipse Rich Client Platform (RCP), a set of libraries that offers the developer the possibility of managing components to be used to form a final rich client application (in a similar aspect to Eclipse).

The Eclipse Rich Client Platform consists of the following components:

- Eclipse Runtime: It provides the foundational support for plug-ins, as well as the extension points and extensions.
- The Standard Widget Toolkit (SWT): It provides efficient and portable access to the user-interface facilities of the operating systems on which it is implemented.
- JFace: JFace is a User Interface (UI) framework, layered on top of the SWT, and it manages many common UI programming tasks.
- Workbench: The workbench builds on top of the Runtime, SWT and JFace to provide a scalable, open-ended, multi-window environment. It is able to manage perspectives, views, editors, wizards, etc.

5.3 Architecture

We are going to do a first approach to the tool, with the objective to understand globally how it is structured and the way it works.

The tool presents a graphical user interface (GUI) where the NOC admin can manage and visualize all the resources in the different network devices, as well as the resources assigned to each user. From the user interface the NOC can also add devices to the network. The GUI communicates with the Ethernet Resource Service, via web service, which stores in a database the resources assigned to each user. Ethernet Resource Web Service manages the resources assigned to the different users of the network. For given offered resource is created an Ethernet Resource that will be assigned to the final user that has asked for it.

The GUI also communicates the actions required by the user to the different devices module, also via web services. These device modules, called Engines (one running per each switch in the network) transform the actions required into switch commands. These commands are communicated via Netconf to the switch that is configured. The switch also sends its new configuration back to the device module, which also sends it to the GUI (and to the resource service).

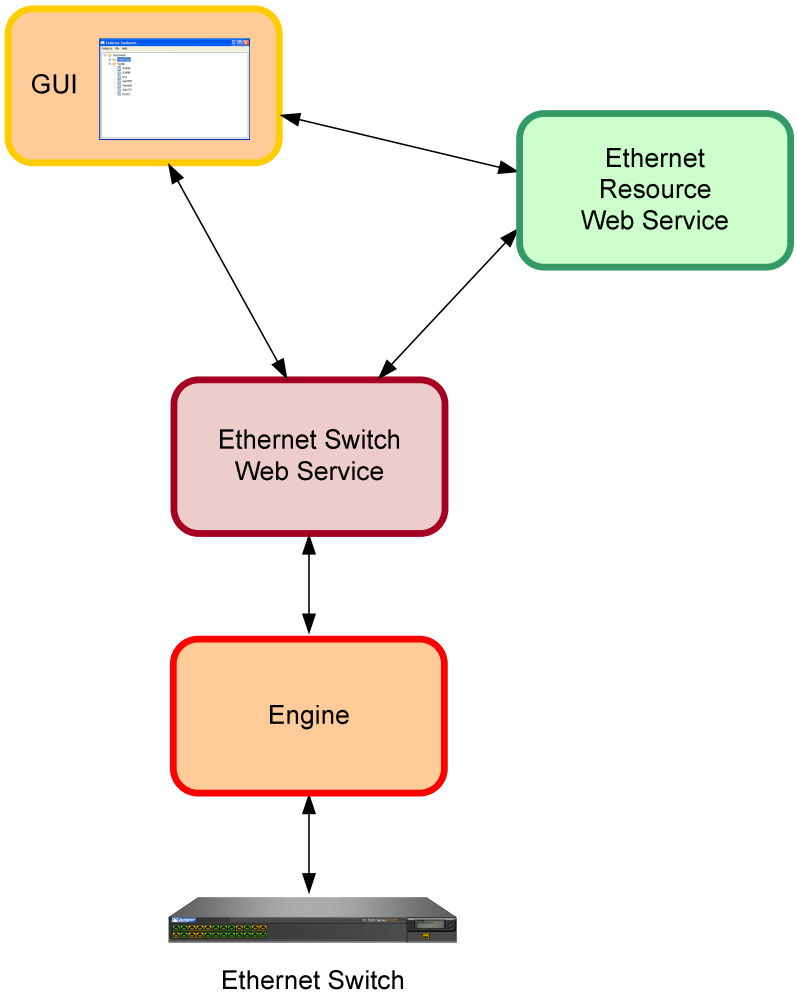

Figure 5-3. Application modules

5.4 Engine Core

5.4.1 Introduction

The Engine is a module defined by the IaaS Framework. Its behavior is equivalent to a driver of the specific physical device it is modeling. This driver is synchronized with the physical device it is controlling, and it is accessed via Web Services. At the same time, the driver also defines completely the way it access to the physical switch, implementing the transport and the protocol to be used in the communication between them.

The Engine describes the device, trying to model it in the most generic way, making it transparent to the specific model and manufacturer of the devices, in order to make it portable from one manufacturer to another, for devices with similar characteristics.

The Engine also defines as Transport protocols such as SSH, TCP, Telnet. Concerning the communication protocols (communication with the device), the Engine defines the protocols NETCONF, CLI and TL1 managing the request and the response of the consequent actions and commands in each one of the specified communication protocols.

5.4.2 Architecture

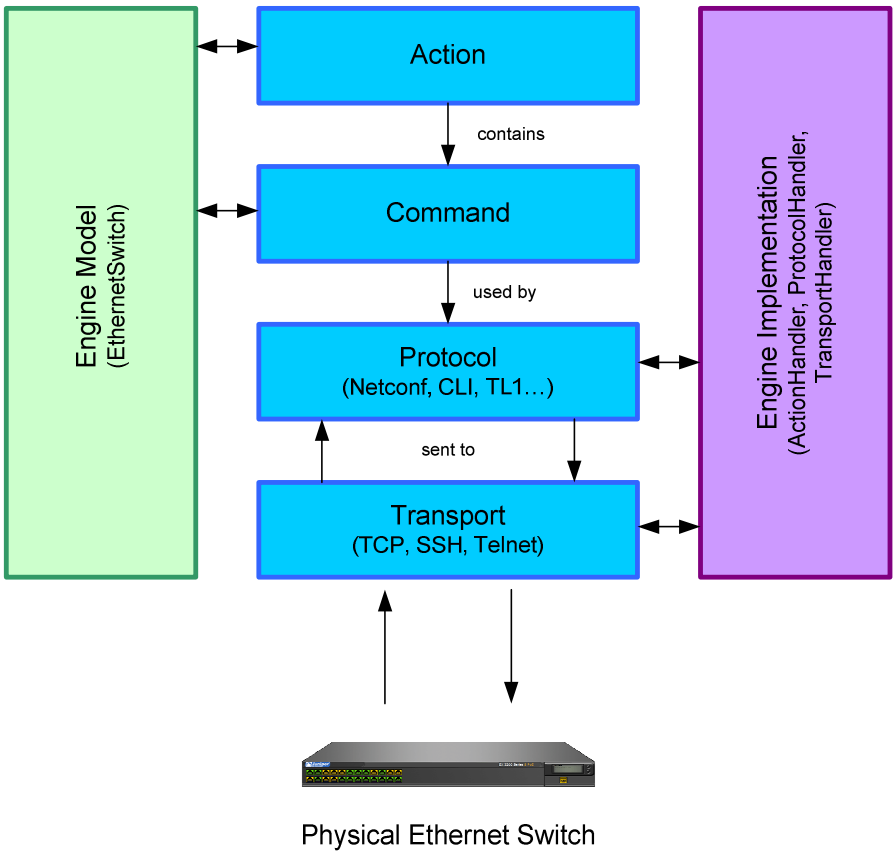

Figure 5-4. Engine architecture

The modular architecture is shown in the previous figure.

5.4.3 Action

The action represents what is wanted to change in the switch, in the higher level, the most similar to the human will. Actions are composed by one or more simple commands, and it does not depend on the protocols being used.

For instance, an action would be “Create VLAN”. This *CreateVLANAction*, would be composed for one or several nuclear steps (commands) to achieve this purpose, such as enter configuration command, edit vlans command, etc.

Changes in the commands are only reflected in this module if, for instance, from one protocol to other, more commands are needed to be sent to do the same action.

5.4.4 Command

Command is a nuclear change to be sent to the physical device. These commands are specific to the communication protocol used to manage the switch, and command groups can be formed depending on the protocol or the device used. For example, Netconf will use a set of commands, and CLI will use a different set. These commands will be part of an action.

5.4.5 Protocol

Protocol is the responsible to communicate to the switch in its language.

Protocol would be the set of commands that the physical device is able to understand to be configured. In other words, each protocol will have a different set of commands. A physical device can understand one or more protocols.

Protocols sometimes depend on the underlying transport. For instance, in the case of the Netconf protocol (as in our project) requires an underlying SSH transport, while CLI can work over TCP and SSH.

This module was already included in the IaaS framework so it has not been developed for us.

5.4.6 Transport

Transport module manages the underlying communication between our program and the physical device. It establishes the corresponding connections to the switch in the transport protocol it understands. It is also the responsible to send to the switch all the messages sent by the upper protocol layer. It must be as abstract as possible in order to work with as most protocols as possible. For

instance SSH transport can be used in our project to send Netconf and CLI as well.

This module was already included in the IaaS framework so it has not been developed for us.

5.4.7 EthernetSwitchModel (implemented)

The Ethernet Switch Model has been implemented in order to model the switches that will be used in the project.

This model is composed by the following data:

- SwitchID: It contains the unique ID of the switch.
- Manufacturer: The device manufacturer is contained. For this project, only Juniper devices have been used.
- Model: It contains the switch model. In this project, only EX-3200 devices are used.
- Protocol: It contains the protocol used to communicate with the switch. NetConf is used for our project.
- Transport: It contains the transport protocol where the switch communication protocol travels. SSH is used in our project.
- Equipment: The resources inventory of the switch is contained. This inventory contains the switch ports and the configured VLANs.
- Security: It contains the pair user-password to be sent to the switch.

5.4.8 Engine Core (used)

The Engine Core is situated over the EthernetSwitchModel, and it is used to call and to initialize the model, transport and protocol taking the device-specific information. It is also the responsible to receive the actions to manage the switch. The Engine Core is composed by:

- actionHandler: actionHandler manages the different actions.
- protocolHandle: It is the responsible to communicate with the device, using an specific protocol handler for each protocol.
- transportHandler: It is the responsible to manage the transport to communicate with device, using an specific handler for each transport protocol.
- engineModel: It has the model instance.

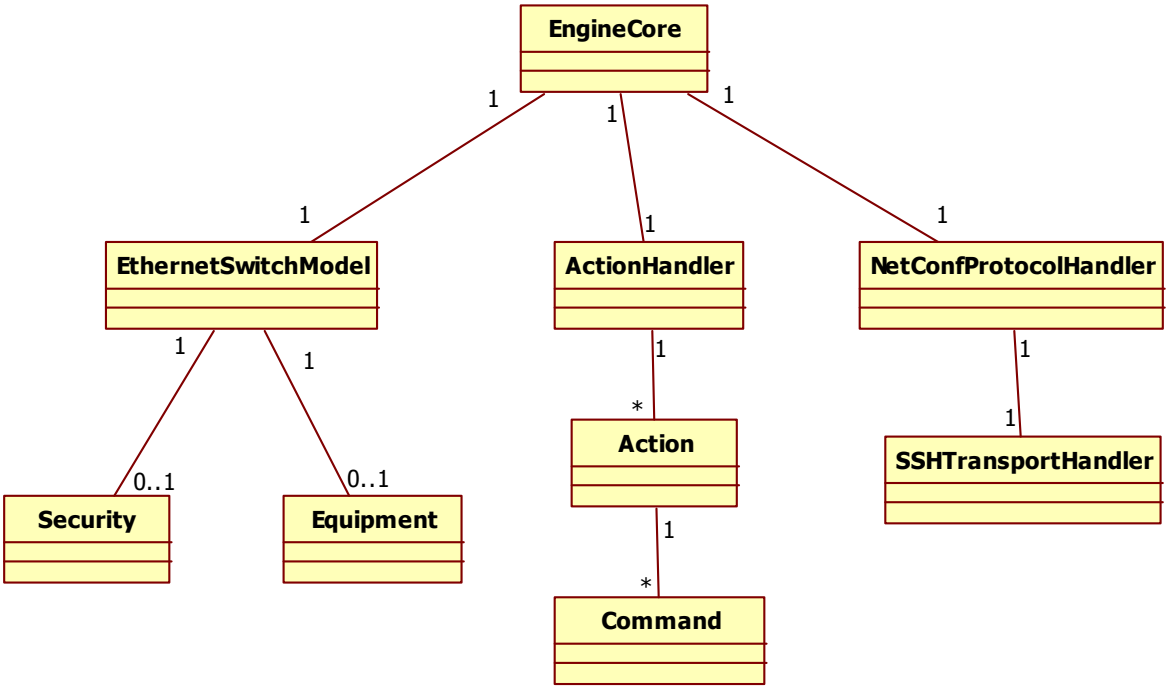

Figure 5-5. Class diagram

5.4.9 NetConfProtocolHandler (used)

NetConfProtocolHandler manages the communication with the device done through NetConf protocol. It contains the actions and functions to send and receive messages. This is done writing the resources into NetConf language, and in the opposite direction, parsing the NetConf messages into resources able to be managed by the program and then stored in the Equipment data structure.

Some of the implemented functions are:

- open: It stablishes the NetConf session with the switch, by initializing the corresponding transport handler.
- sendReceive: This action is used to send a request to the switch, expecting a response message.
- send: No response is expected when this action is used to send a request to the switch.
- close: This action closes the session.
- init: Defines the transport Netconf will be sent over.

5.4.10 SSHTransportHandler (used)

This class is the responsible to establish the SSH transport connection, where the protocol that will communicate with the switch will be sent over. The process is to initiate, send, receive and finally end the transmission with the device. This is done for each action that is wanted to be done.

5.4.11 ActionHandler (implemented)

This class manages the actions to be sent to the device from the engine. All the actions content inside a Command that is sent to the switch (exactly sent to the NetConfProtocolHandler). The actions currently implemented in the Engine are the following:

- CreateLogicalInterfaceAction: This action creates logical interfaces from a physical interface of the switch.
- DeleteLogicalInterfaceAction: This action is used to delete existing logical interfaces from a physical interface.
- UpdateLogicalInterfaceAction: This action updates a logical interface (or more than one), with a new configuration.
- CreateVlanAction: This action creates one or several VLANs in the switch.
- DeleteVlanAction: This action eliminates logical interfaces in a physical interface.
- UpdateVlanAction: This action updates the properties of a VLAN.
- QueryResourcesAction: This action requests to the switch the VLANs and interfaces current configuration. This is done in order to keep up to date the engine model in the program and in the GUI.
- ConfigurePhysicalPortAction: This action set the physical port properties up.

5.4.12 Security (used)

Security parameters are defined in this class. Password setting, certificate management can be configured. Nevertheless, only the user and password necessary to be sent to the switch have been configured for our project.

5.4.13 Equipment (implemented)

Equipment class represents the switch configuration, either the current configuration as the candidate configuration as well (the configuration that is wanted to be sent by the actions). Both configurations contain the resources inventory of the switch.

This inventory contains the switch ports and the configured VLANs:

- configuredPhysicalPorts: The currently configured ports are listed.
- requestedPhysicalPorts: It contains the configuration of the ports that is wanted to be sent to the switch.
- configuredVlans: The currently configured VLANs are listed.
- requestedVlans: It contains the configuration of the VLANs that is wanted to be sent to the switch.

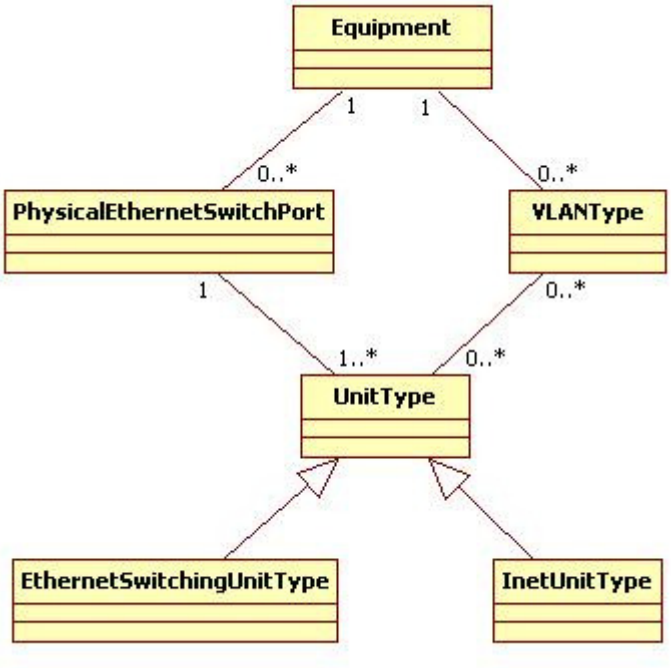

Figure 5-6. Classes relationship

5.4.14 PhysicalEthernetSwitchPort (implemented)

The model of the physical interfaces of the switch is contained in this class. The model includes the following properties:

- name: Physical port name in the switch.
- description: It contains the port description. This information can describe the utility of the port, the kind of traffic that it will contain, etc.
- linkSpeed: It contains the port rate or capacity. In the model we are working with (Juniper EX-3200) these rates are 10 Mbps, 100 Mbps, 1 Gbps.
- linkMode: The transmission mode of the physical port is defined. The possibilities are half-duplex, full-duplex or automatic.
- configuredUnits: The virtual interfaces which the physical port is divided in are listed. This is a list of UnitType.

5.4.15 UnitType (implemented)

Units are a logical division of a physical interface of the switch. If units are configured, they can act as a typical interface. In order to adapt the generic unit type, to the two specific unit types implemented by the switch, an abstract Unit Type has been created, from which EthernetSwitchingUnitType and InetUnitType inherit from the abstract UnitType class.

The common parameters are:

- family: This field determines if the UnitType will be EthernetSwitchingUnitType (family ethernet-switching) or InetUnitType (family inet).

- description: This field contains the description of the logical interface. It can contain useful information about the type of traffic that the unit will manage, where will it be connected, and so on.
- state: It determines the current state of the interface between enabled or disabled.
- interfaceName: It contains the name of the physical interface where the unit is defined.
- unitName: It contains the unit name, a number in the range from 0 to 16384. The unit fullname is composed by the combination of the physical and the logical name: physicalInterfaceName.unitName.

5.4.16 EthernetSwitchingUnitType (implemented)

This class inherits from abstract class UnitType and it is used for the units with family defined as ethernet-switching. These units are defined to work strictly in Ethernet.

The specific attributes that describe the EhtenerSwitchingType are:

- trunkPortMode: Determines if the port is configured as trunk or not.
- nativeVlanId: If the port is configured as trunk, determines if there is a default VLAN configured and the id of such VLAN. The unlabeled packets received by this interface will be tagged with the vlan-id specified.
- configuredVlans: This field contains a list of the VLANs the port is member of (one if the port is not trunk, more than one if it is so).

5.4.17 InetUnitType (implemented)

This class also inherits from abstract class UnitType, but it also has an IP address to work also in layer 3. It is not used in the application. The specific attribute is:

- IPAdress: This field contains the IP address that is assigned to the port.

5.4.18 VLANType (implemented)

This class is created to model the VLANs created in the switch. This model is characterized by the following fields:

- name: This field contains the VLAN name. This name must be unique.
- tag: This field contains the VLAN ID, which will be the tag of the VLAN. It also must be unique.
- description: This field contains the description of the VLAN. It can be used to explain some information related to the use of the VLAN, etc.
- state: This field contains the current state of the VLAN (it can be enabled or disabled).
- units: The interfaces belonging to this VLAN are listed in this field.

5.4.19 ObjectMapper (implemented)

The ObjectMapper class is the responsible to map the response sent by the switch after a request done by the Engine. The functions parse the response of the switch to each action sent by the switch. Usually the response of the switch is a full description of the resources in XML format. ObjectMapper maps this information into the classes explained before.

5.4.20 Other implemented modules (implemented)

There are other relevant modules, such as:

- JuniperEX3200: This class has been developed and contains the information for the specific physical device, such as action list, protocols, transport, model, etc.

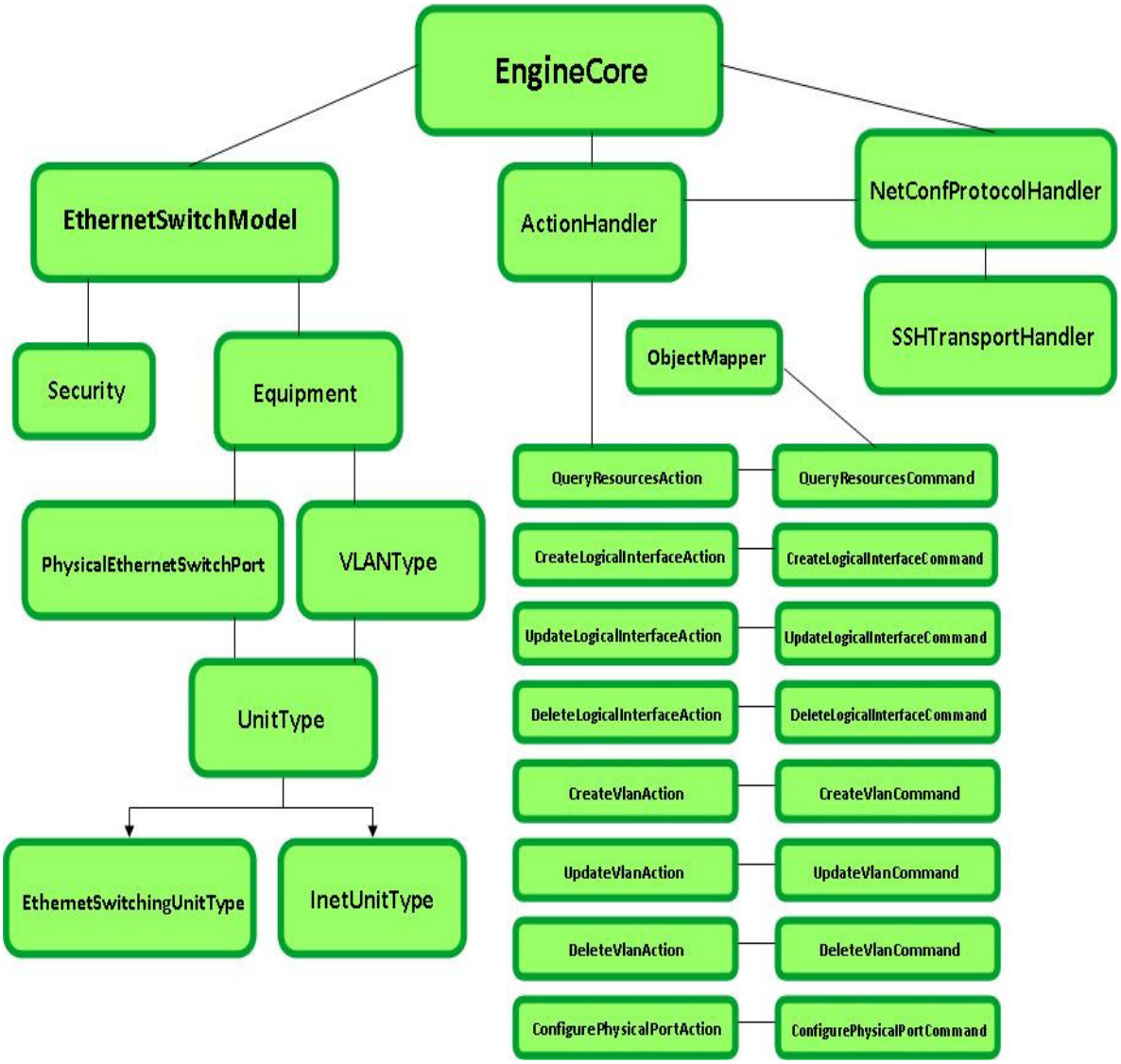

Figure 5-7. Engine class composition

5.5 Use cases

For the moment, the user will have a set of operation to do with the current version of the application. Although some operations have been explained in a very extensive way, in this section all the operations will be explained in a more generic way, in order to have a global inventory of the available operations at a glance.

Use cases available in the GUI

5.5.1 Create a switch

The switch creation is the process to be done when the administrator wants to add a physical switch to the application, introducing the identification and connection information, and obtaining the resources in the switch. The process to follow is:

1. Once the application has been initialized, the administrator right-clicks in the free space and selects the “Create Switch” option from the context menu. The corresponding wizard is launched, and the user is asked to introduce the switch information in different fields, such as device name, manufacturer, model, or configuration parameters as IP address, user and password, and finally transport and protocol.
2. The graphical user interface contacts with the corresponding web service and sends the configuration data. The web service creates a model with the sent configuration and asks for the Engine to load the real configuration of the physical switch.
3. The Engine requests the switch its configuration, and the switch sends a response message with its current configuration. This configuration is stored in the model.
4. The graphical user interface is updated with the resources obtained from the model. The GUI shows the switch and its corresponding interfaces and VLANs.

5.5.2 Refresh Switch

Refresh switch is selected when the administrator wants to update the switch configuration in the GUI (it is not real time updated). The process is:

1. The administrator right-clicks the switch for which he wants to update the configuration.
2. The GUI contacts with the web service sending the request. The web service makes the engine the request for a resources update.
3. The engine asks the switch for its current configuration. The switch sends a response with the updated configuration.
4. The configuration is updated in the GUI.

5.5.3 Create a VLAN

To create a VLAN on a specific switch, the NOC has to follow this event sequence:

1. NOC admin selects the switch that is going to be configured and selects “Create VLAN” option from the context menu.
2. NOC admin configures the new VLAN parameters through the wizard.
3. The GUI contacts with the web service sending the request for a VLAN creation. The engine introduces a candidate configuration in the model, and sends to the switch this candidate configuration.
4. The switch sends a response for this request when the changes have been applied. The model is updated and with it the GUI.

5.5.4 Delete VLAN

The process to delete a VLAN is analog to the VLAN creation, but the request sent is to delete a certain VLAN. The sequence is:

1. NOC admin selects the switch that is going to be configured and selects “Delete VLAN” option from the context menu.
2. NOC admin selects in the wizard the VLAN he wants to delete.
3. The GUI contacts with the web service sending the request for a VLAN deletion. The engine introduces a candidate configuration in the model, and sends to the switch this candidate configuration parsed in Netconf.
4. The switch sends a response for this request when the changes have been applied. The model is updated.
5. The switch is updated in the GUI, where the VLAN has been removed from the model.

5.5.5 Virtualize

A resource from a concrete configuration (user and assigned VLAN) can be created to be assigned to a network user. This is done following as it is explained:

1. The administrator selects an existing logical interface from the switch where it is contained, and selects “virtualize” from the context menu.
2. The interface is selected in the wizard, and the user and VLAN are introduced by the admin.
3. The GUI contacts with the corresponding web service sending the configuration data. The web service creates a new resource with the port configuration.
4. The graphical interface is updated showing the changes in the switch.

5.5.6 Assign Port to VLAN

This action adds a logical interface to a VLAN. This is done as it follows:

1. The administrator selects the switch where the VLAN is created, and selects “Assign port to VLAN” from the context menu.

2. A wizard helps the administrator to select the interface and the VLAN.
3. The GUI sends to the corresponding web service a message with the purpose of adding the selected interface to the selected VLAN.
4. The web service obtains the model of the corresponding switch, and makes the engine to do the operation.
5. The engine sends the configuration to the switch, adding the interface to the VLAN interfaces list.
6. The switch sends a response back. The model and the user interface are updated.

5.5.7 Update Logical Interface

The administrator may want to update the interfaces in the switch. To do this, he must follow these steps:

1. The NOC selects the interface in the switch where he wants to update the information and from the context menu selects “update interface” option.
2. A wizard will guide the NOC to help him changing the interface configuration.
3. The GUI contacts with the corresponding web service sending the desired configuration. The web service sends the operation to the engine of model corresponding to the switch that contains the specified interface.
4. The engine sends a request to the switch modifying the selected interface. The switch sends back a response.
5. The changes are reflected in the graphical user interface.

Use cases not available in the GUI

The following use cases are not yet available in the graphical interface, but the application is able to perform them.

5.5.8 Update VLAN

The administrator may want to update also a VLAN in a switch. To do this, he must follow these steps:

1. The NOC selects the VLAN in the switch where he wants to update the information and from the context menu selects “update interface” option.
2. A wizard will guide the NOC to help him changing the VLAN configuration. Parameters like VLAN name, VLAN ID and description can be modified.
3. The GUI contacts with the corresponding web service sending the desired configuration. The web service sends the operation to the engine of model corresponding to the switch that contains the specified VLAN.
4. The engine sends a request to the switch modifying the selected VLAN. The switch sends back a response.
5. The changes are reflected in the graphical user interface.

5.5.9 Unvirtualize

A previously created resource can be deleted of the configuration. This is done following as it is explained:

1. The NOC selects the resource that wants to delete.
2. The corresponding web service is contacted sending the resource identifier.
3. The resource is deleted from the database.

5.5.10 Configure Physical Port

The administrator can modify the properties of the physical interface of an existing switch. These properties can be the link speed, the link mode, the description of the interface, etc. This is done as follows:

1. The administrator selects the physical port he wants to configure from the corresponding switch and selects the field to be modified.
2. The corresponding web service selects the model of the corresponding switch and initiates the operation in the corresponding engine.
3. The engine introduces the new configuration in the switch sending a request. The switch sends a response back and the model is updated by the engine.

5.5.11 Create Logical Interface

The administrator can create logical units in the physical interfaces of an existing switch. This is done as follows:

1. The administrator selects the physical port where he wants to configure a new logical interface (or unit) and enters its configuration properties.
2. The corresponding web service selects the model of the corresponding switch and initiates the operation in the engine in it.
3. The engine introduces the new configuration in the switch sending a request, creating the new logical interface. The switch sends a response back.
4. The engine updates its model.

5.5.12 Delete Logical interface

The administrator can delete the already existing logical interfaces of an existing switch. This is done as follows:

1. The administrator selects the physical port and the logical interface (or unit) he wants to remove.
2. The corresponding web service selects the model of the corresponding switch and initiates the delete operation in the engine in it.
3. The engine introduces the new configuration in the switch sending a request, deleting the logical interface. The switch sends a response back.
4. The engine updates its model.

6. Conclusions

The requirements for a control plane management tool focused in the FEDERICA project concern some topics as virtualization, operational, end user requirements and monitoring, useful to establish an operative control plane able to manage the resources across a Layer 2 network.

A study of the existing resource management tools has been done as previous work for this project, taking into account the previous requirements. We can conclude that, although all these tools have some functions that fit well with the requirements of the projects, there is no existing tool that fulfills all the requirements needed for FEDERICA.

Based in one of the studied frameworks, a Layer 2 management tool has been developed in order to offer the administrator of the network a way to create and manage virtualized resources across different devices in the network. This management tool is able to manage VLANs and interfaces of the different interfaces, grouping and showing the information by device or by user. The tool is centralized and remotely usable, and these are two important issues and have affected from the root of the application, because it has been designed to offer these two properties, making the application modular, and implementing web services for the modules communication. The application has been developed to easily add more drivers for different devices, from other vendors or other models, as the possibility to add in a very easy way new functions and operations to the existing ones.

Although Q-in-Q and CoS has been studied in this project, even the implementation steps are defined and some tests have been done in the case of Q-in-Q, these topics are not yet implemented in the tool. The implementation of these two important properties will be essential as further steps.

The graphical user interface is limited to the basic operations offered by the engine, and with the certainty of a future integration into a centralized tool bench common for the FEDERICA project. Nevertheless, the current GUI offers the capability of centralizing the management of the different devices across the network.

The development of the application from the communication with the devices to the monitoring to the user, and the storage in a database, require an implicit resource description. Although no explicit chapter has been included concerning this topic, in the device chapter and in the tool chapter information about the description of the resources has been included.

From the point of view of the FEDERICA project, the results of this project are partial, referring that not all the functions required for the project are yet covered, and the GUI is not the final one. There is more work to be done in the year that the project will last. Nevertheless, the under the optics of the master thesis, the objectives have been accomplished.

As further steps to improve the tool, the implementation of Q-in-Q and CoS are essential steps to complete the tool. A substantial improvement in the GUI can be done, in order to make easier the representation of the resources in the network. This improvement has to be focused to the horizon of a full integration to a complete tool bench for control all the resources (in layer 2 and layer 3) in the FEDERICA network, trying to achieve an homogeneous control plane for the entire network.

Also a Network concept should be included in the program, in order to offer end-to-end consistency to the user. For the moment, the individual device configuration and resource management is available. It would be interesting to offer the user the possibility to form a network with these resources.

Also security issues must be studied and implemented to make fully operational the tool in the FEDERICA project.

7. Acronyms

CLI	Command Line Interface
CoS	Class of Service
FEDERICA	Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures
GbE	Gigabit Ethernet
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPv4, IPv6	IP version x
LAN	Local Area Network
MPLS	Multiprotocol Label Switch
NOC	Network Operation Center
QoS	Quality of Service
SOAP	Simple Object Access Protocol
SSH	Secure Shell
VLAN	Virtual LAN
VM	Virtual Machine
VMS	Virtual Machines Server
WS	Web Service
WSDL	Web Service Description Language
XML	Extended Markup Language

8. Bibliography

8.1 FEDERICA

[1] FEDERICA Project: http://www.fp7-federica.eu

[2] “DJRA1.1: Evaluation of current network control and management plane for multi-domain network infrastructure”, C. Cervelló, M. Carabias, R. Machado.

8.2 Juniper Devices

[3] "JUNOS Software NETCONF API Guide. Release 9.3.", Juniper Networks, Inc.

[4] "JUNOS Software - Complete Software Guide for JUNOS Software for EX-series Switches. Release 9.4", Juniper Networks, Inc.

[5] Juniper Website: http://www.juniper.net

8.3 Programming

[6] “SWT /JFace in action”, M. Scarpino et al., Manning Ed.

[7] “Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications”, By J. McAffer, J. M. Lemieux

[8] “The Globus Toolkit 4 Programmer’s Tutorial”, Borja Sotomayor, University of Chicago, Department of Computer Science

[9] “W3 Schools Online Web XML Turorials”: http://www.w3schools.com/xml/

ANNEXES

TITLE: Design and Implementation of a control plane for a L2 virtualized network

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Roberto Machado Calvo

DIRECTOR: Cristina Cervelló i Pastor

DATE: July 14 th 2009

1. Available tools and frameworks

In the previous chapter we have highlighted the requirements of the virtualized network where we want to provide the management tool.

With that in mind, the scope of this chapter is to introduce and analyze a set of available tools and frameworks that are related to the needs that a virtualized resources network presents. These tools are currently deployed and tested, and they could be useful to fix the scope of the tool we are working in.

1.1 ANStool

ANStool (Advanced Network Services) was developed by GRNet. It is a simple and extensible framework for networking engineers to get information about the network. This information can be stored in a database. The tool can also provide optional QoS inside a Layer 2 (or Layer 3) VPN.

ANS also provides a web based application for establishing bandwidth reservations using various technologies within a simple domain.
The ANStool works mainly in the OAM plane. It does not communicate directly with the control plane of the devices. The tool is able to manage some Layer-3/2 devices (currently Cisco and SNMP managed devices) and it is, at the moment, under modification to model Layer 1 (WDM) devices.
It has a centralized topology mapping functionality with distributed control over the devices.

The provisioning is done in a semi automatic way. The configuration is produced automatically but the administrators need to feed it to the corresponding networking devices, until sufficient trust and security is guaranteed.

1.2 ARGON

ARGON (Allocation and Reservations in Grid-enabled Optical Networks) is a centralized management and allocation system of on-demand network services, and in advanced with QoS within a single administrative domain. It is specially focused to be used on Grip applications and Grid scheduling.

ARGON can operate on layer 2 via GMPLS / VLANs / VPLS and layer 3 via MPLS, and also offers different service types for inter-cluster communication and data-transfer. It provides interfaces for co-allocation of heterogeneous resources in the Grid.

There are two core services to be used:

- A pipe service that provides end-to-end connections with user specified QoS within a frame.
- A malleable service that provides connection with a determined QoS to support data transfer tasks.

Resources are manually assigned to ARGON, configured via SNMP and CLI, and are enriched with availability information over time (bandwidth through time awareness, for instance). ARGON also finds and established shortest paths given some user-specified constraints, as could be bandwidth constraints, time constraints (start time, duration), and delay constraints.
ARGON supports multidomain architectures and reservations. A reservation request can be comprised of several services (session concept), and therefore, it can be applied, as a whole, as an admission mechanism of a more complex communication structure.

1.3 AutoBAHN

AutoBAHN (Automated Bandwidth Allocation across Heterogeneous Networks) is a bandwidth reservation and signaling interface system that allows to make advance reservations with automated provisioning between interconnected domains (peer-to-peer) that may be deployed over heterogeneous network technologies (L1 and L2).

AutoBAHN works in a distributed way, where each domain requires a domain manager, who deploys domain-specific policies and implementations. The domain manager is formed by two different modules: one is responsible for inter-domain operations of circuit reservation on behalf of a domain (the Inter-domain manager), and the Domain manager, a module that is responsible for instantiation of BoD instances within a single domain. It contacts the Technology Proxy module either, to request the configuration of the BoD service instance. This Technology Proxy module allows AutoBAHN the support of several technologies and vendors according to multiple domain and global requirements.

As AutoBAHN is a modular system, it allows the support of external services, as the Authorization and Authentication Infrastructure (AAI) for instance. The system is based on a set of specific interfaces and supports unique implementations in individual domains.

The specific technological details are described in a non-ambiguous, simple and common language form, an abstract network resource representation. This describes all the AutoBAHN's system components and the negotiations between domains.

The AutoBAHN system offers a dedicated graphical user interface (GUI) to manage it in a centralized way. The GUI is a web portal which gathers information about the accessibility of the domain managers that have been registered to the GUI via Web services. This GUI allows the request and

cancellation of reservation services, monitoring states of submitted reservations and representation of a map with the registered domains.

1.4 BLUEnet Tool

This configuration tool is based in ANStool, and it has been modified by HEAnet to support MPLS layer 2 circuits. This tool is able to automate the provisioning of Ethernet point-to-point links over the HEAnet network (intra-domain provisioning).

BLUEnet Tool performs semi-automatic discovery and automatically maps new devices added to the network. It also provides a GUI where the users can also select the interfaces (two end points) and configure the link. The creation and deletion of the links are done through Web services. Monitoring tools as Nagios and Cricket are automatically created and deleted for each new circuit.

This tool is mainly focused to be used in a MPLS network it to configure Port mode/VLAN mode links over native Ethernet and layer 2 MPLS VLL clouds.

- Port mode service is a point-to-point port-based transparent Ethernet virtual circuit that provides connectivity for both layer 2 data and control plane data. It is used to connect geographically remote LANs over the HEAnet network. Customer 802.1q ports are configured to perform Q-in-Q.
- VLAN mode service is a point-to-point VLAN-based Ethernet virtual circuit. This is a non-transparent service, where VLANs are mapped to a MPLS label switched path. The user sees a 802-1q trunk which filters customer Layer 2 control protocols.

1.5 DRAC

DRAC (Dynamic Resource Application Controller) allows applications to configure the network without requiring interfacing directly to a wide range of diverse network protocols, features and devices. This application works at layer 1 and layer 2 (optical and packet switching architecture) but it also provides connectivity at Layer 3.

The main scope of DRAC is to provide connectivity to grid applications, trying to connect different hosts used by these applications that require some QoS specifications. These grid applications will access the hosts with a specified bandwidth, delay, jitter and packet loss. The foundation that DRAC uses for the provisioning of these QoS parameters is “cut-through”; a switching method for packet switching systems, wherein the switch starts forwarding a frame (or packet) before the whole frame has been received, normally as soon as the destination address is processed. This technique reduces latency through the switch, but decreases reliability. DRAC uses this method and steers packets in Layer 1 instead of Layer 3. Doing that, it can achieve better QoS requirements, and as a result, a better network performance.

The order of the operations to be done by the tool is the following: First of all, applications request DRAC a path between two end points of the network. QoS parameters are also specified in these requests. Once the request is received the tool tries to find an available path across the optical and packet switching architecture. When a path is found the QoS requirements are validated, and if they are achieved the path is virtualized, and it becomes a virtual switch where the two end-points of the path are the two interfaces of the switch. A transparent connectivity is provided to the application. After the virtual switch is generated, DRAC validates the QoS requirements in the extremes of the virtual switch, for finally sub-leasing the instantiated path to the application.

1.6 DRAGON

DRAGON (Dynamic Resource Allocation via GMPLS Optical Network) is a network architecture and defines a Control Plane able to manage provisioning across multi-domain, multi-layer and multi-service. The architecture is based on switching and forwarding nodes supporting GMPLS. The architecture defines multiple autonomous networks domains, where internal traffic management policies are defined internally by each one of these domains. Also the peering arrangements with other domains are managed by each one of them.

The main objective of DRAGON is to instantiate inter-domain end-to-end LSPs and dynamically provision them across different domains and heterogeneous network technologies. Nevertheless, there are some missing capabilities and technologies in order to make available this capability to end systems such as a simple interface, an end-to-end proper instantiation (with AAA), the ability to signal across non-GMPLS enable network elements, etc.).

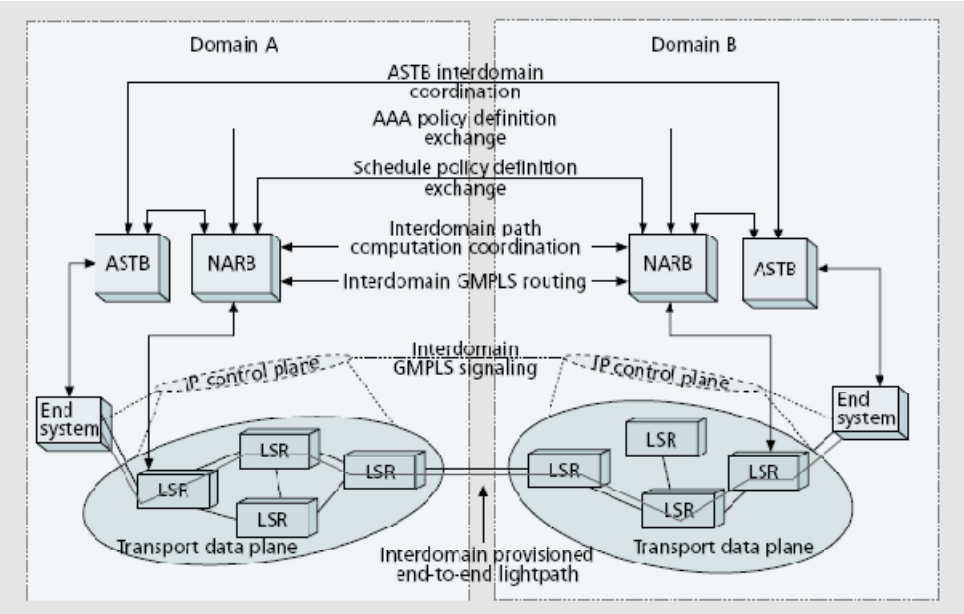

Figure 1-1. DRAGON architecture.

DRAGON is structured in four basic modules:

- Network-Aware Resource Broker (NARB): The NARB module represents the local Autonomous System (AS) or domain and is used as a path computation engine and inter-domain routing. NARBs peers across domains and exchange topology information based on the actual topology as discovered by listening to the local OSPF-TE protocol, or optionally based on an “abstracted” view of the domain topology. It also includes advanced algorithms which allow path computation with multiple constraints such as GMPLS -TE parameters, AAA policies, scheduling, etc.
- Application-Specific Topology Builder (ASTB): Application-specific Topologies (ASTs) are requested by an end user and are generally a set of LSPs in which an application domain wants to be set up as a group. The ASTB accepts requests from users or end systems for multiple network connections, and uses the NARB services to determine if the requested network paths are available with appropriate AAA and schedule the applied constraints. The NARB views these requests as individual LSPs and the ASTB is responsible for the assembly of multiple LSPs into a specific topology.
- Virtual Label Switch Router (VLSR): A non-GMPLS capable network device (Ethernet, TDM and Optical switches) is converted to a VLSR by the addition of a small UNIX-based V-Node which runs a GMPLS Control Plane consisting of OSPF-TE and RSVP-TE. The VLSR V-Node acts as a GMPLS proxy agent for a device and translates protocol events into commands that the local switching element understands, such as SNMP, TL1, or even scripted CLI commands. This allows non-GMPLS devices to be included in end-to-end path instantiations.
- End-System Agent (ESA): This software agent runs on the end-system that terminates the Data Plane link of the provisioned service. It also allows the initiation of a provisioning action on behalf of the end system.

1.7 MANTICORE/Argia & IaaS

The IaaS (Infrastructure as a Service) Framework is an open-source framework descendant from the UCLP Research Program and licensed under the Apache Software License version 2. This framework is a group of libraries, tools and applications, as well as documentation and best practices to follow in the developing process of physical to virtual (P2V) solutions.

This framework goal is to enable a uniform development frame, providing interoperability across different middleware implementations. It also avoids over-provisioning of IT infrastructure, and is able to create virtual resources to partition networks into multiple subnetworks. It seeks the use of virtual resources within a single domain or across multiple domains. The framework also provides common resources to be used by different technologies.

The functionalities provided by the tools and libraries that compose the framework allow developers to choose which web service stack will be used to expose the physical infrastructure as a service (supported SOAP engines include Axis2, CXF and Spring-WS). A series of modules are also provided to plug-in capabilities like security, reservation management and data persistence to the service. The Framework also provides libraries to speed up the development of drivers to communicate with the physical devices, like protocol parsers (TL1, NetConf), transport handlers (TCP, SSL, SSH) and a driver architecture called the IaaS Engine.

The framework manages different domains devices and allows their interconnection. Each device (router, switches, etc.) is managed by a Web-Service and is represented using WSRF as a Resource. This framework offers common functionalities that can be used by several types of networks and devices. The offered services can be activated through a GUI or with an application able to call the web service API.

Some examples of tools or services that use this framework are MANTICORE and Argia. The former provides virtualization mechanisms for IP networks and the latter provides virtualization mechanisms for optical networks.

The architecture of IaaS framework it is described in the next picture:

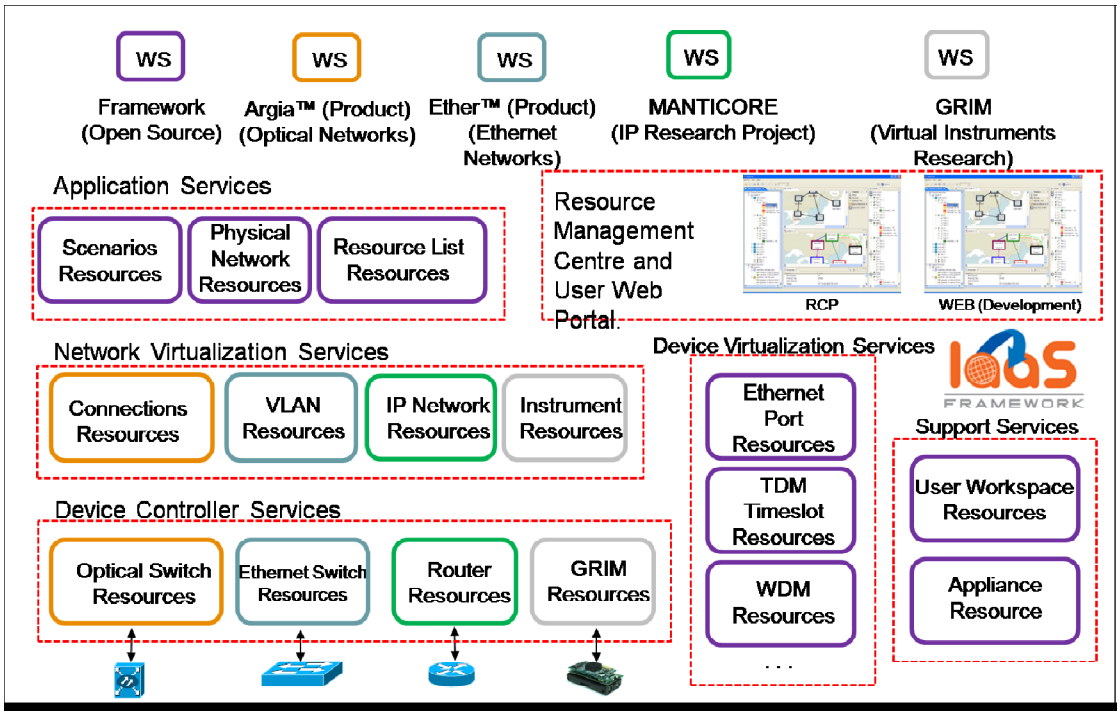

Figure 1-2. IaaS framework architecture.

1.8 Juniper SRC

The Juniper Networks Session and Resource Control (Juniper SRC) Portfolio is a carrier grade policy and control solution that integrates third party platforms

and applications that enable the end-to-end delivery of differentiated services across multi-vendor network infrastructures. It consists of hardware and software modules providing reliable and scalable IP network layer solutions to the policy and control layer. Key policy and control layer functions include policy management, subscriber management and authentication, authorization and accounting (AAA), and network resource control.

SRC supports the Policy and Control Management Plane, and it is connected to the Service Plane and the Transport Plane. The focus of SRC is to control the device of the operator domain, translating a service request into a policy in the Transport/Control Planes. For example, SRC supposes all the interdomain coordination is performed at the Service Plane Layer.

Juniper SRC presents functionalities in admission control, bandwidth management and adjustment and accounting.

1.9 PL-VINI

PL-VINI is an evolution of Planet Lab. PL-VINI uses currently available routing and forwarding software (Click modular routers, XORP router control plane) to build on-demand network topologies as overlay on top of existing infrastructure. The tool has a centralized control, although the provisioning is done automatically across the different nodes, which are able to manage different slices. It works in a single domain although the resources are distributed over the Internet.

PL-VINI manages resources partitions using Vserver virtualization software and VNET socket to isolate the different slices.

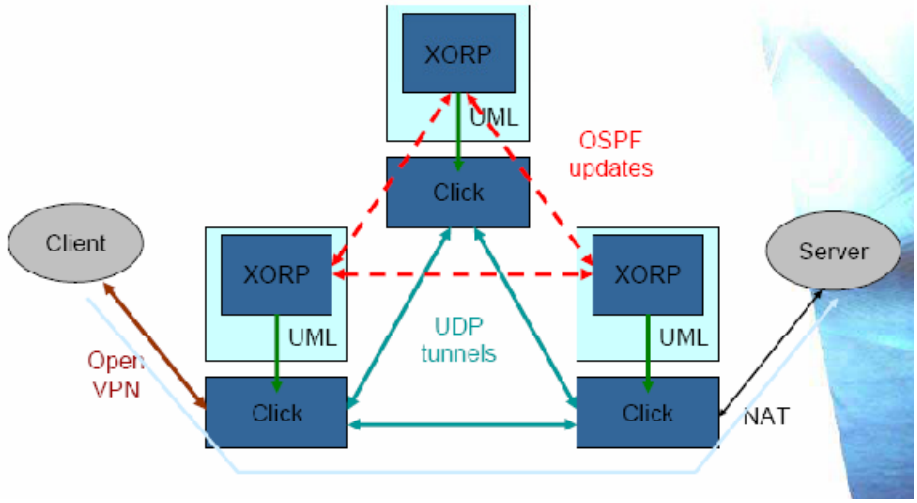

Figure 1-3. PL-VINI application

2. Other tool modules

Other important modules form the structure of the tool. These other modules are web services and the graphical user interface. In this annex we have explained these modules and how do they interact with the core of the application.

2.1 Web services

The remote communication between the different modules of the application is implemented via Web Services.

2.1.1 Web Services introduction

A web service is defined as a software system designed to support machine-to-machine interaction over a network. Web Services usually can be accessed over a network, and executed on a remote system hosting the requested services. This technology is used by distributed applications, and also allows creating client-server applications. The most important aspect is that the web service is oriented to a machine-to-machine communication, as has been commented before.

In one hand, the main advantages of the Web Services over the other technologies focused in distributed applications (such as CORBA, RMI, etc.) are the following:

- Web Services use XML standards. This fact makes Web services independent of the programming language and of the platform as well. In other word, client and server can be running in different operative systems and be developed in different languages.
- HTTP is used as communication protocol. This fact means that web services are more robust than other technologies for the same purpose. As they are based in HTTP, web services can go through firewalls, for example.

In the other hand, the disadvantages are:

- Lack of versatility: The services can be invoked only in few ways.
- Loss of effectiveness or overhead. Transmitting all the information in XML is not as efficient as transmitting in an own binary code if the amount of data to be transmitted between the nodes is large enough.

Our web services cover the communication between the GUI and the different services, and it has been developed framed with the Globus Toolkit, explained in the section 5.2.2 . Among its functionalities, this project uses the capability to create services (Java Web Service Core) with persistency given by Hibernate of some information.

2.1.2 Web Services architecture

The typical architecture of any Web Service is the following:

- Service Processes: This part of the architecture usually involves more than one Web Service. For instance, this part of the architecture contains the discovery functions. As our project has the WS directions (URI) defined, there is no need to use these discovery functions.
- Service Description: Web Services are self-describing. This means that, after being discovered, the web service response to "description-requests" and describes itself, indicating which operations supports and the way to invoke it. This is handled by the Web Services Description Language (WSDL).
- Service Invocation: Invoking a web service involves passing messages between the client and the server. How the request to the server must be formatted, and how the server should format the responses is specified by SOAP (Simple Object Access Protocol). There are more invocation languages (such as XML-RPC, etc), but SOAP is the most used.
- Transport: All the messages are transmitted between server and client through HTTP (HyperText Transfer Protocol). This protocol is the same used to access conventional web pages on the Internet. Thanks to this fact, Web services are able to pass through the firewalls.

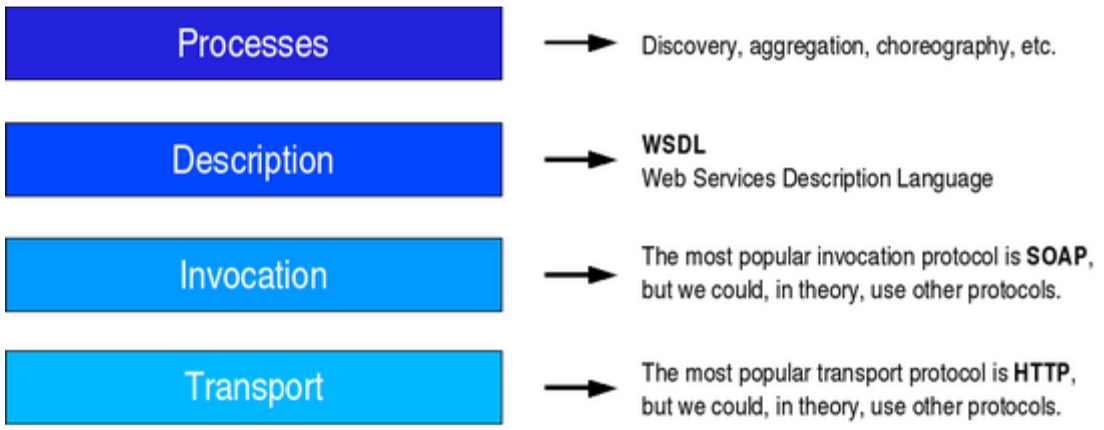

Figure 2-1. Web Services architecture

2.1.3 How do Web Services work?

A typical web service invocation process is explained here. We suppose that the service is already located. This is the case of our project, where the Web service URI is defined.

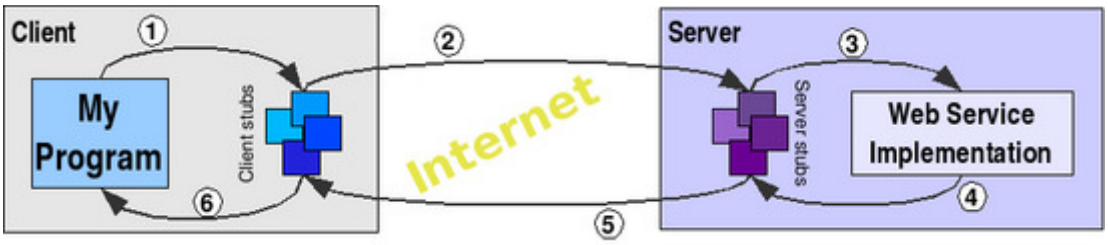

Figure 2-2. A typical Web Service invocation

1. Whenever the client application needs to invoke the Web Service, it will really call the client stub. The client stub will turn this 'local invocation' into a proper SOAP request (this is called serialization).

2. The SOAP request is sent over a network using the HTTP protocol. The server receives the SOAP requests and hands it to the server stub. The server stub will convert the SOAP request into something the service implementation can understand (this is called deserialization).

3. Once the SOAP request has been deserialized, the server stub invokes the service implementation, which then carries out the work it has been asked to do.

4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.

5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.

6. Finally the application receives the result of the Web Service invocation and uses it.

2.1.4 How does the server work?

The server hosting the web service is structured as follows:

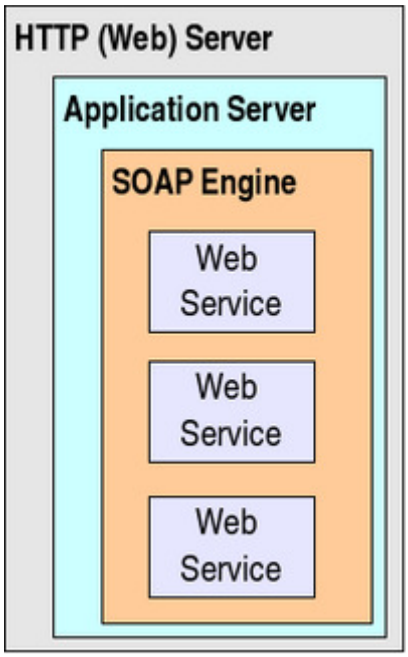

Figure 2-3. The server side in a Web Services application

- Web Service: First of all it is the web service, the piece of the software that exposes a set of operations. As we want a set of clients to be able to invoke those operations, and the web service does not know anything about how to interpret SOAP requests and to create SOAP responses, a SOAP engine is needed.
- SOAP Engine: This part of the software is the responsible of handle SOAP requests and responses. A common SOAP engine is Apache Axis, and it is the SOAP engine used by Globus Toolkit, the tool used in our project. The SOAP engine is hosted by an application server.
- Application service: The application server provides the frame for applications that must be accessed by different clients. The SOAP engine runs as an application inside the application server.
- HTTP Server: The web server is the software that knows how to manage HTTP messaged. One of the most popular web servers in the Internet is Apache.

2.1.5 The Web Services Resource Framework

Web services have certain limitations by themselves. In fact, plain web services would not be able to build complex applications because they are not able to keep statefulness. In other words, Web Services are stateless by themselves. They cannot “remember” information, or keep state from one invocation to other.

The solution is to keep the web service and the state information separated, this way, the web service keeps being stateless, and we are able to have a state for

our requirements. So, instead of putting the state information in the service, we will keep this information in a separate entity called resource. This resource will store all the state information. Each resource will have a unique key or identifier, so in the moment to cal l the web service, the resource has to be specified.

These resources can be stored in the memory or in a data base. The resource specification is done through WS-Addressing, which provides a more versatile way of addressing Web Services, comparing it to the plain URIs. So, a pair Web Service – Resource is called WS Resource, and the address of a particular WS-Resource is called an endpoint reference (EPR).

The Web Services Resource Framework (WSRF) is a collection of different specifications related to the management of the WS-Resource. The most useful specifications for the present project are:

- WS-ResourceProperties: A resource can be composed by zero or more resource properties. WS-ResourceProperties specifies how resource properties are defined and accessed. The resource properties are defined in the web service's WSDL interface description.
- WS-BaseFaults: This specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.
- WS-Addressing: This specification provides a versatile mechanism to address the Web Services rather than plain URIs. WS-Addressing can be used to address a Web Service + resource pair (a WS-Resource).

There is a terminology explanation that must be mentioned. During this Web Services section, Resource will refer to the web service module responsible to keep the state of the different variables. Out of this section, the term resource will keep its meaning of virtualized capabilities of the switch (VLANs and interfaces).

2.1.6 Web services implementation

To communicate with the Engine, the following web services have been implemented:

- EthernetSwitchService: There will be an instance of this web service, but one resource for each switch managed by the application is contained by this service. It manages the following operations:
    - Invoke: Actions over the engine must be invoked by this method.
    - Refresh: This operation makes the engine to update the information from the switch.
    - Virtualize: This operations creates a resource to be used by an end user of the network (EthernetResource).
- EthernetSwitchFactoryService: This non-persistent instance manages the created EthenetSwitchServices. It offers these operations:
    - Create: this operation creates a WS-Resource. This is an EthernetSwitchService and a EthernetSwitchResource pair.

- Clone: This operation creates a new EthernetSwitchService from an existing one.
- Find: Given a concrete template, this operation gives back the End Point References of the EthernetSwitches matching with this given template.

Following the methodology explained by Globus to deploy a stateful Web services, there are five steps to follow.

These steps are:
1. Defining the interface in WSDL
2. Implementing the services in Java
3. Configuring the deployment in WSDD and JNDI
4. Create a GAR file with Ant
5. Deploy the service into a Web Services container

1. Defining the interface in WSDL

The first step is to define the service interfaces; this is to specify the operations that will be available to the users (independently of how the services are internally implemented). The service interface is usually called port type.

This interface specification is done by a special XML language, the Web Services Description Language (WSDL), and each service has its own description, stored in the following files:

- EthernetSwitch.wsdl: Describes in XML language the services offered (also known a portType), the corresponding resource and the request/response messages declaration.
- EthernetSwitchFactory.wsdl: Doing the same functions as EthernetSwitch, describes the EthernetSwitchFactory in this case.
- EthernetSwitch.xsd: This file describes the structure and the restrictions of the operation messages and the different resources of the wsdl files, in a more extended and exact way, beyond the syntactical rules that the XML language allows.
- namespace2package.mappings: The way to connect the WSDL specifications to the specific language implementations, a connection point must be created. This is done by the stubs, a Java files generated from the WSDL files by the Globus Toolkit (GT4). Namespaces defines where (in which Java packet) to place them, mapping the WSDL namespaces into Java packets.

  We can see here the mapping. The first line maps the WSDL files into packages. The other namespaces are generated automatically when Globus Toolkit performs their descriptions:

```
http\://core.upc.cat/2008/12/ethernetswitch=cat.upc.core.stubs.ethernetswitch
http\://core.upc.cat/2008/12/ethernetswitch/bindings=cat.upc.core.stubs.ethernetswitch.bindings
http\://core.upc.cat/2008/12/ethernetswitch/service=cat.upc.core.stubs.ethernetswitch.service
```

2. Implementing the services in Java

Now is the moment to implement what has been explained in the previous step. In other words, how does the service do what it says it does? With this purpose, the classes that implement the services previously explained are introduced here:

- EthernetSwitchQNames: The parameters included in the Web Services are represented by qualified names (QNames). Their structure is the namespace followed by the proper referred name. As the reference to QNames is really frequent, these QNames have been grouped in an own class. In Java they have the following form:

```
public static final String NS = http://core.upc.cat/2008/12/ethernetswitch;

public static final QName RP_NAME = new QName(NS, "Name");

public static final QName RP_MODEL = new QName(NS, "Model");

public static final QName RP_MANUFACTURER = new QName(NS, "Manufacturer");

[…]
```

- EthernetSwitchService: This service presents the implementation of the remote operations that can be done over the service (invoke, refresh and virtualize). All parameters and return values must be well formed in order to be correctly interpreted by the stubs (automatically generated). Also the operations related to the corresponding resources are included in this class.
- EthernetSwitchResource: This class contains the following resource properties:
    - Name: The name of the switch is stored (with persistency in the database).
    - Model: The model of the switch is stored (with persistency in the database).
    - Manufacturer: The manufacturer of the switch is stored (with persistency in the database).
    - Configuration: All the information required to communicate with the switch is stored with persistency in the database. This information is IP address, port, user, password, transport and protocol.

In order to work correctly, is required that these attributes are named as in the resources properties defined in the WSDL file. This class also keeps an instance of the Engine, where all the operations and configuration of the switch is stored (but not persistently, not in database, but loaded each time the Web Service is called). It also implements the initialization, storage and load in the database.

- EthernetSwitchResourceHome: It is used by EthernetSwitchFactory WS to create resources for a gien EthernetSwitch WS.
- EthernetSwitchStore: It stores the active engines during the web services execution. It does this storage in a hash table, where EthernetSwitchResource access to it to manage its own engine.
- EthernetSwitchFactoryService: It implements the three operations described in its WSDL file.

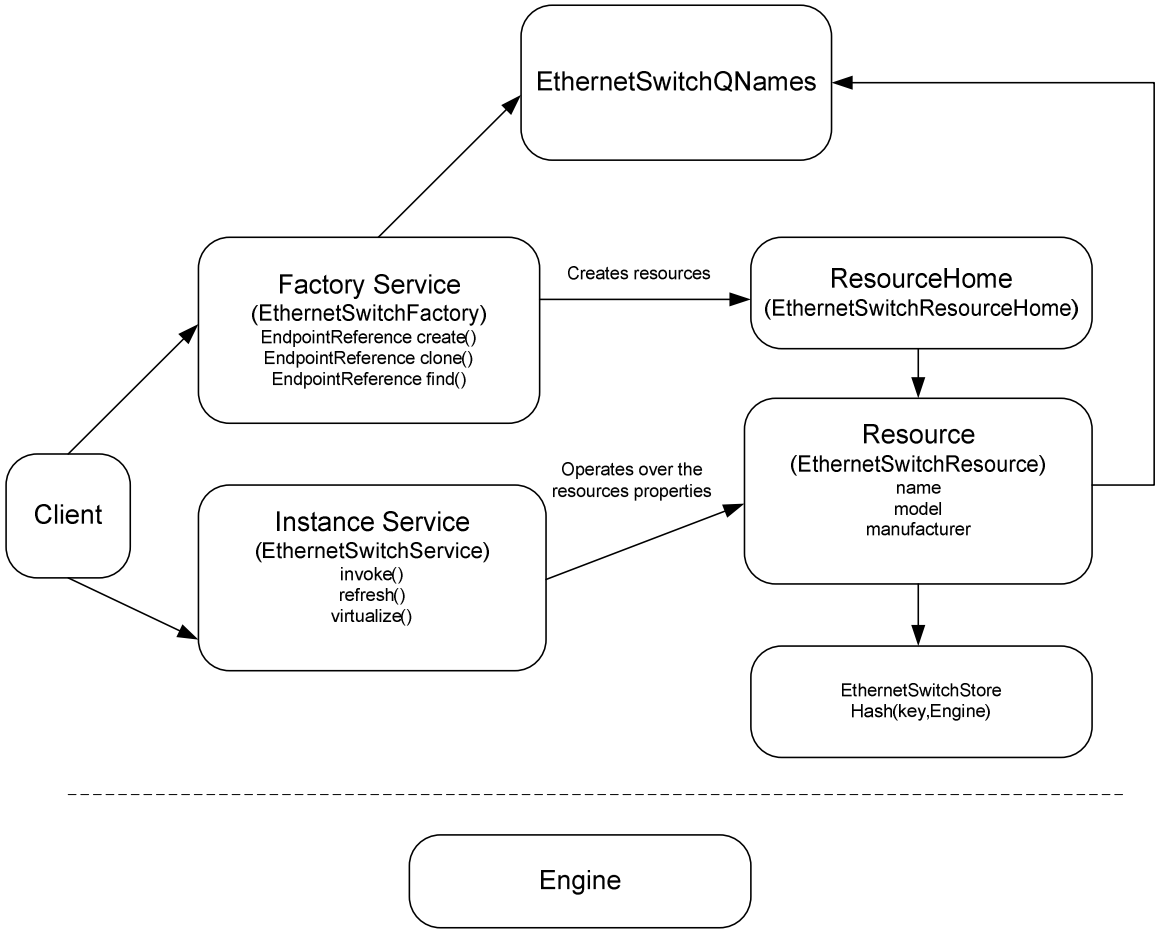

Figure 2-4. Ethernet Switch Web Service class interaction

Other interesting classes that are part of the Ethernet Switch Web Service:

- EthernetSwitchConstants: Constants used during the execution are included here.
- ObjectMapper: This class maps the switch configuration into the message structure that stubs are able to manage.
- EthernetResourceWrapper: It is the responsible of the communication between EthernetSwitchService and EthernetResource. It is used in the virtualize operation.

3. Configuring the deployment in WSDD and JNDI

Once implemented the different web services, they have to be deployed in order to make it available to the users.

For this purpose there are two important files:

- deploy-server.wsdd: WS Deployment Descriptor (XML). It describes how the web services must be published. It defines the services directions (URIs), points the Java classes that they implement, etc.

```
[…]
<service name="core/EthernetSwitchService" provider="Handler" use="literal" style="document">
[…]
<service name="core/EthernetSwitchFacoryService" provider="Handler" use="literal" style="document">
[…]
```

These lines specify the web services location. Taken also the server base direction, the URI can be obtained for each one of them. For instance, if the server base direction is:

  - http://147.83.118.238:8080/wsrf/services

We would obtain the URIs:

  - http://147.83.118.238:8080/wsrf/services/core/EthernetSwitchService
  - http://147.83.118.238:8080/wsrf/services/core/EthernetSwitchFactoryService

- deploy-jndi-config.xml: It defines which resource home must be used for each service, and some other parameters to obtain the correct deployment of the services.

4. Create a GAR file with Ant

Up to this moment we have obtained the service WSDL definitions, the Java implementation, the deployment description (WSDD and JNDI) explaining how the definitions and the implementation is presented to the outer world. We have these different parts, which must construct a whole service to be offered. How these files must be placed in the server, and some other issues are solved by Globus generating a Grid file (GAR) containing all the web services information needed for them to be deployed in the server.

This GAR creation process is complex and it requires the following steps:

- WSDL files process.
- Stubs creation from WSDL.
- Stub class compilation.
- Services implementation compilation.
- File management and distribution.

This is done automated by a Java building tool called ANT (from Apache project). A build.xml file is needed, specifying some parameters (files location, Java packets location, etc.) and the gar file is generated. If this build.xml is configured and executed from Eclipse, the file ethernetSwitch.gar is obtained.

This is shown in the figure below:

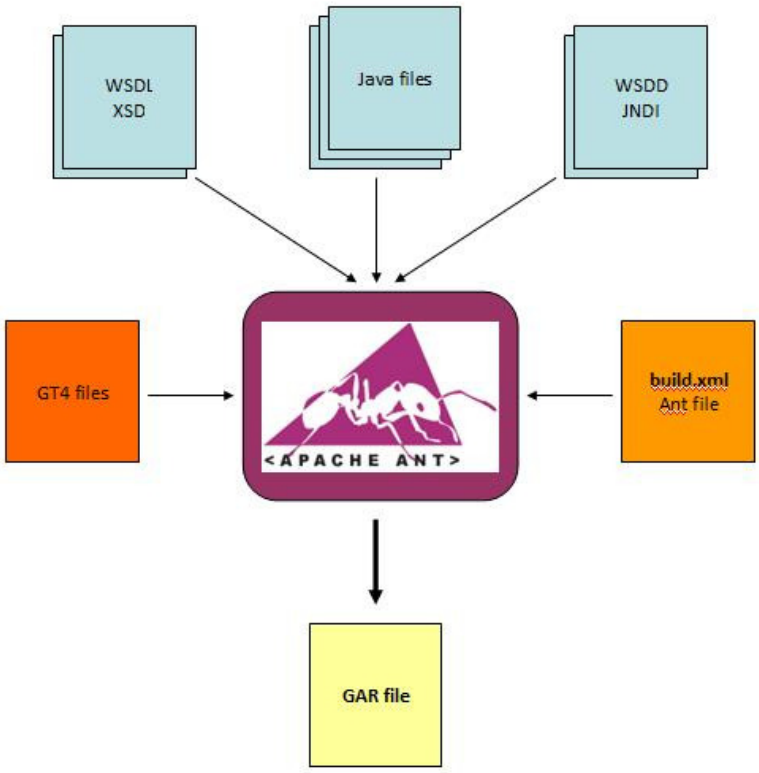

Figure 2-5. GAR file Generation with ANT

5. Deploy the service into a Web Services container

Globus Toolkit manages the deploy process, using Ant to unpackage the gar file, and copying the content files (WSDL, stubs, WSDD, etc.) into a structured tree inside the Globus directory.

Once in the server the GAR file is deployed and undeployed with these instructions:

Deploy:

```
globus-deploy-gar
$GAR_LOCATION/cat_upc_core_services_ethernetswitch.gar
```

Undeploy:

```
globus-undeploy-gar cat_upc_core_services_ethernetswitch
```

Once this step is done, the only step that has to be done is to activate the container:

```
globus-start-container -nosec
```

Once the container is activated, the deployed services URI are shown. We can see the implemented services among them:

```
http://147.83.118.238:8080/wsrf/services/core/EthernetSwitchService
http://147.83.118.238:8080/wsrf/services/core/EthernetSwitchFactoryService
```

Once the services are initialized, the existing Ethernet Resources in the database are loaded to be remotely accessed.

The complete process is similar for EthernetResource Web Service, being different only the operations done over the WS and the resources parameters. EthernetResourceService is the service that manages the virtualized interfaces instances. Two more URIs are obtained from this service:

```
http://147.83.118.238:8080/wsrf/services/core/EthernetResourceService
http://147.83.118.238:8080/wsrf/services/core/EthernetResourceFactoryService
```

EthernetResources resources present some important fields, some for layer 3 interface representation (IP addresses, etc.) and the following concerning to general properties of the interface:

- networkElement: The virtualized interface name is contained in this field, keeping the format *interfaceName.unitName (ge-0/0/3.0).*
- physicalBinding: It contains the physical device name where the interface is placed.
- owner: It contains the name of the user which the resource has been assigned to.
- vlans: This field contains the VLAN assigned to the interface.
- isTrunk: It determains the access or trunk configuration of the interface.

All these properties described previously present persistency in the database and they have been added up during the process of the creation of the application.

2.2 Graphical user interface

2.2.1 Introduction

The graphical user interface (GUI) is a simple interface to offer to the user the main actions that the tool is able to do. This tool is based in the Eclipse Rich Client Platform (RCP), a set of libraries that offers the developer the possibility of managing components to be used to form a final rich client application (in a similar aspect to Eclipse). For more information about RCP see section 0.

This interface offers the view of the devices and the resources configured in each of them. The user is able to add new devices to the tool, thanks to the wizards that will guide him. In each one of these configured devices, the user will be able to see and configure the VLANs and the interfaces.

This GUI presents some functionality limitations that will be solved for the final application to be delivered for the FEDERICA project. For instance, all the application is prepared to assign multiple interfaces to a VLAN at time, while the user interface only allows one at time. Improving this in the graphical interface would reduce the configuration time, because just one connection to the switch would be necessary, instead one connection for each action.

Eclipse RCP applications can be presented as stand-alone applications or as Eclipse Plug-in as well. In our case, due to the early stage in the development of the user interface, the application is presented as an Eclipse pug-in.

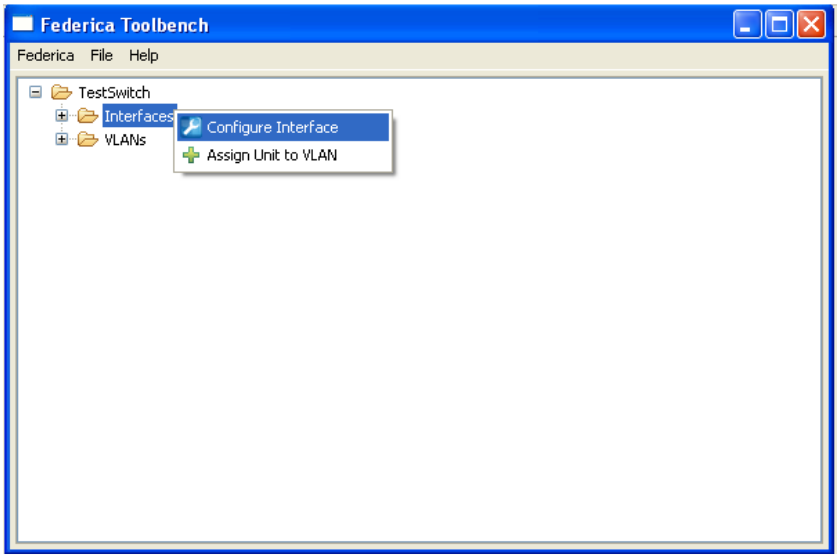

Figure 2-6. GUI general view

2.2.2 Structure

There are some important classes that define the interface structure. Over this main structure, the GUI is developed. The main classes that compound this structure are:

- Plugin.xml: This file contributes and positions the actions, views, perspectives, wizards, extensions, etc.
- MANIFEST.MF: The manifest includes a description of the plug-in. This file is generated when the application is executed, and it contains the loaded libraries when the application starts and the rest of the execution-related information such as classpath, etc.
- Application: Application is the main execution class, is where the main routine of the application is specified. The main actions to be done in this light class are to create the workbench where the parts of the interface will be placed and to add an ApplicationWorkvenchAdvisor to it.
- Perspective: Each Perspective class contains the different sets of views of the application. Several Perspectives can be defined, and one of them must be initialized. Each one of these perspectives defines a layout with the views that that perspective will include. This can be navigation windows, editors, etc.

In our project, two perspectives have been defined: The former is the perspective for the NOC, where all the resources are shown by device membering, and the latter is the perspective where the resources are shown depending on the user each resource belongs to. Both are similar in aspect.

- Advisors: Once the application windows are open and configured, the advisors give advice (as their name implies) to the workbench. Advisors are aggregated to the workbench. For instance, when the workbench opens a window, it asks an advisor to determine whether or not it should include a menu bar calling the WorkbenchWindowAdvisor.preWindowOpen() method.
    - ApplicationWorkbenchAdvisor: The default perspective is defined by this advisor.
    - ApplicationWorkbenchWindowAdvisor: Window configuration parameters (such as size, title, etc) are defined by this advisor.
    - ApplicationActionBarAdvisor: The tools bar (or action bar), the state bar and the menus are configured by this advisor.

In the next figure is explained the lifecycle of the Workbench advisors.

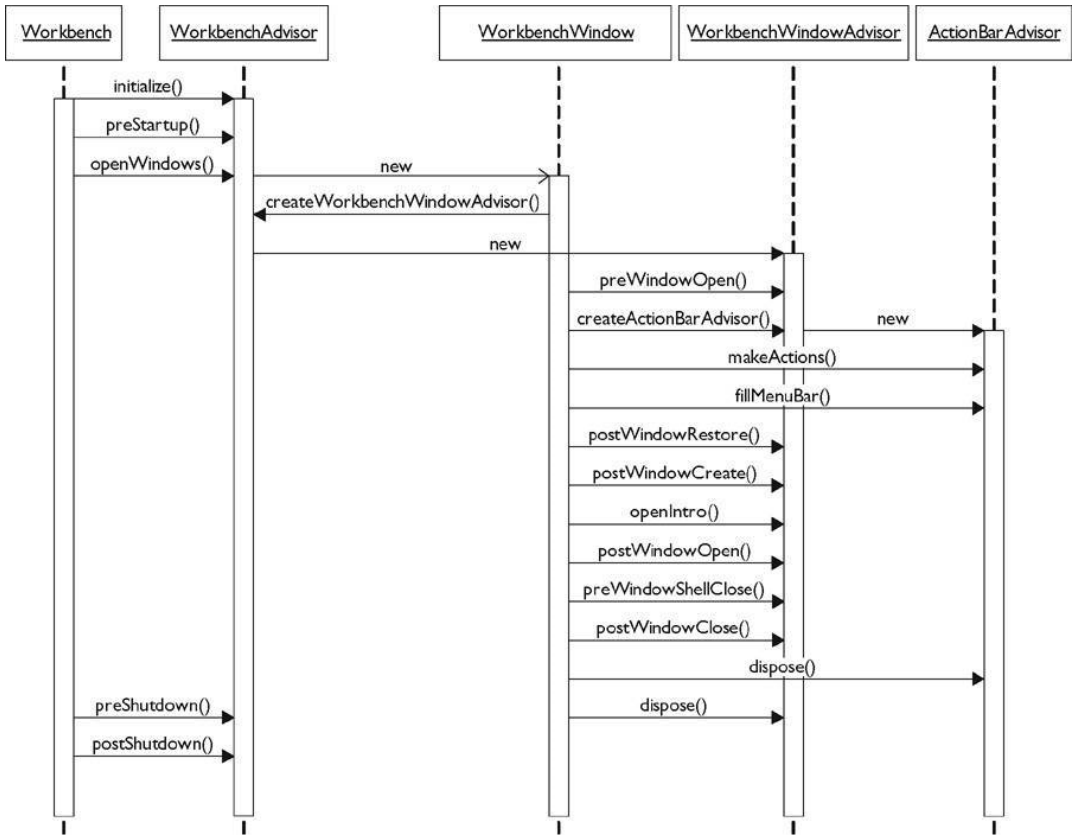

Figure 2-7. Workbench Advisors lifecycle

When the Workbench runs, it allows the WorkbenchAdvisor to participate from the very beginning by calling WorkbenchAdvisor.initialize(). This happens before any windows are opened. As each window is opened, a WorkbenchWindowAdvisor is created and associated with that window instance.
This WorkbenchWindowAdvisor instance is consulted throughout the window's lifecycle. The ActionBarAdvisor is created by the WorkbenchWindowAdvisor before the window is opened and participates in populating the menu, toolbar, and status line of each window.

The following figure shows an approach to the architecture of the user interface, with the basic components that compound it:

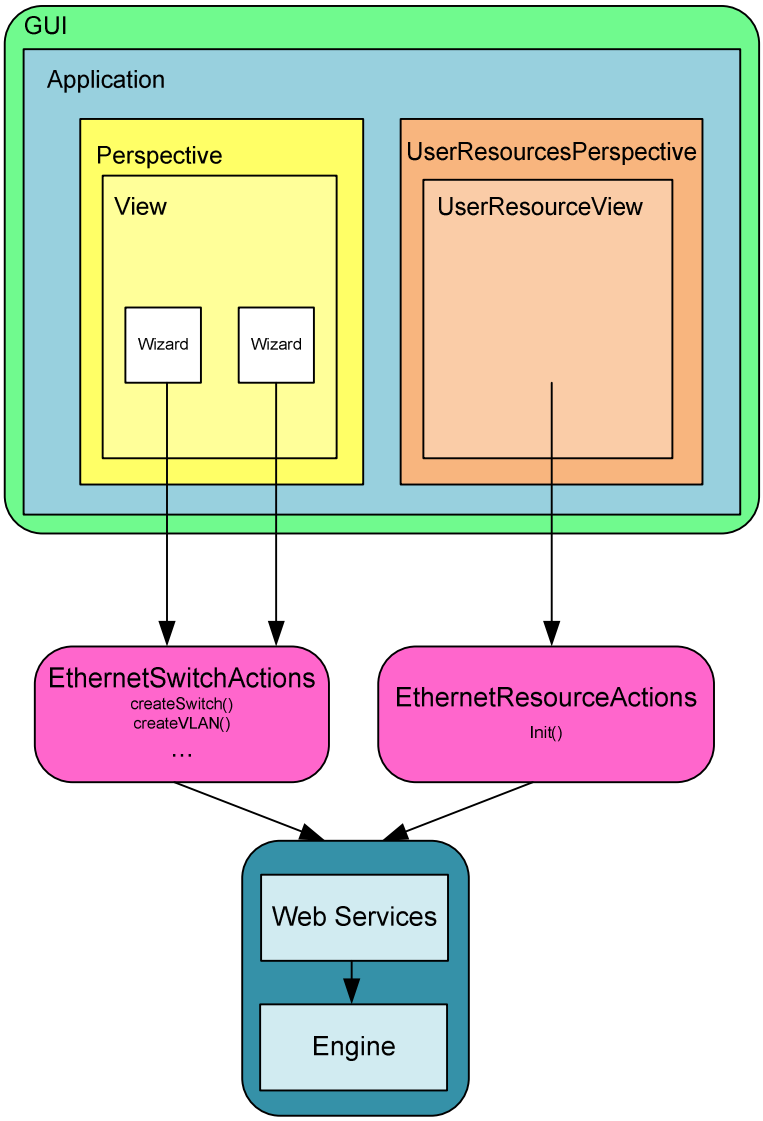

Figure 2-8. Basic components of the graphical user interface

The project specific classes are the following:

- Navigation view: This view contains a tree showing the resources (the interfaces pending of the switch where are placed, and the VLANs created on each switch). From these elements the user can select to do some of the specific actions to each type of element (some actions for VLANs, some actions for interfaces). The switches and switch configuration is listed from the database, through web services communication.
  As has been mentioned before, pending of each switch in the list are placed the interfaces and the VLANs configured in that switch. Pending of the physical interfaces are listed the logical interfaces, each one of them with its own configuration, as description, if it is an access or a trunk port, etc. In this view, new switches can be added through a specific wizard.

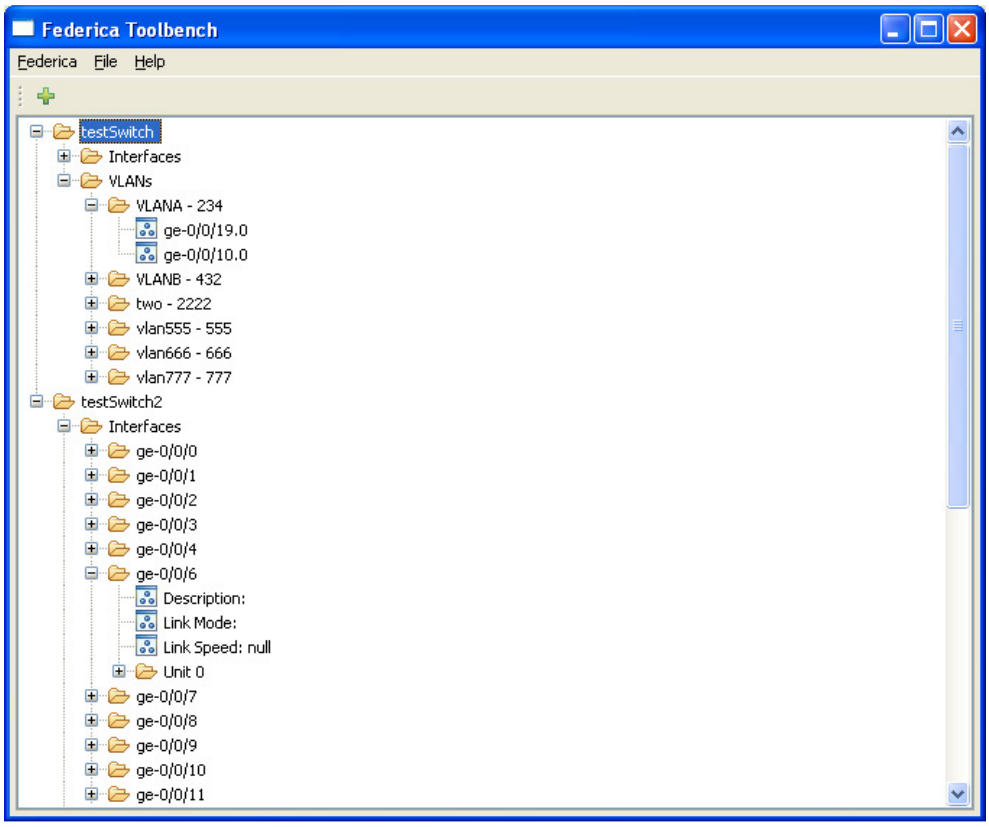

Figure 2-9. Navigation view listing all the resources in the devices

- UserResourcesView: This view is similar to NavigationView, but instead to order the resources depending on the physical device they are allocated, here the resources are ordered depending on the user they have been assigned. This way, the NOC can see which resources each user has in the network.

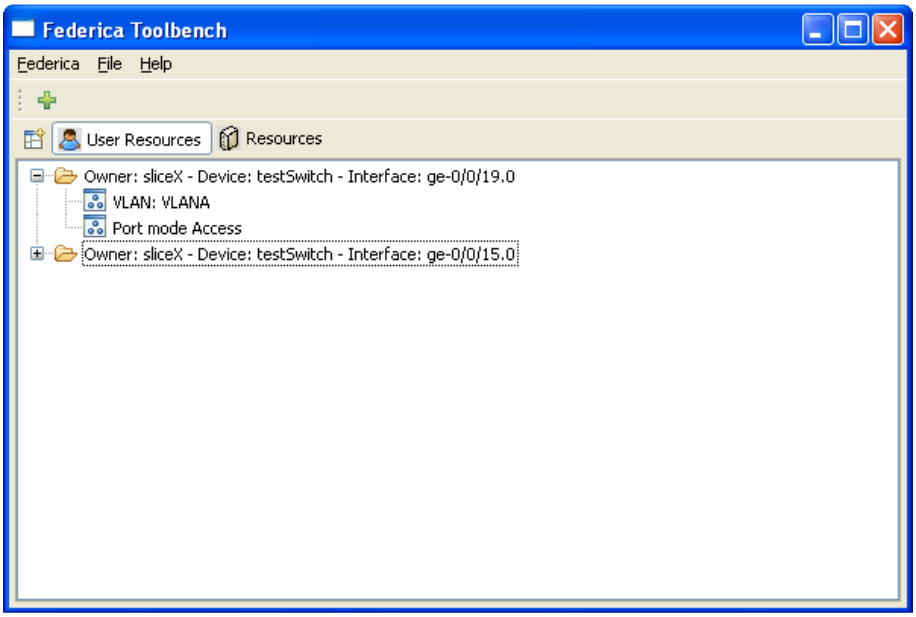

Figure 2-10. User Resource view

- Wizards: Several wizards have been developed in order to make easier the configuration of the resources for the user of the application, dividing this resource configuration in different steps, showed in panels, one per each configuration step. Navigation among these panels is done thanks to the typical Next and Back buttons. Some of the next buttons must be conditional; this means that without certain complete information in the current wizard panel, next button is not activated. As usual in the wizards, a cancel button exists. The following wizards represent the possible actions to be implemented in the application:
    - AssignVlanToInterface: An interface is assigned to an existing VLAN. This is done in three steps, VLAN selection, available interfaces, and final result of the operation.
    - ConfigureUnit: This three step wizard configures a logical interface. First of all, the interface is selected. In the second step, the new configuration is introduced and the first step is to confirm the success of the operation. For the moment, the unique parameter to be configured of the interface is to define if it is access or trunk. In a future implementation, more parameters will be configurable.
    - CreateSwitch: This wizard is used to add a new switch to the infrastructure, and has two steps. The first one to introduce the configuration information and the other one to confirm the success of the operation.
    - RefreshSwitch: This wizard is used to force the update of the information of the switch. The first is a confirmation step, and the second one is the information about the success of the action.
    - CreateVLAN: This wizard creates a new VLAN in the selected device (in the NavigationView). The first panel is to introduce the new VLAN information (name and ID) and the second one is to confirm the successful creation.
    - DeleteVLAN: This wizard deletes a VLAN from a device. The first step selects the VLAN to delete, and the second one confirms the operation.
    - Virtualize: This wizard creates a resource to be assigned to a network user. This is a three step wizard. The first step selects the physical interface to virtualize, the second one selects the VLAN to be given to the user and the owner as well, and finally the third step shows the result of the operation.

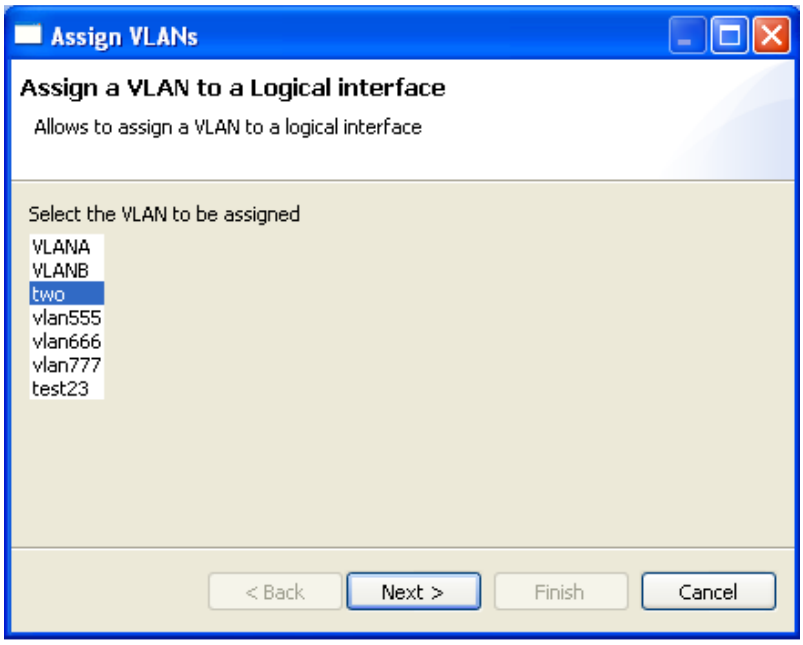

Figure 2-11. VLAN assignment wizard

- EthernetSwitchActions: This class manages the communication with the different Ethernet Switch Web Services. It contains a relation with all the switches and their configuration, and the actions required to realize all the operations. It interacts directly with the factory and with the web services instance.
- EthernetResourceActions: This class manages the communication with the Ethernet Resource Web Services. All virtualized resources are listed in this class as well as the actions required to present the resources in the “tree-form” of the program, shown in the UserResourcesView. When this class is initialized it obtains from the corresponding service the corresponding resource list stored in the database, being help by auxiliary classes to catch and understand the stubs information.

2.3 How do the modules interact?

Once all the different components in the program are known, we are going to explain how they work together. Some working examples are going to be explained.

2.3.1 Graphical interface initialization

When the application is executed, the graphical user interface (GUI) is initialized. This GUI obtains a list with the stored switches and their corresponding configuration. This is done by the init operation, the responsible to interact with the web services (EthernetSwitchActions).

The graphical interface communicates with the EthernetSwitch WS. One of the stubs of the client is a locator, concretely factoryLocator that taking the WS URI returns the WS factory portType (the class containing the service). The operation sent this moment is the search operation (*find*). If a new switch would be added, the operation to be sent would be *create*, having all the same process.

The find operation tries to find out all the already existing devices stored in the database. This request is interpreted by the corresponding stub and it is transformed into an Endpoint references (WS-Resource directions) vector inside the findResp. The information of each one of the elements contained in the vector is obtained. This is to say that the configuration of every switch is obtained, thanks to the function getEthernetSwitch that access to the resource to obtain this information, which will be added finally to EthernetSwitchActions. NavigationView will access to this list to represent the switches in the interface.

This is reflected in the following figure:

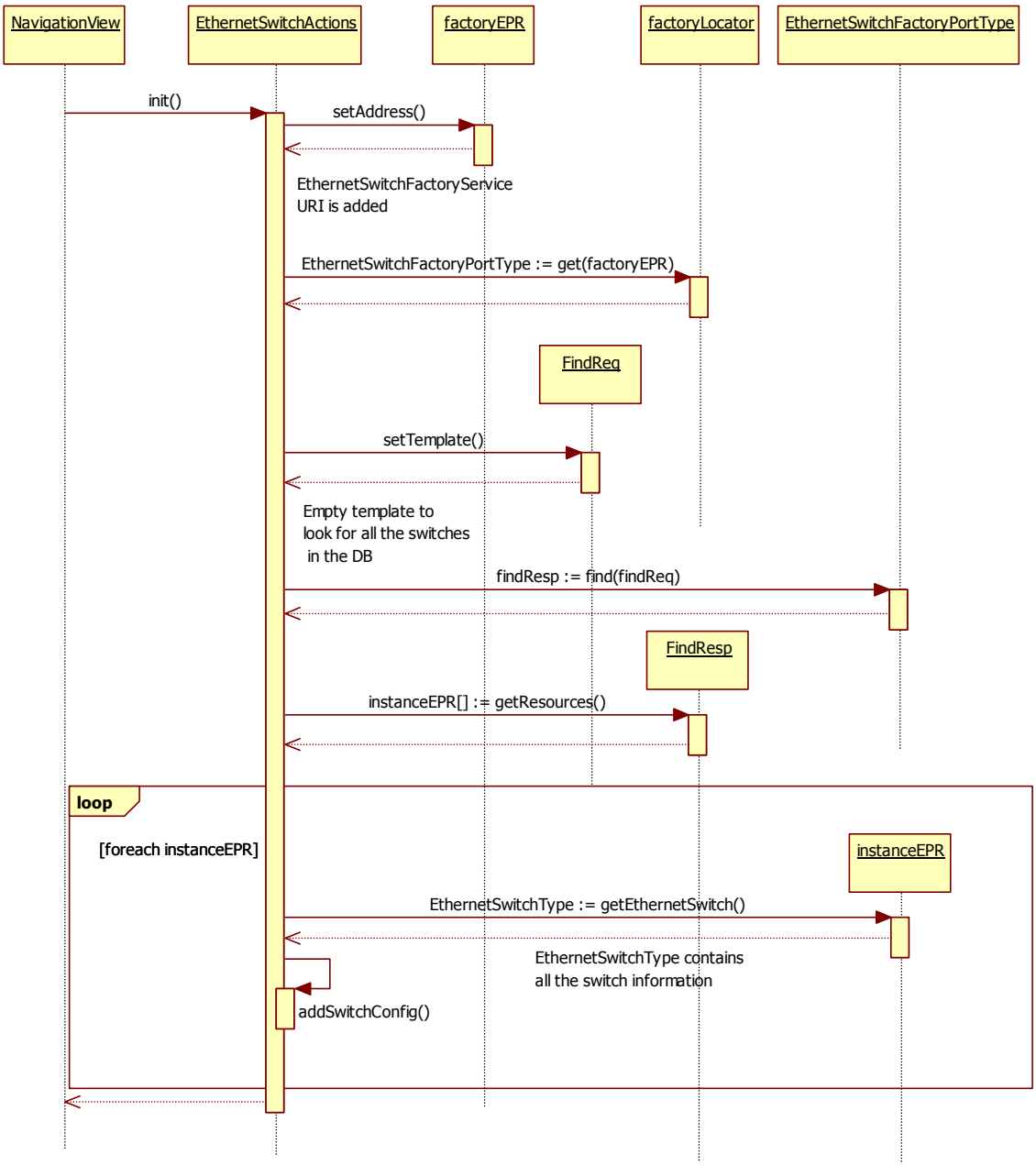

Figure 2-12. Graphical interface initialization

2.3.2 Switch addition

Once the graphical interface is loaded, maybe the NOC wants to add a switch to the network. There is an option in the menu, to start a wizard that will guide him step by step. Name, IP address, and some other properties must be introduced, user and password as well. When the creation has to be done, EhernetSwitchActions communicates with Ethernet Switch Factory WS. Again, the process to find the portType is the same as the previous operation, but this time the operation to be sent is “create”. The factory sends the “create”

message to the EthernetSwitchResourceHome class, doing the proper over the switch resource (EthernetSwitchResoure).

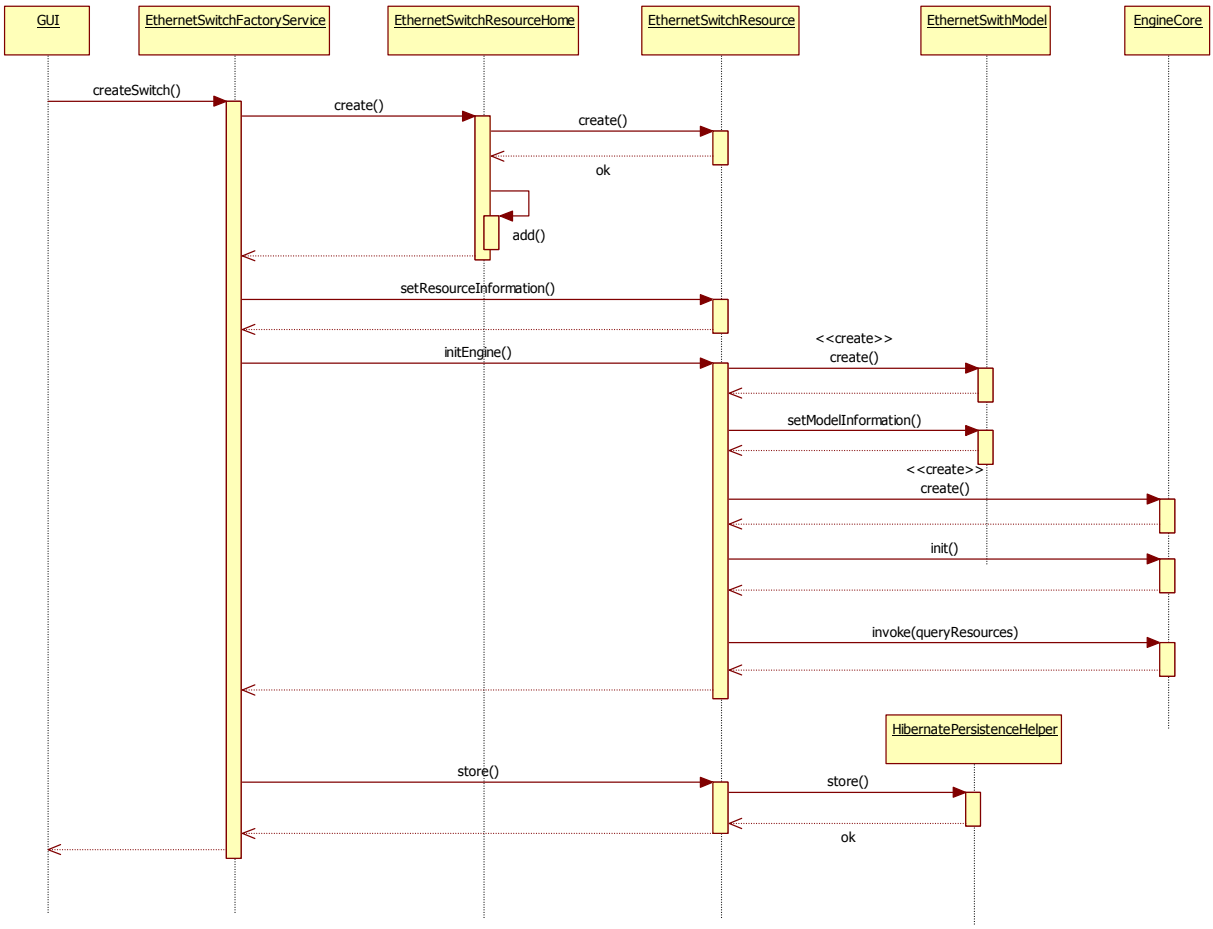

Figure 2-13. Switch addition

The switch is added to the switch list and its properties (IP, transport, protocol, etc) are set through setResource information as it is shown in this figure:

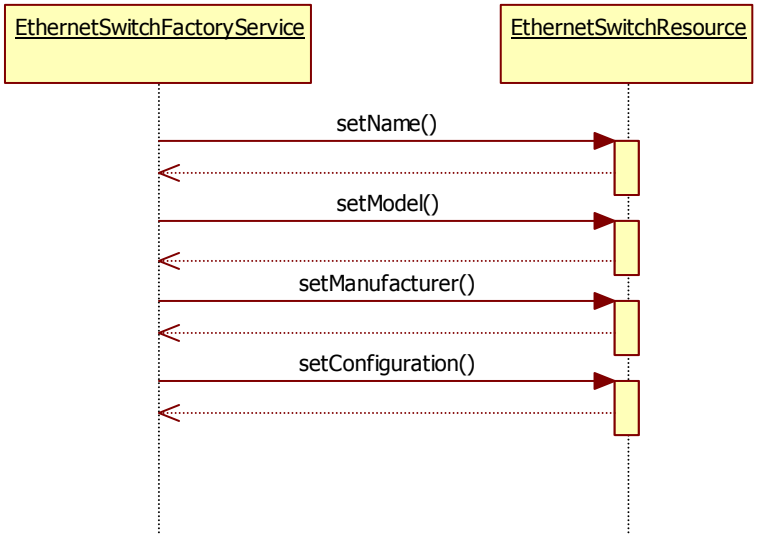

Figure 2-14. Resource configuration

Now, the communication between the WS and the Engine is going to start. The Engine is initiated (initEngine) through the EthernetSwitchResource. The engine will be the “driver” of the physical device. Once the model is created, the information needed for the communication is added.

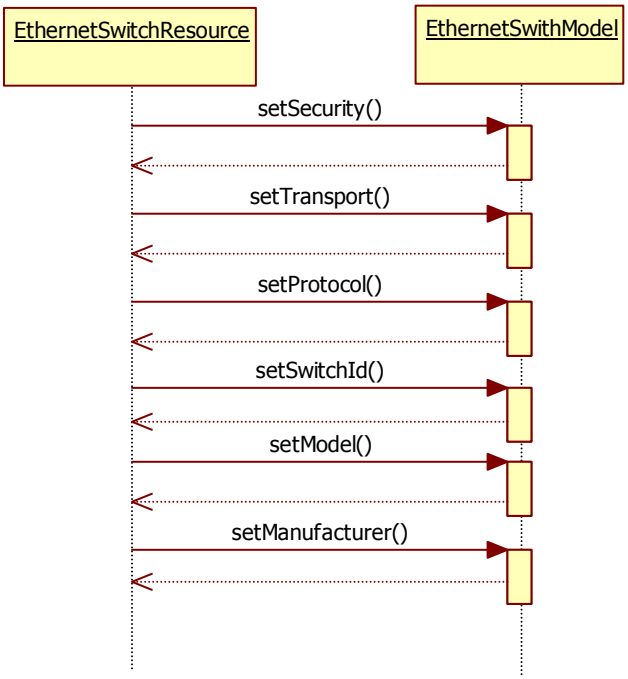

Figure 2-15. Parameters setting

Is now when the EngineCore receive the init operation; taking the new model, will initialize the transport and protocol management classes. The action

management classes are initialized as well. Those classes, in our case are SSHTransportHandler, NetConfProtocolHandler and ActionHandler. This processcan be seen in the following picture:

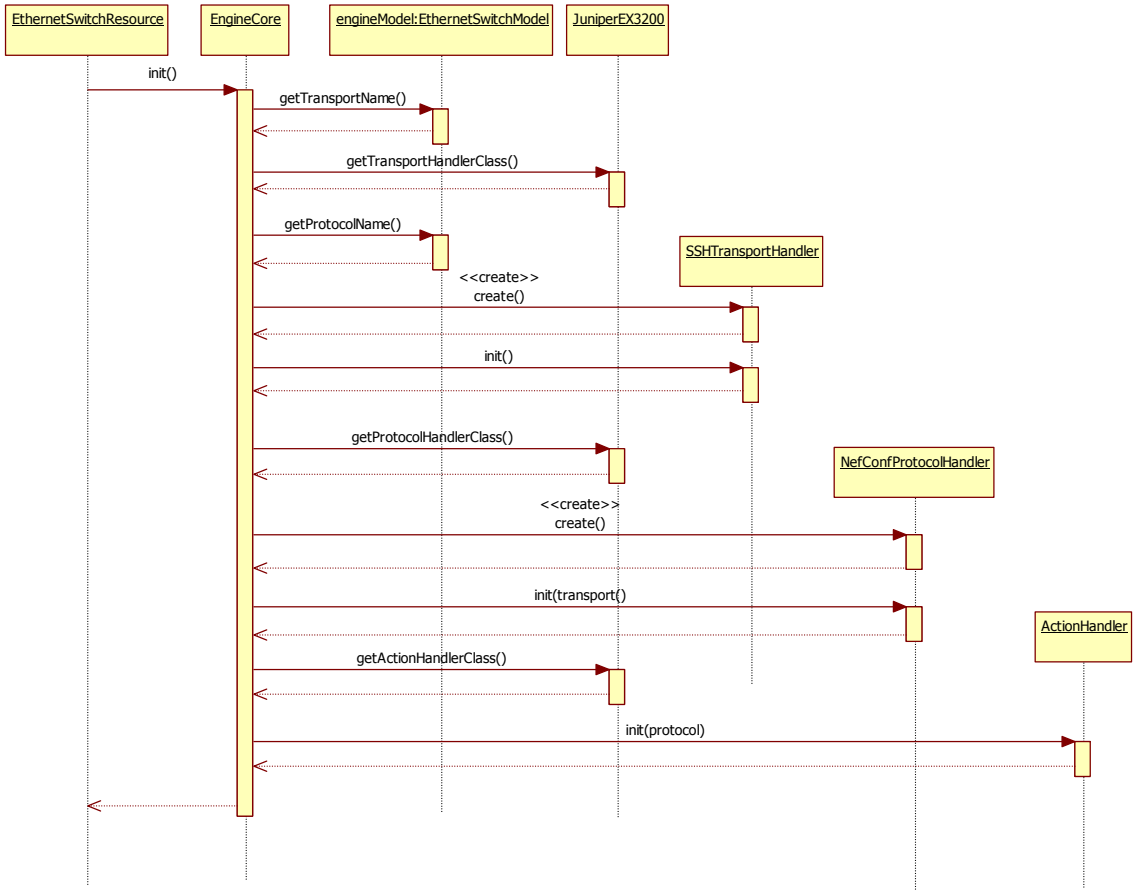

Figure 2-16. EngineCore initialization

2.3.3 Switch configuration load

To obtain the current configuration of a physical switch in the network (this is, for the moment, to obtain the configured VLANs and the interfaces configuration), the actionHandler of the switch model instance (stored in the EthernetSwitchResource) receives the “invoke” command with the queryResources action. The process that follows now would be the same for any operation, just changing the commands and the way the switch response is parsed into the model. (UML11)

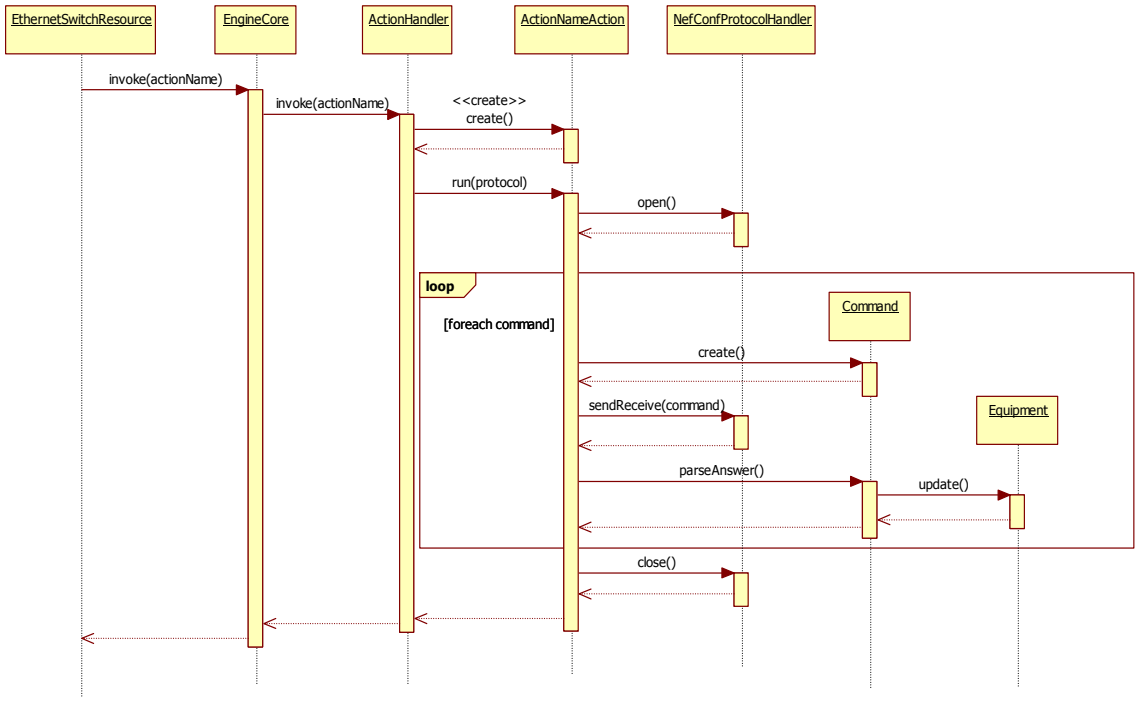

Figure 2-17. Generic invoke

ActionHandler in the EnginerCore creates the corresponding action. As we want to obtain the configuration in the switch, the action to be sent will be QueryResourcesAction. ActionHandler will invoke the protocol handler, which will invoke the transport handler. These steps are necessary to establish the communication with the switch. Through this communication channel, the corresponding commands would be sent to the switch. In our case, the command is a Netconf message (with XML form), with the corresponding code to ask for the VLANs and interfaces configuration.

This message is the following:

```
<rpc>
     <get-config>
          <source>
               <candidate/>
          </source>
          <filter>
               <configuration>
                    <interfaces></interfaces>
                    <vlans></vlans>
               </configuration>
          </filter>
     </get-config>
</rpc>
```

As has been explained before, the commands are sent to the switch via SSH and the switch sends a response with the configuration, in a message similar to the following one:

```
<interfaces>
        <interface>
          <name>ge-0/0/0</name>
               <unit>
                  <name>0</name>
                  <description></description>
                  <family>
                     <ethernet-switching>
                        <port-mode>trunk</port-mode>
                        <vlan-id>120</vlan-id>
                     </ethernet-switching>
                  </family>
               </unit>
        </interface>
(…)

</interfaces>

<vlans>
        <vlan>
             <name>VLANA</name>
             <description>testing</description>
             <vlan-id>234</vlan-id>
             <interface>
                 <name>ge-0/0/0.0</name>
             </interface>
             <interface>
                 <name>ge-0/0/19.0</name>
             </interface>
        </vlan>
(…)

</vlans>
```

More information about the Netconf commands can be found in section **¡Error! No se encuentra el origen de la referencia.**.

These response messages are parsed by the class ObjectMapper that obtains the VLANs and the interfaces configuration from the XML code sent by the switch (selecting only the useful information), and saves this configuration into the model Equipment.

Then the web service resource has loaded the initial state of the switch. With that, the factory stores such information into the database to obtain persistence for the resource. This is done with the class HibernatePersistenceHelper. Finally, the graphical user interface, updates the tree showed to the user with the new information.

2.3.4 Switch configuration

Now the switch has been added, and the current configuration has been loaded into the program, we are going to explain how to make changes in the configuration of this existing switch. For instance, we are going to add a new VLAN to the switch.

In the GUI the user will right-click over the switch he wants to add the new VLAN and he will select “Create VLAN”. This action will start the corresponding wizard that will guide the user through the configuration steps, making him introduce the different configuration parameters (VLAN ID and VLAN name).

The GUI will look for the EthernetSwitch WS from its factory, in a similar procedure than the explained before and shown in the figure UML6. Once the service instance is obtained, an invoke operation is sent using the invokeReq message, and containing the data introduced by the user. The resource is obtained from the WS and from this its engineModel. The invokeReq message, with the operation will be parsed into the model. Depending on the operation sent, the model will be filled in different ways. For instance, for the createVLAN request, a VLANType of the Engine will be filled and placed into requestedVLANs vector. At the same time, some interfaces could be added to the VLAN in the same step, but as we told before, the GUI limits for the moment this option. Now the *invoke* message is sent to the engine proceeding in the same way than figure UML6.

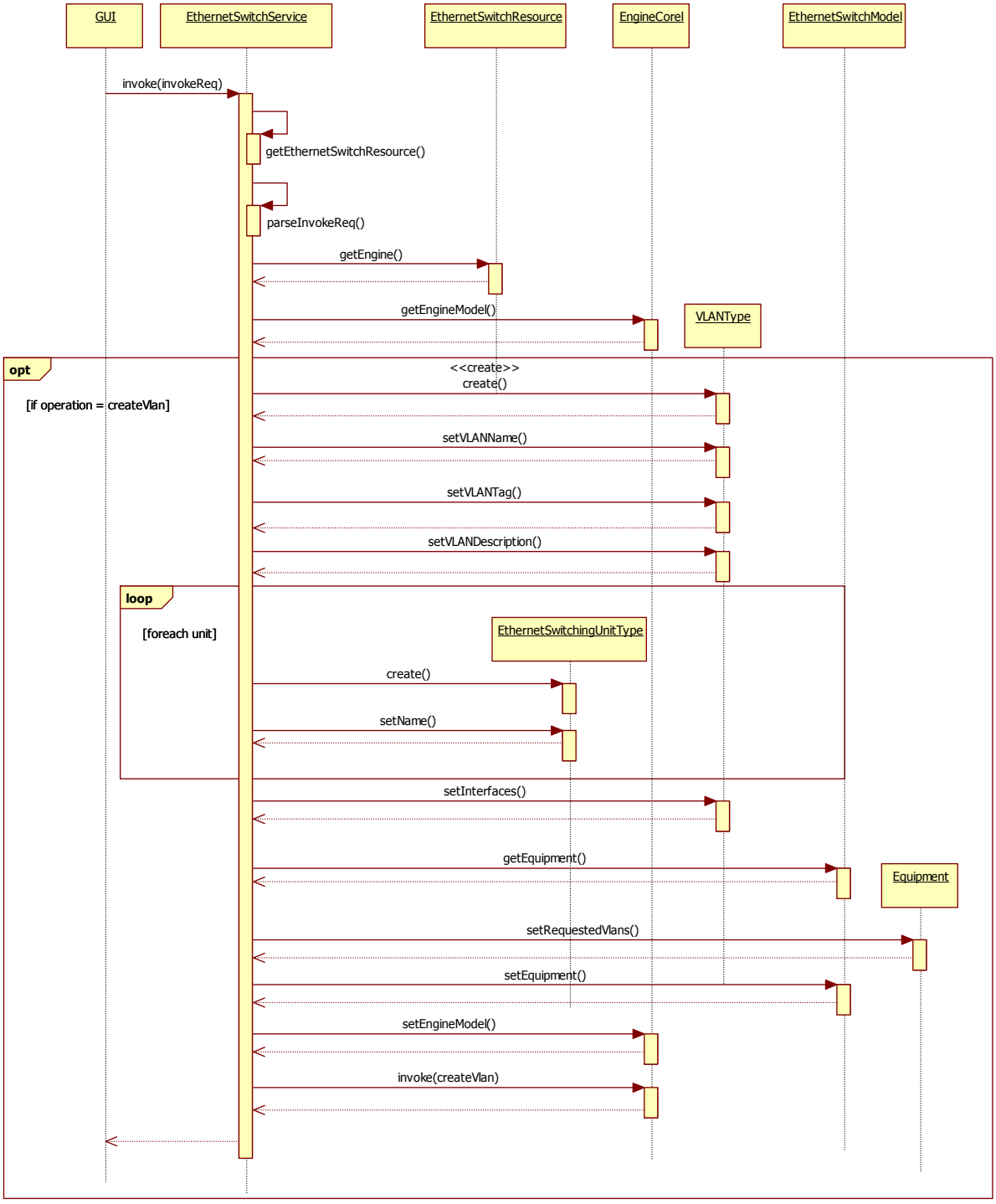

Figure 2-18. Invoke containing create VLAN operation

All the other operations that operate over the switch (virtualize interface, delete VLAN, etc.) proceed in a similar way to what has been explained here, but changing the content of the messages, and the configuration sent to the switch.

For instance, we can see the Virtualize operation process in the following figure:

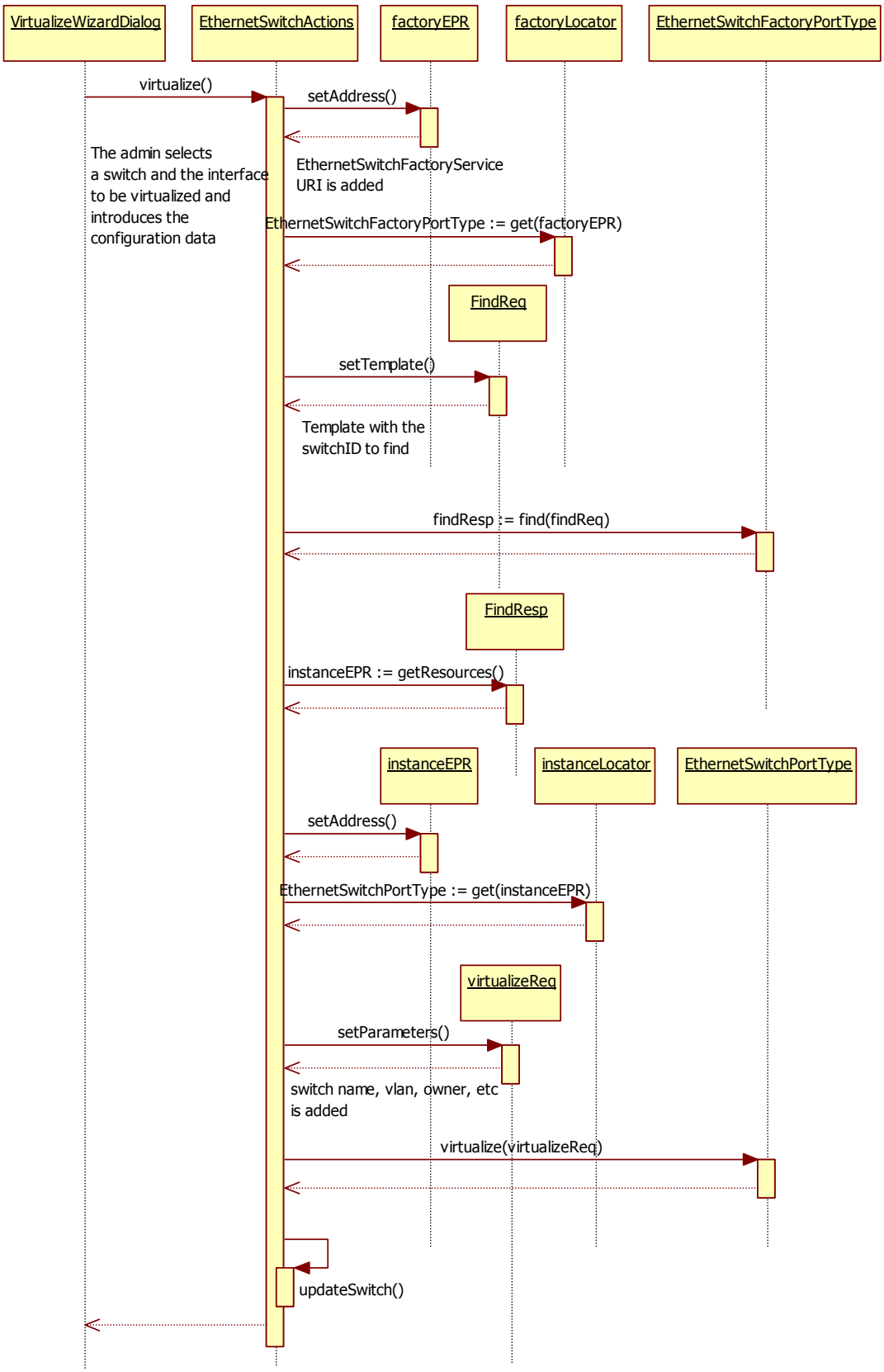

Figure 2-19. Virtualize over EthernetSwitchActions