Títol: Herramientas para virtualización y suministro de red a nivel 2

Alumne: Álvaro Monje Hernández

Director/Ponent: Roc Messeguer Pallarès

Codirectora: Cristina Cervelló-Pastor

Departament: Arquitectura de Computadores

Data: 16 de junio de 2009

DADES DEL PROJECTE

Títol del Projecte: Herramientas para virtualización y suministro de red a nivel 2

Nom de l'estudiant: Álvaro Monje Hernández

Titulació: Ingeniería Informática

Crèdits: 37,5

Director/Ponent: Roc Messeguer Pallarès

Codirectora: Cristina Cervelló-Pastor

Departament: Arquitectura de Computadores

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Dolors Royo

Vocal: Anna De Mier Vinué

Secretari: Roc Messeguer Pallarès

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Tabla de contenidos

1 Introducción 6

2 Escenario 9

2.1 El nuevo Internet 9

2.2 Concepto 10

2.3 Infraestructura 11

2.4 Investigaciones expuestas 13

3 Objetivos 14

3.1 Objetivos de FEDERICA 14

3.2 Objetivos del PFC 16

4 Metodología 19

4.1 UML 19

4.2 Metodología en espiral 21

5 Entorno de programación 23

6 Desarrollo del proyecto 25

6.1 Visión completa del proyecto 26

6.2 Casos de uso y diagramas de secuencia genéricos 28

6.3 Descripción de las tareas 37

6.3.1 Estudios 37

6.3.2 IaaS Framework 45

6.3.3 Desarrollo del Engine 46

6.3.4 Desarrollo de los Web Services 56

6.3.5 Desarrollo de la GUI (Graphical User Interface) 70

6.3.6 Interacción de los módulos 78

6.4 Tests 93

6.4.1 Tests del Engine 94

6.4.2 Tests de los Web Services 95

6.4.3 Tests de la GUI 96

6.4.4 Tests de las VLANs sobre las máquinas virtuales 97

6.4.5 Demo 102

7 Trabajo por realizar 105

8 Planificación 107

9 Conclusiones 109

10 Glosario 113

11 Bibliografía 116

12 Anexo A – Herramientas utilizadas 117

13 Anexo B – Estudios de Q-in-Q y QoS/CoS 121

1 Introducción

En esta era de nuestra historia, donde Internet forma parte de todos los ámbitos de la sociedad actual, los investigadores necesitan infraestructuras sobre las cuales desarrollar sus pesquisas. Infraestructuras que ofrezcan seguridad, flexibilidad y solidez para llevar a cabo su trabajo. El proyecto europeo FEDERICA pretende satisfacer estas necesidades de una forma innovadora pues no hay estándares ni referencias robustas sobre las posibilidades que se brindan a este tipo de usuarios.

FEDERICA (Federated E-infrastructure Dedicated to European Researchers Innovating in Computer network Architectures) nació el 1 de enero de 2008 y finalizará el 30 de junio de 2011. Se trata de un proyecto financiado por la Unión Europea (Séptimo Programa Macro) que pretende ser la primera red experimental europea para investigaciones en el ámbito de redes e Internet.

FEDERICA proporcionará a los investigadores *slices* (rebanadas, rodajas) sobre una infraestructura de red física que podrán ser repartidas como recursos virtuales para sus experimentos. Se trata de una infraestructura tecnológicamente agnóstica; consta de equipos de transmisión (routers, switchs) interconectados por circuitos Gigabit Ethernet (o túneles MPLS) y máquinas capaces de virtualizar con el objetivo de hospedar actividades de investigación de nuevos protocolos y arquitecturas de Internet.

La infraestructura de FEDERICA está basada en las redes europeas multi-gigabit de investigación y educación (European Research & Education multi-gigabit Networks). Sus conexiones utilizan los puntos de presencia (PoPs) de NREN (National Research and Education Network) y GÉANT2. Éstos proporcionan a FEDERICA nodos capaces de ser virtualizados. Esta virtualización de los elementos de la infraestructura es la que permite crear una serie de rodajas de la red física, las cuales serán asignadas a los investigadores para que testeen cualquier tipo de experimentos. Incluso aquellos que puedan ser peligrosos a nivel de red, evitando de esta manera efectos colaterales e interferencias sobre otras redes de desarrollo existentes. Todo ello sobre un sólido sustrato. Los usuarios tendrán pleno control sobre el entorno que les haya sido

proporcionado y no habrá interferencias (pero con la posibilidad de interactuar) con el resto de *slices*.

El objetivo es que un usuario final disponga de una red personalizada para trabajar con ella cómodamente. Para ello, el sujeto solicitará todos aquellos recursos que sean necesarios para sus investigaciones: una configuración base sobre la cual podrá hacer lo que desee: crear una red, establecer protocolos, monitorizar recursos… Será capaz de experimentar tanto a nivel IP como de enlace de datos. Además, todos los servicios proporcionados por FEDERICA tendrán, debido al ineludible límite que acarrea una infraestructura física, ciertas restricciones (número de *slices* simultáneas, ancho de banda asignado a cada una de las interfaces virtuales…) así como las garantías necesarias sobre los recursos adquiridos.

Para llevar a cabo todo lo descrito en el párrafo anterior, existirá un centro de operaciones de red (NOC) que será el responsable de gestionar los recursos de la infraestructura. Comprobará que todo pueda ser proporcionado (según las políticas de usuario) y creará la *slice* para el usuario: máquinas virtuales (VMs), arranque de los sistemas operativos, routers lógicos, virtual LANs…

Todo lo anterior es una visión global del proyecto FEDERICA. No obstante, el proyecto final de carrera que aquí se presenta abarca una pequeña parte de éste. Es necesario tener una idea general de todo el producto (el cual se irá conociendo a medida que se avance en la memoria) para entender sus características específicas. Como dice el título, este PFC se ocupa de las herramientas para crear *virtual slices* y satisfacer las necesidades de la red a nivel de enlace de datos (nivel dos de la pila OSI). Es una parte fundamental de FEDERICA ya que la configuración a este nivel es indispensable a la hora de crear los *slices*. En este ámbito encontramos la gestión de VLANs (creación, destrucción, configuración) y manejo de las interfaces lógicas, entre otras cosas. También hay que tener en cuenta aspectos de calidad de servicio (QoS) y/o clase de servicio (CoS). Se entrará en detalle en los puntos siguientes.

Por tal de hacer de la memoria un instrumento útil y manejable, se establece una estructura contundente y ordenada. Se proporciona una visión general del proyecto FEDERICA: un telón de fondo sobre el cual sea más sencillo captar su esencia. También se mostrarán más ampliamente los objetivos de ambos (FEDERICA y PFC). Antes de exposición del trabajo desarrollado, se proporciona al lector información sobre la

metodología empleada y el entorno de programación por tal de que se haga una idea de todas aquellas herramientas que se han utilizado en el proyecto. Con esta base, se presentarán los casos de uso y diagramas de secuencia generales de la aplicación para después explicar el estudio y elaboración de cada uno de los componentes que intervienen en ellos. Para acabar con el punto se verá la interacción detallada entre los módulos y los tests que se han llevado a cabo para comprobar el correcto funcionamiento de la aplicación.

Por último se hace un análisis del trabajo que todavía queda por realizar para abastecer todas las necesidades de FEDERICA así como la planificación del PFC y las conclusiones obtenidas después de su realización.

Al final de la memoria se adjuntan dos anexos: El anexo A recoge el listado de las herramientas que se han utilizado para desarrollar el PFC. El anexo B incluye los estudios realizados en paralelo a la elaboración de la aplicación pero que finalmente no se han implementado.

2 Escenario

2.1 El nuevo Internet

Internet está en continuo crecimiento; cambia en diversos aspectos: tamaño, uso, requisitos… Cada vez hay más a investigar; más posibilidades a exprimir en la red global. Y por lo tanto, cada vez hay más cosas que aprender. Es por estos motivos que ambos sectores (Investigación y Educación) necesitan herramientas para llevar a cabo sus funciones. Un proyecto como FEDERICA viene motivado por hechos como los anteriores y los siguientes:

- El crecimiento de la capacidad de la red. Cada vez se hace más difícil controlarla. Velocidades de 1 a 10 Gbps, mallas cada vez más complicadas, la incursión de la telefonía móvil, sensores…
- Necesidad de servicios para el crecimiento descrito en el punto anterior. Así como requisitos de seguridad, calidad etc.
- Importancia de las conexiones seguras, móviles y continuas.
- Los usuarios quieren cada vez más (en todos los aspectos).
- Se pretende innovar y experimentar con nuevos protocolos y arquitecturas; pero es complicado. Faltan muchas necesidades básicas para llevar estas investigaciones a buen puerto (más aun si pueden ser arriesgadas para el ámbito de la red).
- Incremento de las comunicaciones. Lo que implica poner a disposición de este enormemente amplio ambiente una serie de tecnologías y servicios que satisfagan sus exigencias (multi-dominio, tiempo real…).
- Internet hospeda cada vez más máquinas, además de los archiconocidos routers y switchs. Actualmente son indispensables para ciertos servicios que tiene que ver con el manejo, control y monitorización de la red.
- Muchos bancos de pruebas son locales. Muy cerrados y orientados a tecnologías demasiado específicas.

- “Virtualizar”. Es una palabra clave que está tomando cada vez más importancia. Significa (en el ámbito de Internet) crear redes virtuales a partir de una infraestructura física. Es aplicable a los niveles de datos y de red de la pila ISO/OSI. Por ejemplo, en la creación de routers lógicos a partir de un router físico. Es un concepto muy potente. Permite tener funcionando una gran variedad de tareas en una misma máquina física.

2.2 Concepto

Todo lo mencionado en el punto anterior son motivaciones por las cuales surgen proyectos como FEDERICA. Dichos asuntos no pueden ser absorbidos por redes de producción normales y necesitan de los frutos de las investigaciones que se están llevando a cabo actualmente. FEDERICA pretende facilitar el desarrollo de éstas últimas, consiguiendo además juntar el mayor número de investigadores posibles exportándolos, para que sean más efectivas, a nivel europeo.

El mejor modo de satisfacer este reto es creando una infraestructura orientada exclusivamente a los investigadores y capaz de abastecer los requisitos para todo tipo de experimentos.

FEDERICA desplegará y promoverá una infraestructura virtual sobre nodos existentes de NREN y GEANT2 junto con las herramientas necesarias (tool-bench) para gestionar el entorno virtualizado del que gozará cada usuario.

La siguiente figura muestra un esquema de la infraestructura de FEDERICA:

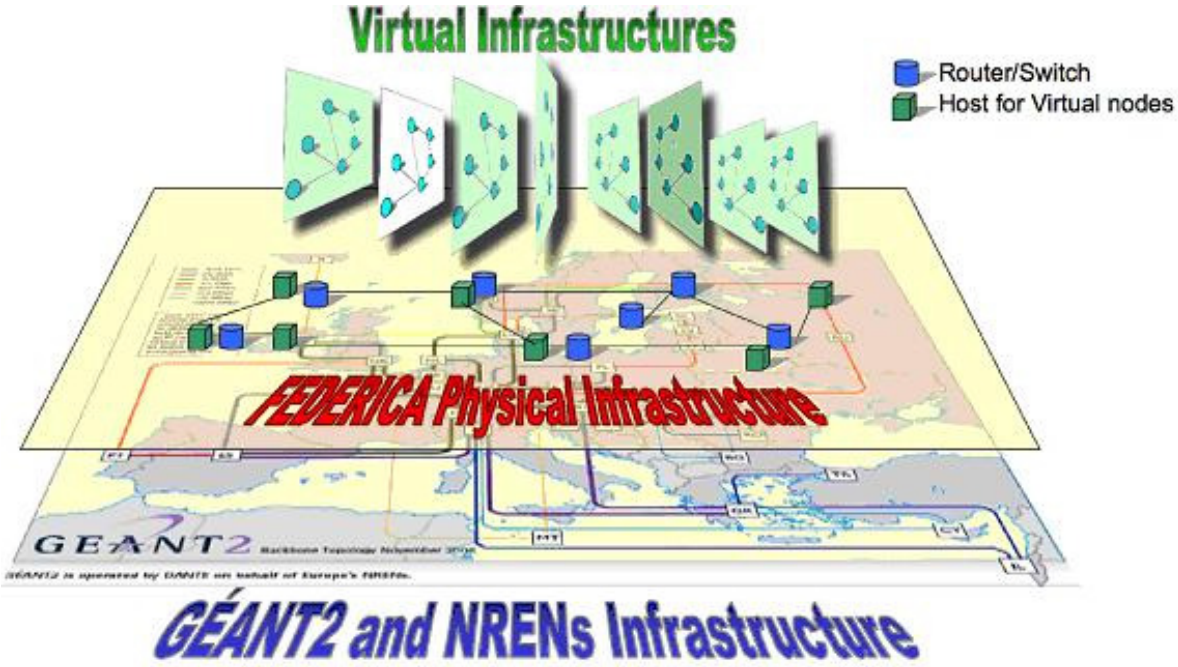


Figura 1: Infraestructura lógica de FEDERICA

Los grupos de investigación, tanto académicos como industriales, están trabajando continuamente en enfoques novedosos, protocolos, arquitecturas y herramientas *middleware*. Por lo tanto, este proyecto supondrá una oportunidad para unir sus fuerzas a todos aquellos que sientan interés por alcanzar lo antes posible el "Internet del Futuro".

2.3 Infraestructura

La infraestructura está basada en un conjunto de recursos físicos de red (ondas, circuitos, nodos) y una serie de conexiones virtuales creadas a partir de circuitos de NREN y GEANT2.

Cada punto de presencia (PoP), trece en total, contendrá recursos de red (hardware de red: routers y switchs) y de computación (servidores con software de virtualización). Cada PoP puede hospedar hardware de los usuarios y/o establecer una conexión con otra infraestructura ajena a FEDERICA.

Un elemento de computación es un nodo capaz de hospedar recursos virtuales tales como nodos finales, routers virtuales o elementos de medición. El usuario podrá

pedir una cantidad de éstos y configurarlos a su antojo (y/o pedirlos con una configuración previa por parte de FEDERICA).

La infraestructura será capaz de crear redes virtuales, o *slices*. Cada red virtual tendrá su propio dominio con su propia arquitectura, mapas de ruta, y servicios finales. FEDERICA albergará varios *slices* concurrentemente totalmente independientes (o conectados entre sí si se desea). Además, FEDERICA dispondrá de las herramientas necesarias para asegurar el control de la red, su correcta virtualización, monitorización etc. Dichos activos están basados en *Open Source* (código abierto) y correrán sobre máquinas virtuales (VMs) tales como VMWare o Xen.

Cabe subrayar que FEDERICA será capaz de acoger la prueba de aplicaciones de cualquier tipo. De hecho, la infraestructura está diseñada para evitar filtraciones respecto al tipo de arquitectura utilizada en el nivel de aplicación.

La siguiente figura muestra un mapa de la infraestructura física de FEDERICA:

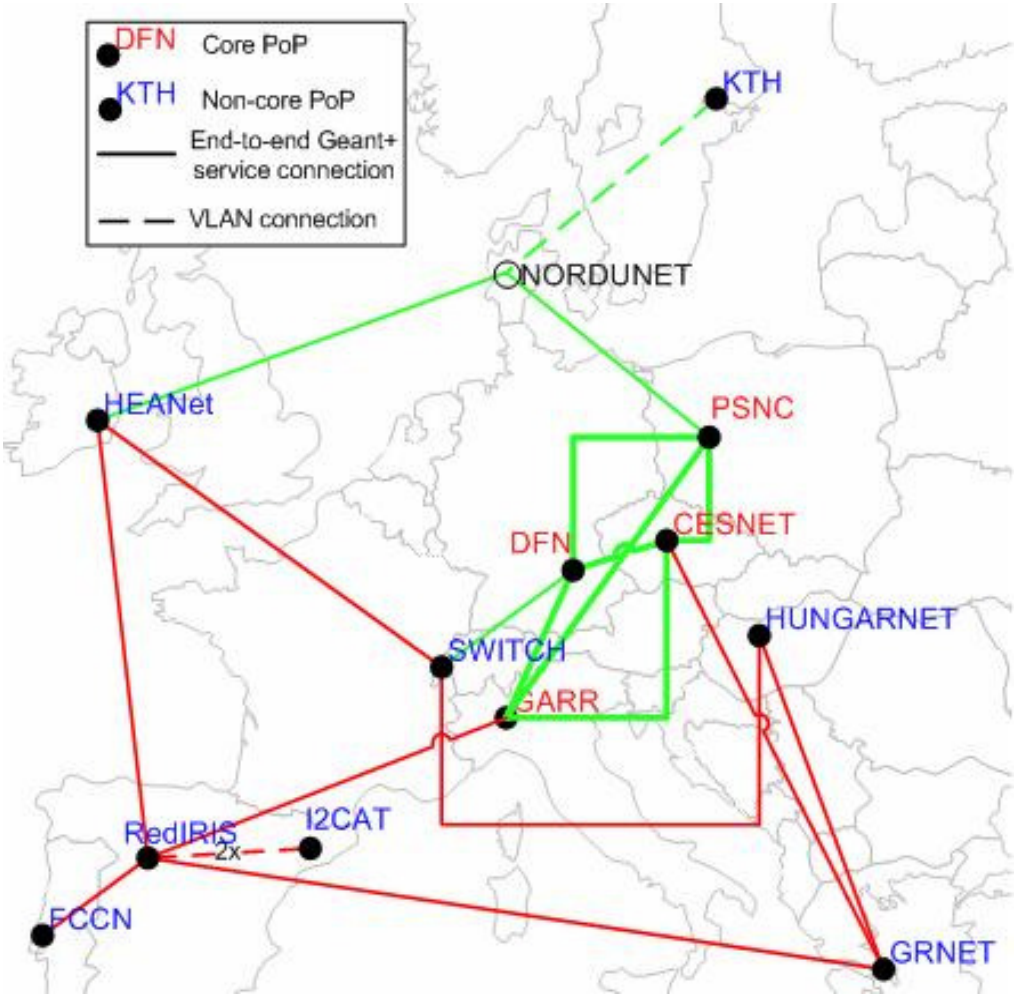


Figura 2: Infraestructura física de FEDERICA

Los PoP núcleo se encuentran en Alemania (DFN), Polonia (PSNC), República Checa (CESNET) e Italia (GARR).

2.4 Investigaciones expuestas

FEDERICA será una opción muy apropiada para aquellos investigadores que quieran llevar a cabo experimentos que potencialmente puedan dañar la red en algún aspecto. Debido a que cada prueba solo puede ser desarrollada dentro de su propio *slice*, el efecto sería automáticamente limitado y no se propagaría al resto.

Creando redes virtualizadas en la infraestructura, los científicos serán capaces de desarrollar nuevas herramientas y protocolos sobre todo tipo de dominios, de tecnologías y de vendedores, pudiendo evaluar interoperabilidad en todos los campos (datos, control, administración), escalabilidad y soporte a los usuarios. Situados en este contexto, se desplegará un esquema de manejo, habilitando una cooperación consistente con el resto de comunidades. El banco de herramientas de FEDERICA gozará de todo lo necesario para monitorizar y aprovisionar a estos multi-dominios, además de garantizar seguridad y aislamiento a la infraestructura sobre la que esté construida.

3 Objetivos

Hay que diferenciar lo que son los objetivos del proyecto FEDERICA de los que son específicos de este PFC. Conociendo los generales es más fácil encajar en el puzzle que compone FEDERICA la pieza que se detalla en esta memoria.

3.1 Objetivos de FEDERICA

Los principales objetivos del proyecto FEDERICA se dividen en tres áreas: soporte, conexiones de red e investigación; se detallan a continuación:

1. Soporte
    a. Soporte a los experimentos de investigación de protocolos y nuevas arquitecturas de Internet.
        i. Crear una infraestructura europea tecnológicamente agnóstica, versátil y escalable; aislada de las redes de producción pero con la posibilidad de inter-operar con ellas.
        ii. Capaz de hospedar las aplicaciones, pruebas y hardware de los investigadores.
        iii. Adoptar el paradigma que dice que "la infraestructura es la red". El modelo tradicional de núcleo y extremos se sustituye por una red "casi homogénea", donde cada nodo no está definido en función de su posición, sino de sus quehaceres. Una topología lógica y sus funciones deben tener la posibilidad de cambiar fácilmente. La infraestructura se debe amoldar a cualquier cosa que necesite el usuario.
        iv. Uso simultáneo de la infraestructura por parte de los grupos de investigación (slicing).
    b. Proporcionar las facilidades necesarias para proyectos FIRE (Future Internet Research and Experimentation).

c. Aprovecharse del enorme potencial de las redes europeas que existen actualmente; especialmente de NREN y GEANT y sus tecnologías y servicios.

2. Conexiones de red
    a. Facilitar la comunicación entre los especialistas, sobretodo cuando se trata de resultados experimentales, diseminación de conocimientos y experiencias sobre sus respectivas redes virtuales.
    b. Proporcionar información preliminar y resultados para la siguiente generación de las redes NREN y GEANT2. Así como para una fase experimental GEANT3.
    c. Contribuir, a través de una serie de tests y resultados, con los cuerpos de estandarización, tales como IETF, ITU-T, OIF o IPsphere.
3. Investigación
    a. Entender y producir soluciones iniciales para la administración y el control de redes virtuales distribuidas y paralelas, las cuales pueden estar conectadas entre ellas o con Internet.
    b. Desarrollar experiencia en el manejo de una infraestructura virtual desde el punto de vista de una combinación de redes y sistemas. Actualmente, este tipo de administración y control se encuentra en su infancia. Se necesitan herramientas para monitorizar y dar servicio a dichas tecnologías.
    c. La infraestructura estará abierta a cualquier tipo de aplicación y/o test.
    d. Usar, testear y avanzar en todo aquello relacionado con las herramientas y servicios que están siendo desarrollados por GN2, NRENs y otros socios; en particular para servicios finales.
    e. Habilitar una implementación elegante de la nueva capa de servicios inter-dominio.

3.2 Objetivos del PFC

Este PFC se centra en las herramientas para crear *virtual slices* y satisfacer las necesidades de la infraestructura y del cliente en la capa de enlace de datos (nivel dos de la pila OSI). Es una de las funcionalidades de la aplicación completa de FEDERICA (tool-bench) que se encarga de suministrar y gestionar las redes virtualizadas (slices). Es importante que dicha gestión sea lo más centralizada posible. Es decir, que desde un mismo punto de control (NOC) se pueda estructurar toda la red. En este proyecto se configuran los switchs, los cuales pueden estar repartidos por toda la comunidad europea (aquellos países implicados en FEDERICA). Con lo cual, los objetivos específicos del PFC son:

1. Implementar una herramienta que configure los switchs que se encuentran dentro de la infraestructura de FEDERICA. Es un programa en Java que se comunica con los dispositivos vía SSH y NetConf y que es capaz de ejecutar las órdenes de configuración. Debe gestionar VLANs y parametrizar los puertos del switch. También debe mantener el estado de los dispositivos; para ello se hace una petición de su configuración al dispositivo parseando los resultados (devueltos por el switch en XML) y volcándolos en las clases Java correspondientes. Para llevar a cabo la labor, el objetivo se divido en varias partes:
    a. Estudio de los requisitos: Averiguar los comandos que se tienen que enviar a cada switch en el escenario de FEDERICA y agruparlos en acciones.
    b. Implementación del modelo del switch: El modelo tiene que capturar todo aquello que sea relevante para FEDERICA: configuraciones, inventario…
    c. Implementación de las acciones y los comandos para los switchs: basados en los requisitos obtenidos del primer sub-objetivo.
    d. Tests: validación del modelo (engine) mediante tests JUnit.

2. Implementar los Web Services (WS) que sean necesarios para el control de los aparatos desde cualquier lugar de Internet. Esto es, un WS que ofrece los servicios del punto anterior (crear VLANs, destruirlas, configurarlas, gestión de los puertos, etc.) sobre los dispositivos. Permite también, por lo tanto, añadir switchs a la infraestructura y gestionarlos. Un segundo Web Service se encarga de mantener los recursos que se ofrecen a los usuarios para sus respectivas *slices*. Los sub-objetivos quedan de la siguiente manera:
    a. Crear el esquema para los WS: implementación de los XMLs necesarios para la definición de las operaciones de cada uno de los Web Services (.wsdl, .xsd).
    b. Creación de los servicios: implementación de los servicios definidos en el .wsdl
    c. Implementación del recurso: esto es, aquella información de los switchs y los recursos cuyo estado debe mantenerse en los WS. Así como aquella que persistirá en la base de datos.
    d. Tests: validación de los WS mediante tests.

3. Implementar una interfaz gráfica (GUI) que gestione de manera intuitiva los recursos a nivel dos de la red. Unifica bajo el mismo entorno los dos apartados descritos anteriormente. Se trata de una aplicación muy simple programada en RCP (Rich Client Platform). Esta GUI es de carácter temporal ya que finalmente esta funcionalidad se incorporará en el *tool-bench* de FEDERICA. Está por definir si lo hará de una forma autónoma o si estará integrada en MANTICORE (aplicación que configura una red en la capa de red (nivel tres de la pila OSI)). Con lo cual tenemos:
    a. Implementación de la interfaz gráfica.
    b. Integrar la comunicación con el modelo (engine).
    c. Integrar la comunicación con los WS.
    d. Tests: validación del funcionamiento de la GUI.

Los objetivos anteriores están basados en la tendencia IaaS (Infrastructure As A Service) y están implementados siguiendo los principios de IaaS Framework. Los detalles sobre este marco de desarrollo se explican más adelante.

4 Metodología

Para la modelización de este proyecto se ha utilizado UML (Unified Modeling Language). A continuación se explican sus elementos básicos por tal de que el lector comprenda la documentación de este tipo que aparece a lo largo de la memoria.

En el mismo ámbito, la metodología de desarrollo que se ha seguido es la metodología en espiral; concepto que también se explica en esta sección.

4.1 UML

UML (Lenguaje de Modelado Unificado) es el lenguaje que se ha utilizado para modelar la estructura y las operaciones que se llevan a cabo en este proyecto. Se puede emplear para visualizar, especificar, construir y documentar cada uno de los componentes que confirman un sistema software. Ofrece un estándar para describir un plano del sistema (modelo). Los bloques que conforman este lenguaje son los siguientes:

- Elementos: Son los bloques de construcción básicos que componen los modelos UML. Hay cuatro tipos.
    - Estructurales: Son las partes estáticas del modelo. Entre ellos, están, entre otros, los casos de uso, las clases y las interficies.
    - De comportamiento: Son las partes dinámicas del modelo. Sirven para representar la evolución de los elementos UML. En este tipo se encuentran las máquinas de estado y las interacciones.
    - De agrupamiento: Organizan los elementos UML.
    - Notacionales: Comentarios para matizar ciertos conceptos relativos a los elementos del modelo.
- Relaciones: Su función es la de establecer modelos bien formados. Hay cuatro tipos:
    - Dependencias: Relación semántica entre dos elementos de manera que un cambio puede afectar a otro.

    - Asociación: Describe un conjunto de enlaces. Esto es, una conexión entre objetos. Normalmente representa el uso de un elemento por parte de otro.
    - Generalización: Es una relación donde el elemento especializado es substituido por el elemento generalizado. Es decir, el hijo comparte la estructura y función del padre.
    - Realización: Relación semántica donde se especifica un contrato que cierto elemento garantiza llevar a cabo. Se pueden dar entre interficies y sus clases y entre los casos de uso y sus colaboraciones.
- Diagramas: Representación gráfica de un conjunto de los elementos descritos anteriormente. Se puede decir que un diagrama es una proyección de un sistema. Los diagramas utilizados en este proyecto son:
    - Diagrama de clases: Muestran un conjunto de clases y sus relaciones. Son los más comunes en los sistemas orientados a objetos.
    - Diagramas de casos de uso: Muestra un conjunto de casos de uso y las relaciones de éstos con los actores. Un actor es un elemento externo que interactúa con el sistema. Éste puede ser una persona física u otro sistema.
    - Diagramas de secuencia: Es un tipo de diagrama de interacción. Consiste en un conjunto de objetos y sus relaciones incluyendo los mensajes que se envían entre ellos. Además queda representada la evolución temporal de la operación.

Existen ciertas reglas en el lenguaje UML necesarias para modelar una aplicación. Con lo cual, un sistema bien formado es aquel que semánticamente es consistente con sí mismo y con los modelos relacionados. Hay cuatro mecanismos para ejecutar esta tarea:

- Especificaciones: Proporciona una descripción textual de la sintaxis y semántica de los bloques de construcción.
- Adornos: Notación gráfica dirigida y única que representa visualmente los aspectos más importantes de los elementos.

- Divisiones comunes: Presenta la diferenciación entre clase (elemento abstracto) y objeto (manifestación concreta de la clase). UML modela ambos conceptos. Además, diferencia también entre interficie (declara un contrato) e implementación (manifestación concreta de una interficie).
- Mecanismos de extensibilidad: UML facilita que el programador extienda de manera controlada los elementos del sistema cuando sea necesario.

4.2 Metodología en espiral

El concepto de ingeniería del software propone un conjunto de técnicas y herramientas que se utilizan para poner en funcionamiento una aplicación que resuelva cierto problema.

Con este objetivo, la ingeniería del software parte del concepto de "ciclo de vida" de un programa informático que tiene en cuenta desde que la herramienta se empieza a gestar hasta que deja de ser utilizada. Todo ciclo de vida consta de cinco fases:

- Análisis de requisitos: Documento que proporciona una descripción general de la aplicación que se quiere desarrollar.
- Especificación: Define qué debe hacer la aplicación. Se definen los casos de uso y diagramas conceptuales para cubrir las funciones que realizará el sistema.
- Diseño: Se cambia el enfoque de las relaciones lógicas entre objetos al borrador físico del sistema. Es decir, se establece cómo se lleva a cabo lo definido en la especificación sin entrar en detalles. Se definen los diagramas de secuencia para ver cómo interactúan las diferentes partes.
- Implementación: Lo realizado en los dos puntos anteriores se convierte en código.
- Pruebas: Consiste en una serie de tests por tal de verificar el correcto funcionamiento del sistema.

La metodología en espiral consiste en alcanzar la aplicación definitiva mediante el desarrollo de aproximaciones sucesivas a la misma. En estas aproximaciones se reejecutan las fases de especificación, diseño, implementación y pruebas. Primero se

desarrolla el núcleo de la aplicación que será el prototipo inicial. Después de cada iteración (el número depende de la complejidad del sistema) aparece un prototipo hasta que finalmente se obtiene la aplicación deseada.

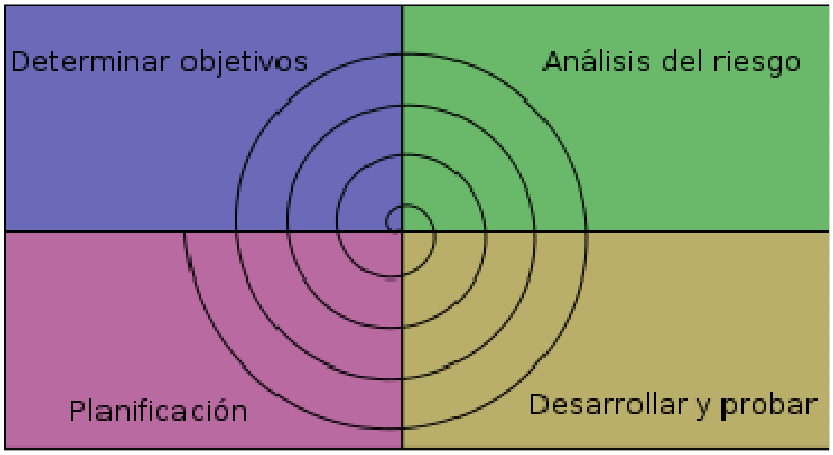


Figura 3: Metodología en espiral

En este proyecto se sigue la metodología en espiral en el módulo de desarrollo del engine. Esta parte, como se verá más adelante, es la que contiene la estructura de clases y operaciones que condicionarán a los otros módulos. Es por ello la fase más importante de analizar y especificar.

De todos modos, debido a que la estructura está fuertemente definida y a que las operaciones están muy bien especificadas de entrada, estas fases han sido simplificadas.

Las iteraciones consisten en, pasando por las cuatro fases que se ven en la figura anterior, lograr que funcione cierta operación; cuando ésta es estable, se vuelve a empezar. De este modo se alcanza finalmente el prototipo que aquí se presenta. Realmente no es el programa definitivo, ya que en etapas futuras de FEDERICA gozará de nuevas funcionalidades.

5 Entorno de programación

El sistema operativo utilizado para la elaboración del proyecto ha sido principalmente Windows XP, el cual es capaz de lidiar con todas las herramientas que han sido necesarias; si bien es cierto que el trabajo se podría haber llevado a cabo bajo Linux. Éste último SO, instalado en una máquina virtual (VM), se ha empleado para alojar los WSs y la base de datos.

El lenguaje de programación es Java. Es indiscutible el potencial que proporciona a sus usuarios y es el más utilizado actualmente. Además, el software fundamental para desplegar el proyecto (Globus toolkit) ofrece sus librerías para este lenguaje. Eclipse, con su condición de código abierto y sus funcionalidades adecuadas a las necesidades de la aplicación (compilación en tiempo real, compatibilidad con JUnit, integración con Ant, control de versiones con Subversion, entre otras) es la herramienta de desarrollo en Java que se ha creído más oportuna. Además, gracias a log4j, se facilita el debuggado del programa.

El sistema de base de datos es relacional. En ella se mantiene la persistencia de la información de los switchs que posee la infraestructura así como de los recursos publicados para los usuarios. El acceso se realiza mediante el mapeo de las clases Java utilizando Hibernate (integrado en Globus toolkit). El sistema de gestión de base de datos (SGBD) empleado es PostgreSQL.

La GUI está implementada mediante Rich Client Platform (el propio Eclipse, entre otras muchas herramientas, está programado con este software), el cual proporciona una manera sencilla de crear aplicaciones de este tipo. Es muy potente, por lo que es más que suficiente para satisfacer el propósito temporal y simplista de la interfaz gráfica en este proyecto.

Otras herramientas más sencillas, pero no por ello menos importantes son: PuTTy (para acceder remotamente al switch), pgAdmid (para acceder remotamente a la base de datos), WinSCP (para acceder remotamente al sistema de ficheros de la máquina virtual que contiene los WSs), entre otras que se irán descubriendo a lo largo de la memoria.

Una de las razones fundamentales de la elección de muchos de los instrumentos que componen el proyecto es que son software libre, cumpliendo así con una de las propiedades fundamentales de FEDERICA. La siguiente lista resume las herramientas principales que se han utilizado para llevar a cabo el proyecto:

- Lenguaje de programación: Java 1.6
- IDE: Eclipse 3.4 (Ganymede)
- Repositorio: SVN (Subversión, hospedado en un servidor)
- Herramientas de compilación: Apache Ant (integrado en Eclipse)
- Tests: JUnit (integrado en Eclipse)
- Herramienta de mapeo de los datos de la BD: Hibernate 3.0
- Web Services Framework: Globos Toolkit 4 (Java WS Core)
- Desarrollo de la GUI: Eclipse RCP (Rich Client Platform)

Por tal de clarificar y debido a la cantidad no trivial de herramientas utilizadas en este PFC, se adjunta al final del documento un listado (Anexo A) con las definiciones de todas ellas.

6 Desarrollo del proyecto

Es necesario para facilitar la comprensión del documento retomar momentáneamente parte de la visión global del proyecto. Una de las secciones importantes de FEDERICA es la que conforma la implementación del *tool-bench*, del cual este PFC forma parte. Dicho *tool-bench* se encargará de proporcionar las herramientas necesarias para la gestión de la infraestructura y de ofrecer soporte al usuario de la red virtualizada que le haya sido proporcionado; permitirá configurar y personalizar su *slice*. Donde un *slice* es una red que ha sido proporcionada al investigador y que abastece todas sus necesidades. Ésta funciona sobre la infraestructura física de FEDERICA, compartiéndola con el resto de *slices*. Estas herramientas permitirán crear las topologías específicas que hayan demandado los usuarios, incluyendo direccionamiento y enrutamiento. Además deberá asegurarse de que la red se mantiene operativa; para ello se necesitan mecanismos de administración y de monitorización. Por lo tanto, el panel de control del *tool-bench* interactuará con todas las capas, niveles, y dominios para que el servicio sea de calidad. Finalmente, se proporcionará soporte para mantener información redundante y/o añadir recursos alternativos por tal de incrementar la fiabilidad de la infraestructura.

Desde el punto de vista del administrador de la infraestructura, es decir, del NOC, las principales funcionalidades del *tool-bench* quedan resumidas en la siguiente figura:

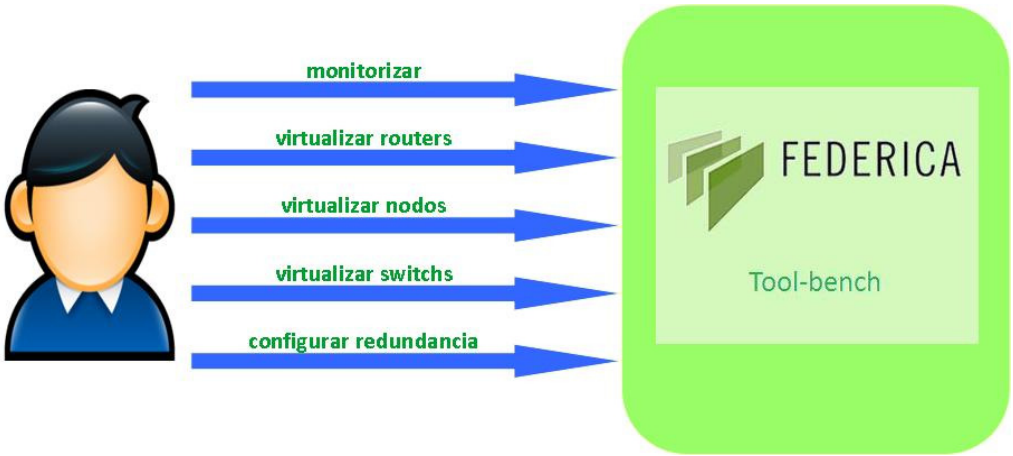


Figura 4: Operaciones básicas sobre el *tool-bench* de FEDERICA

En este diagrama se muestran las operaciones básicas y muy generalizadas que podrá realizar el administrador de la infraestructura (Network Operation Center).

- Monitorización: Para asegurar el correcto funcionamiento de lo que está sucediendo en la red.
- Redundancia: Garantiza fiabilidad a los usuarios.
- Virtualización: Las tres fechas de virtualización hacen referencia a la creación de los *slices*. Se tiene que configurar todo lo necesario en los diferentes elementos de la infraestructura: routers, nodos (que son máquinas virtuales con algún sistema operativo instalado) y los switchs.

Para entender mejor el proyecto, el lector puede imaginar que lo que se explica de ahora en adelante es un botón incrustado en el *tool-bench*; el cual lanzará la GUI correspondiente sobre la que se configuran los switchs mediante los procedimientos que se describen a lo largo de la memoria.

Con el contexto bien definido, se procede a detallar la parte correspondiente al PFC: configuración de la red virtual a nivel dos (capa de enlace de datos).

6.1 Visión completa del proyecto

A partir de este punto, se procede a describir el funcionamiento global de la aplicación por tal de entenderla y poder explicar en los siguientes apartados la elaboración detallada de cada uno de los elementos. La siguiente figura muestra el esquema completo de la aplicación.

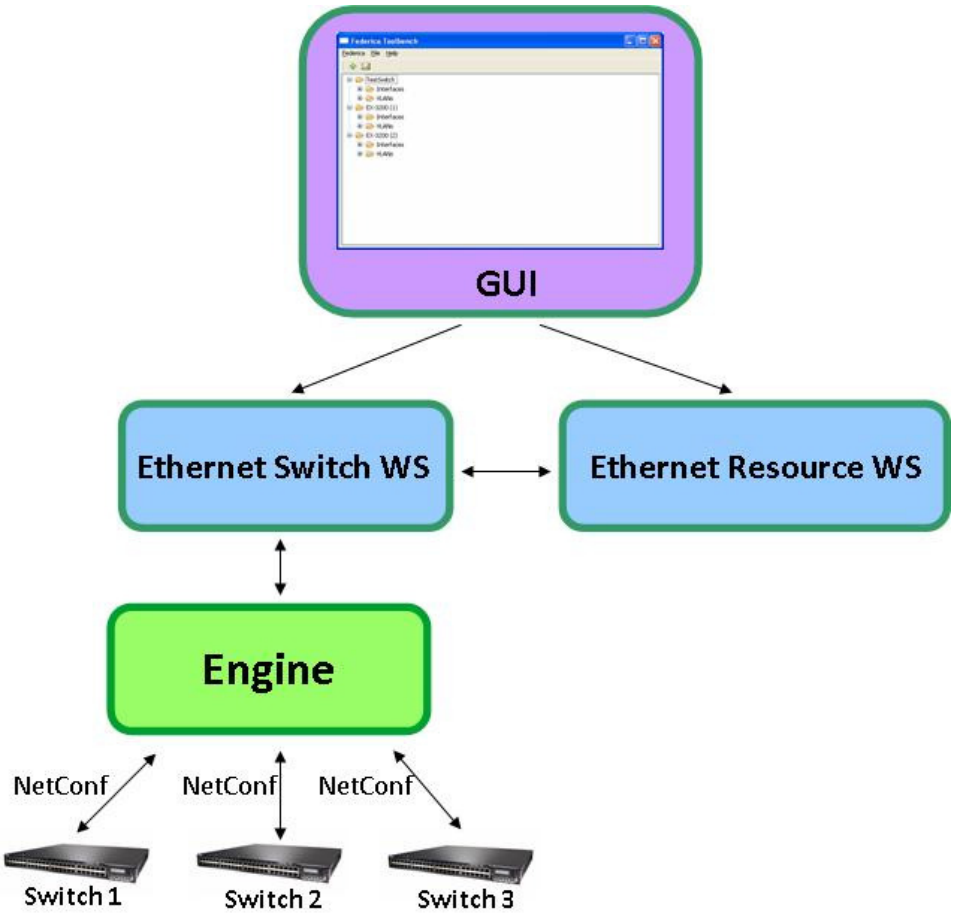


Figura 5: Esquema del sistema completo de la aplicación

El usuario, es decir, el administrador o NOC, operará sobre la interfaz gráfica proporcionada. Desde ella, será capaz de gestionar todos los aspectos necesarios a nivel de enlace acorde a los objetivos de FEDERICA. Todos ellos quedan cubiertos en los casos de uso que se presentan en el punto siguiente. Las peticiones que actúan sobre el switch serán enviadas al Ethernet Switch Web Service, el cual administra los dispositivos de la infraestructura remotamente. Desde él se pueden crear switchs, modificarlos y gestionarlos. Hay una instancia de este WS por cada aparato físico que contenga la infraestructura.

Hay un nivel intermedio entre el Web Service y el dispositivo: el Engine. Éste se encarga de mantener la configuración y de realizar las operaciones sobre el switch.

El otro Web Service que aparece en la figura (Ethernet Resource) es el que se encarga de los servicios relacionados con los recursos que se proporcionan a los usuarios de la infraestructura de FEDERICA. Si se quiere ofrecer, por ejemplo, cierta interfaz a cierto usuario (*slice* de usuario), se crea un Ethernet Resource (proceso de

virtualización) que posteriormente podrá ser asignado a un grupo de investigación determinado.

Todos los módulos aquí descritos se verán con mayor detalle durante el transcurso del documento.

6.2 Casos de uso y diagramas de secuencia genéricos

Debido al carácter temporal de la GUI, los casos de uso que se especifican a continuación no son todos los que permiten realizar tanto los WS como el engine. Únicamente abarca aquellos que son necesarios para una configuración básica de los switchs de la infraestructura por tal de hacer tests y demostraciones. El siguiente diagrama muestra todos los casos de uso que actualmente son soportados por la implementación de los dos módulos más importantes (WS y Engine). Los que están marcados en morado son aquellos que además pueden ser lanzados a través de la interfaz gráfica.

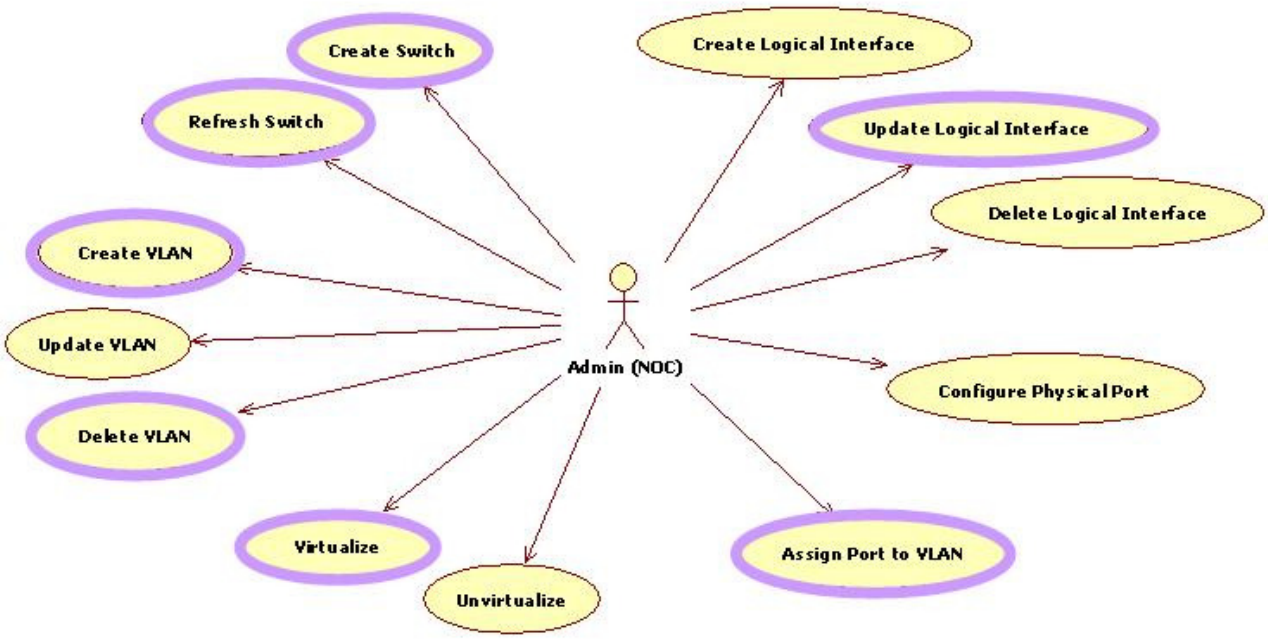


Diagrama UML 1: Casos de uso de la aplicación

La descripción de los casos de uso es bastante genérica en su composición por tal de entender los aspectos globales. Se adjuntan además algunos diagramas de

secuencia básicos de algunos de aquellos que están integrados en la GUI; esto es, diagramas en los que se presenta cómo interactúan las tres piezas de la aplicación (GUI, WS y Engine). En el apartado en el que se describen las tareas se verán algunos de los casos de uso con las interacciones, mucho más detalladas, entre y dentro de los módulos.

Crear Switch:

- Descripción: Añadir a la aplicación un switch físico. Se obtiene toda la información necesaria para identificarlo y establecer un canal de comunicación.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona la opción de añadir/crear un switch. Se abre una ventana donde éste debe introducir los datos del dispositivo: nombre, fabricante, modelo, usuario, contraseña, dirección IP, transporte y protocolo; pulsa siguiente.
2. La interfaz gráfica se comunica con el WebService enviándole estos datos. Éste último crea un modelo con la información y pide al engine la configuración del switch.
3. El engine se comunica con el switch mediante la información proporcionada por el administrador (usuario, contraseña, dirección IP, protocolo y transporte) y envía la configuración del dispositivo al WebService.
4. Se guarda aquella información necesaria en la base de datos.
5. Se actualiza la interfaz gráfica por tal de mostrar el switch recientemente añadido junto a la configuración de sus interfaces y VLANs.

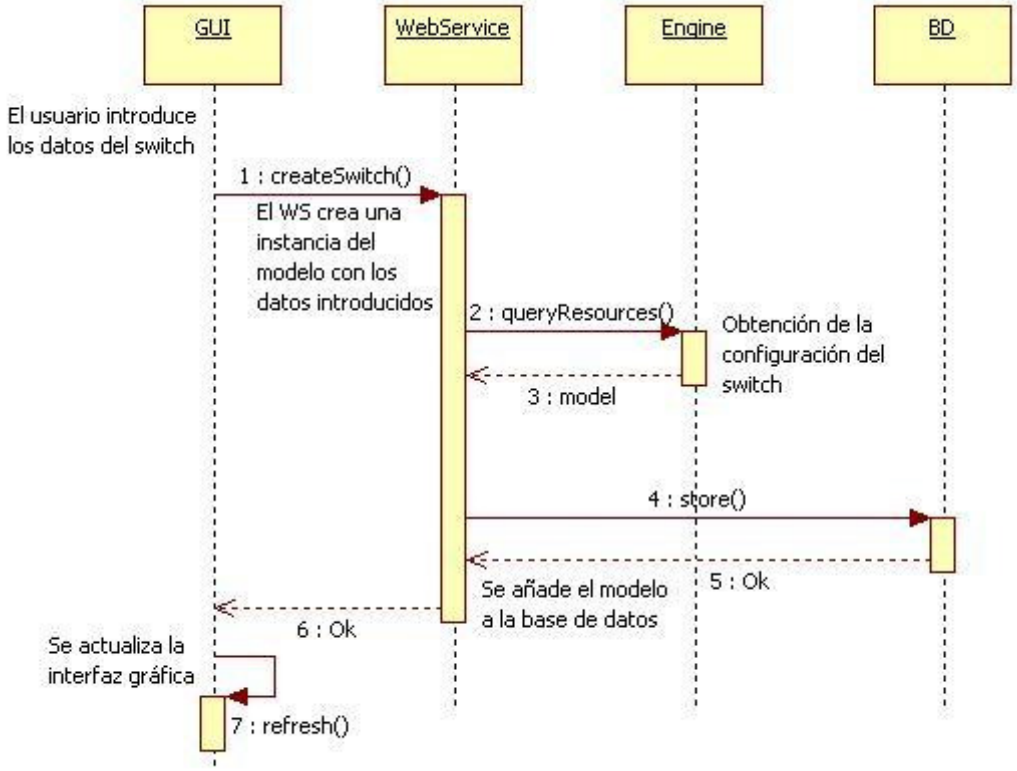


Diagrama UML 2: De secuencia básico - Crear switch

Actualizar Switch:

- Descripción: Fuerza la actualización de la configuración de un switch físico. Es decir, se comunica con el dispositivo y renueva la información en todo el sistema.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el switch que desea actualizar.
2. La interfaz gráfica se comunica con el WebService enviándole la petición. Éste último se comunica con el modelo (engine) y le pide la configuración actual del switch.
3. El engine se comunica con el switch y obtiene la configuración del dispositivo.
4. Se actualiza la interfaz gráfica por tal de mostrar el switch recientemente actualizado junto con la nueva configuración de sus interfaces y VLANs.

Configurar puerto físico:

- Descripción: Actualizar un puerto físico de alguno de los switchs de los que se dispone.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el puerto físico del switch que desee y elige la opción de modificarlo. Éste debe introducir los nuevos datos del puerto: descripción, link-speed, link-mode…
2. El WebService del switch interpreta estos datos, busca el modelo sobre el cual se desea actuar y envía la operación al engine.
3. El engine se comunica con el switch actualizando el puerto con los datos proporcionados por el administrador.
4. El engine actualiza el modelo.

Crear puerto lógico:

- Descripción: Crea un puerto lógico en una interfaz física de alguno de los switchs de los que se dispone.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el puerto físico del switch sobre el que desee crear la interfaz lógica (unit) y la configura.
2. El WebService del switch interpreta estos datos, busca el modelo sobre el cual se desea operar y envía la operación al engine.
3. El engine se comunica con el switch creando la interfaz lógica proporcionada por el administrador.
4. El engine actualiza el modelo.

Actualizar puerto lógico:

- Descripción: Actualizar un puerto lógico de alguno de los switchs de los que se dispone.
- Actores: Administrador.

- Flujo de eventos:

1. El administrador selecciona el puerto lógico del switch que desee y elige la opción de modificarlo. Se abre una ventana donde éste debe introducir los nuevos datos del puerto: trunk/no trunk; pulsa siguiente.
2. La interfaz gráfica se comunica con el WebService enviándole estos datos. Éste último busca el modelo sobre el cual se desea operar y envía la operación al engine.
3. El engine se comunica con el switch actualizando la interfaz lógica con los datos proporcionados por el administrador.
4. El engine actualiza el modelo.
5. Se actualiza la interfaz gráfica por tal de mostrar los cambios realizados sobre la interfaz.

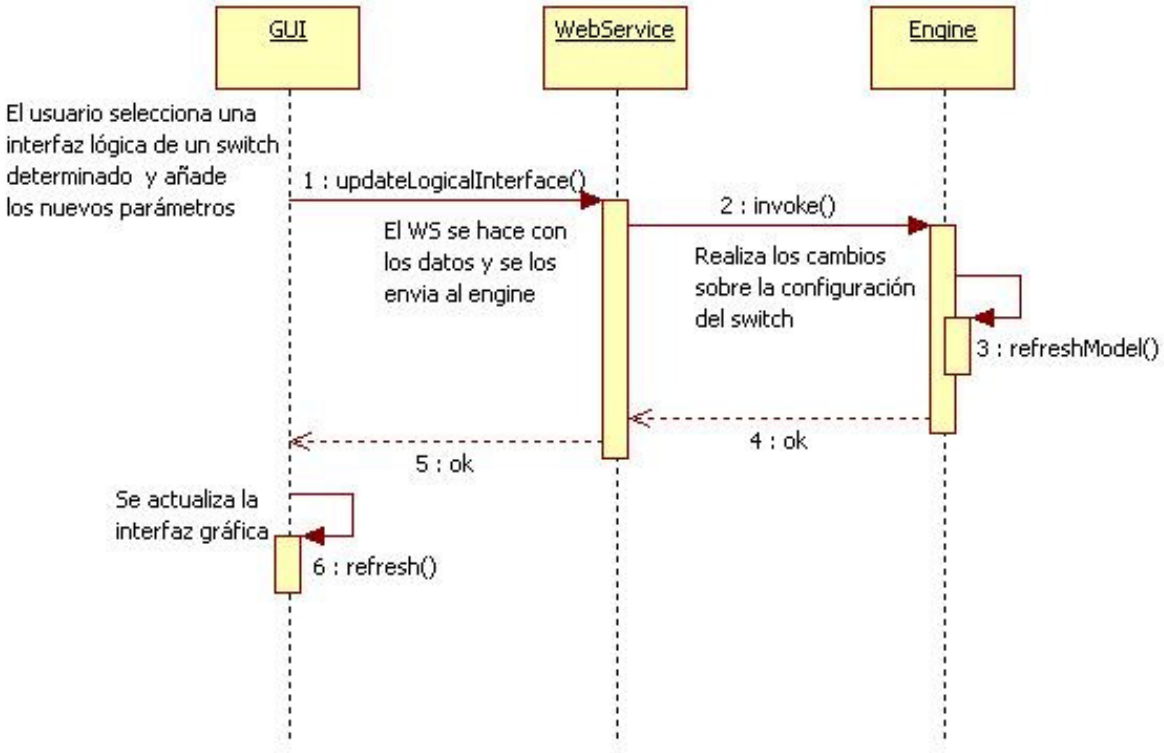


Diagrama UML 3: De secuencia básico - Actualizar Puerto Lógico

Eliminar puerto lógico:

- Descripción: Destruye un puerto lógico de alguno de los switchs de los que se dispone.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el puerto lógico del switch que desee y procede a eliminarlo.
2. El WebService del switch interpreta estos datos, busca el modelo sobre el cual se desea operar y envía la operación al engine.
3. El engine se comunica con el switch destruyendo la interfaz lógica proporcionada por el administrador.
4. El engine actualiza el modelo.

Crear VLAN:

- Descripción: Se crea una VLAN sobre un switch determinado con la información necesaria.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el switch sobre el que desee crear la VLAN. Se abre una ventana donde éste debe introducir los datos de la VLAN: nombre e identificador; pulsa siguiente.
2. La interfaz gráfica se comunica con el WebService enviándole estos datos. Éste último obtiene el modelo sobre el cual se desea crear la VLAN y envía la operación al engine.
3. El engine se comunica con el switch creando la virtual LAN a partir de los datos proporcionados por el administrador.
4. El engine actualiza el modelo.
5. Se actualiza la interfaz gráfica por tal de mostrar los cambios realizados sobre el switch.

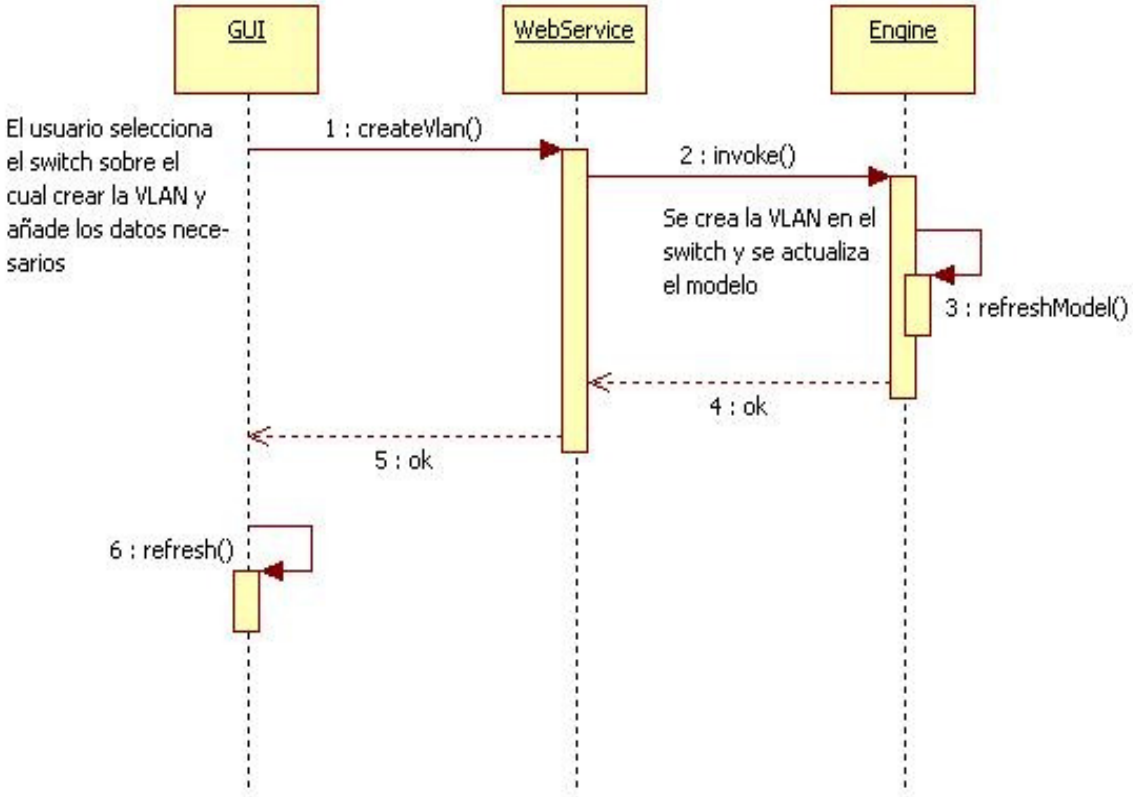


Diagrama UML 4: De secuencia básico - Crear VLAN

Actualizar VLAN:

- Descripción: Se actualiza una VLAN sobre un switch determinado con la información necesaria.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona la VLAN que desee modificar de cierto switch e introduce los nuevos datos de la VLAN: nombre, identificador, descripción; pulsa siguiente.
2. El WebService del switch interpreta estos datos y obtiene el modelo con la VLAN que se desea actualizar y envía la operación al engine.
3. El engine se comunica con el switch actualizando la virtual LAN a partir de los datos proporcionados por el administrador.
4. El engine actualiza el modelo.

Borrar VLAN:

- Descripción: Se elimina una VLAN de un switch determinado.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona un switch dentro del cual esté la VLAN que desea eliminar y selecciona la opción correspondiente.
2. El administrador selecciona la VLAN que desea borrar; pulsa siguiente.
3. La interfaz gráfica se comunica con el WebService enviándole el mensaje de borrado. Éste último obtiene el modelo sobre el cual se desea destruir la VLAN y envía la operación al engine.
4. El engine se comunica con el switch destruyendo la LAN virtual a partir de los datos proporcionados por el administrador.
5. El engine actualiza el modelo.
6. Se actualiza la interfaz gráfica por tal de mostrar los cambios realizados sobre el switch.

Asignar puerto a VLAN:

- Descripción: Se añade un puerto lógico (unit) a una VLAN.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador busca el dispositivo y sobre éste clica en la opción de asignar puerto a VLAN.
2. El administrador selecciona el puerto (lógico) y la VLAN que desee asociar; pulsa siguiente.
3. La interfaz gráfica se comunica con el WebService enviándole el mensaje de añadir la interfaz a la VLAN dada. Éste último obtiene el modelo sobre el cual se desea efectuar el cambio y envía la operación al engine.
4. El engine se comunica con el switch asignando a la virtual LAN el puerto lógico seleccionado por el administrador.
5. El engine actualiza el modelo.
6. Se actualiza la interfaz gráfica por tal de mostrar los cambios realizados sobre el switch.

Virtualizar puerto:

- Descripción: Se virtualiza un puerto de un switch determinado. Es decir, se crea un recurso con una configuración (usuario y VLAN que se le asigna) determinada que podrá ser utilizado por algún usuario final de FEDERICA.
- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona un switch dentro del cual esté configurado el puerto lógico que desea virtualizar y selecciona la opción homónima.
2. Se selecciona la interfaz que se desea y se introducen los datos para la virtualización (usuario y VLAN).
3. La interfaz gráfica se comunica con el WebService enviándole estos datos. Éste último crea un recurso con la configuración del puerto.
4. Se actualiza la interfaz gráfica por tal de mostrar los cambios realizados sobre el switch.

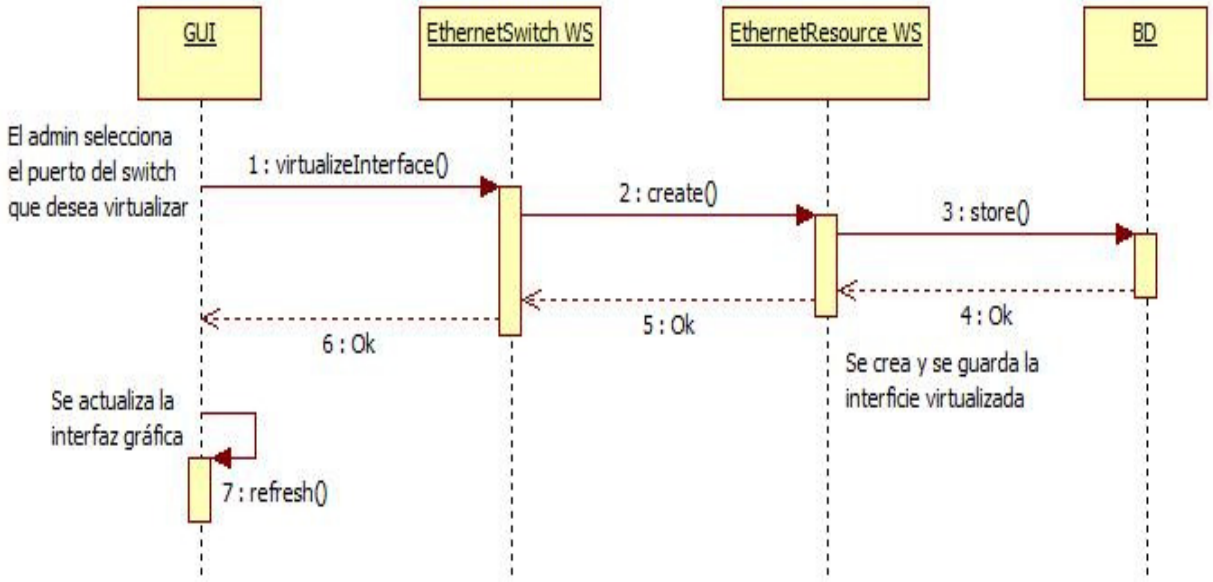


Diagrama UML 5: De secuencia básico - : Virtualizar puerto lógico

Desvirtualizar recurso:

- Descripción: Destruye un recurso de lo que se han publicado para los usuarios finales de FEDERICA (puertos virtualizados).

- Actores: Administrador.
- Flujo de eventos:

1. El administrador selecciona el recurso que desee y procede a eliminarlo.
2. El WebService Resource interpreta estos datos, busca el recurso y lo destruye.

El proyecto tiene tres tareas de elaboración bien diferenciadas que coinciden con los objetivos planteados en el punto anterior correspondiente. Las dos primeras (Implementación del modelo y de los WS) están basadas en IaaS Framework. La interfaz gráfica está desligada de este conjunto de recursos y es de carácter temporal. Todo ello se describe en el siguiente apartado. Además, se incluye el estudio previo realizado sobre el switch y sobre el cual se basó la elección de las diferentes herramientas.

6.3 Descripción de las tareas

6.3.1 Estudios

Para implementar las tareas, se ha realizado un estudio previo sobe los dispositivos que se deben configurar. En FEDERICA los switchs utilizados son Juniper, que a su vez es uno de los *partners* del proyecto. Se trata del modelo EX-3200. El sistema operativo que llevan integrado los aparatos de este fabricante se denomina JUNOS y sobre él se realizan los estudios aquí presentados.

Figura 6: Juniper EX-3200

Además, paralelamente, se han llevado a cabo pruebas sobre la aplicación de Q-in-Q y QoS/CoS que, debido a que actualmente no se han implementado, quedan plasmadas en el anexo B de este documento.

A continuación se ven los pasos a través de los cuales se ha llegado a las soluciones que se han creído más apropiadas así como una pequeña explicación del funcionamiento del switch, las interfacez, las VLANs etc. necesaria para ubicar al lector en el entorno apropiado.

6.3.1.1 Arquitectura JUNOS

Junos está basada en dos planos de gestión: En el plano de control (Control Plane) se maneja cualquier configuración que se introduzca, ya sea vía CLI, Netconf, etc. Este plano controla al plano de transferencia (Forwarding Plane), donde los paquetes se gestionan y envían/reciben de manera desligada al plano de control. La siguiente figura muestra un diagrama de la arquitectura de este SO.

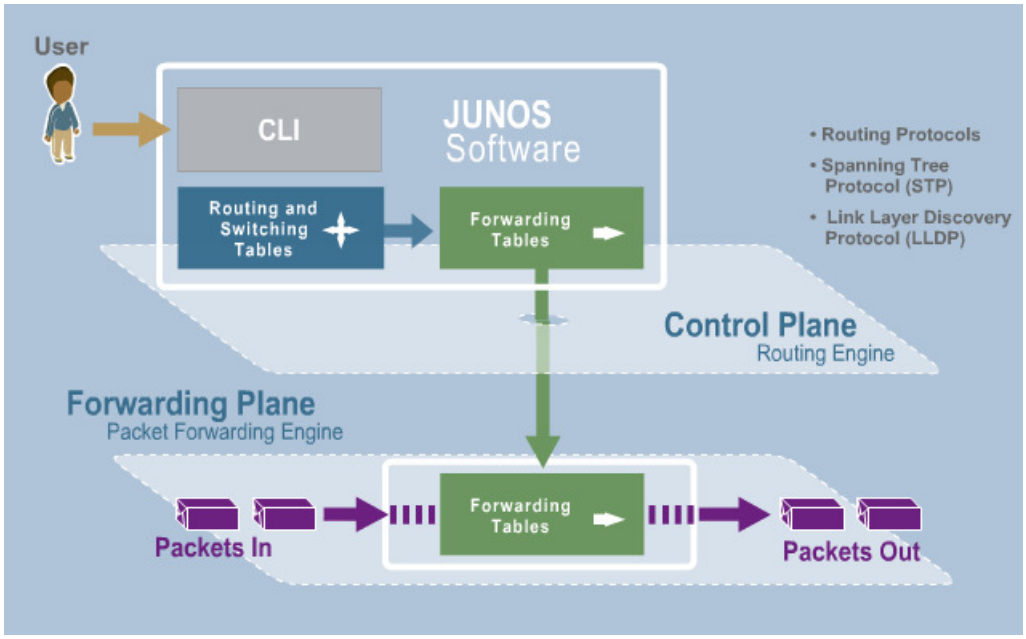


Figura 7: Arquitectura de JUNOS

Esta modularización de los planos es una de sus ventajas, ya que si el plano de control está ocupado realizando, por ejemplo, una actualización del sistema, el plano de transferencia sigue funcionando independientemente.

Junos ofrece un gran repertorio de posibilidades para configurar sus dispositivos:

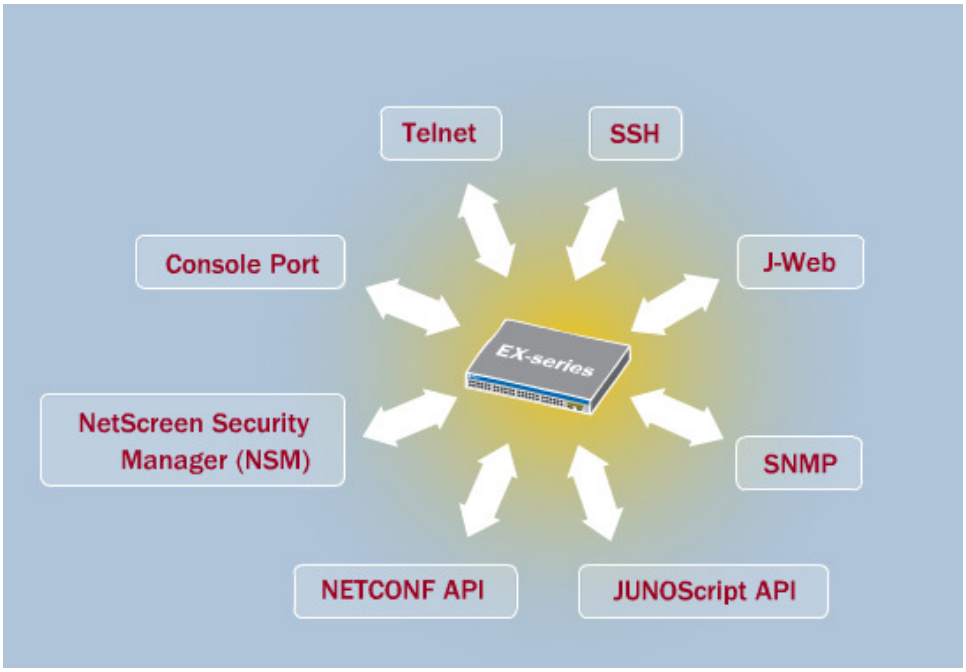


Figura 8: Métodos de configuración sobre JUNOS software

Todos ellos son muy diversos. Tanto en el acceso mediante el puerto de control (Console port), como Telnet y SSH, la configuración se realiza mediante CLI (Command User Interface). En cambio, J-Web es una interfaz gráfica accesible vía Web que proporciona una manera muy sencilla de tratar con los elementos del dispositivo en todos sus aspectos (gestión, configuración y monitorización entre otras cosas).

Los métodos de configuración que más se adaptan a los requisitos del proyecto son CLI y NetConf debido a su capacidad para ser integrados en el resto de módulos del sistema. Sobre ellos se centra el estudio desarrollado en los inicios del proyecto.

Otra de las propiedades más destacadas de JUNOS es que todos los cambios que se van realizando en la configuración de los dispositivos que controla, quedan guardados en un fichero “candidato”. Éste mantiene la configuración que se está desarrollando sin tocar la que realmente está funcionando en el switch, llamada “configuración activa”. Esto es realmente útil para reducir los errores en tiempo real durante los cambios de configuración.

Para activar la configuración candidata, se debe introducir el comando *commit.* El sistema detecta si la configuración es correcta o no y no efectuará los cambios hasta que los errores sean solucionados.

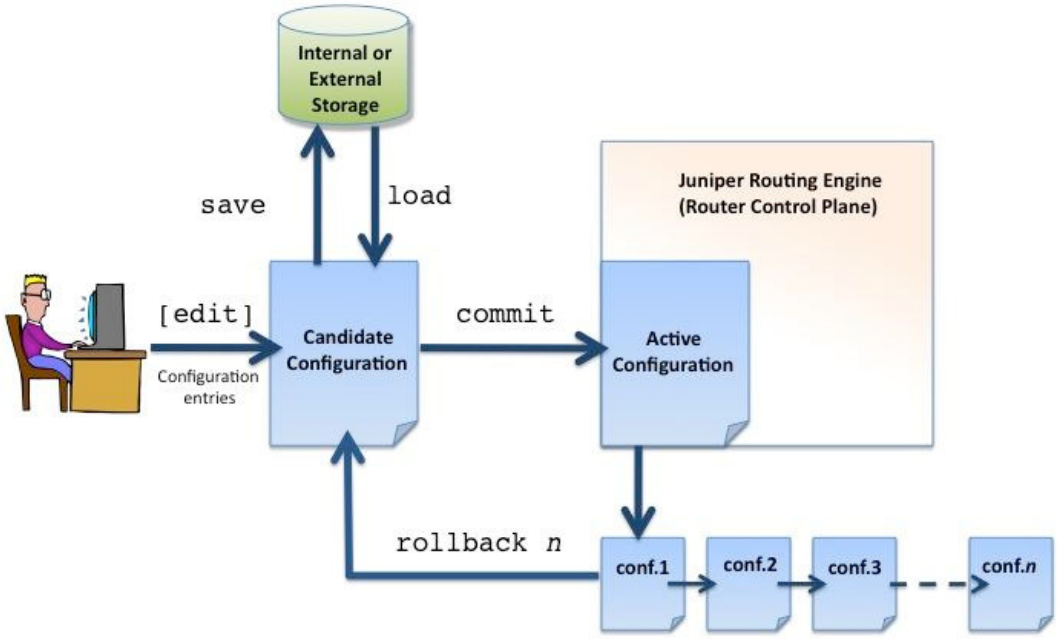


Figura 9: Sistema de cambios en la configuración de JUNOS

Como se muestra en la figura, cada vez que se realiza un *commit* se guardan las configuraciones antiguas en una serie de ficheros por tal de poder recuperar estados anteriores mediante la operación de *rollback*. Al hacerlo, la configuración se carga como candidata y no será aplicada hasta que se efectúe de nuevo un *commit*.

6.3.1.2 Interfaces

Existen dos tipos de interfaces en los switchs EX-3200:

- Interfaces de red: transportan tráfico de red. Éstas pueden ser de varios tipos.
    - Interfaz LAN *access*: El switch consta de 24 puertos de red que se pueden utilizar para conectar PCs, servidores, portátiles, etc. El modo acceso es el que viene configurado por defecto.

- Interfaz LAN *trunk*: Un switch de acceso conectado a uno de distribución o a un router debe ser explícitamente configurado como *trunk*. Un puerto configurado de esta manera puede transportar múltiples VLANs.
- Interfaz *Power over Ethernet* (PoE): Estos puertos aportan energía a los dispositivos conectados. Es útil para conectar teléfonos VoIP, puntos de acceso inalámbricos, cámaras de vídeo etc. A su vez, permite conectarlos con la red.
- Interfaz Ethernet agregada: Interfaces que se pueden agrupar al nivel físico para formar un único *link*. Son útiles para el balance de tráfico e incrementar del ancho de banda.

- Interfaces especiales: se utilizan para acceder al dispositivo. Éstas pueden ser de varios tipos.
  - Interfaz de administración: permite conectarse al switch de manera independiente al resto de interfaces. Para ello, se debe configurar el puerto lógico de la interfaz me0 (me0.0) con una dirección IP válida. Se puede acceder mediante Telnet o SSH así como SNMP.
  - Interfaz de control: Se trata de un puerto serie por el cual se puede configurar el dispositivo mediante consola. No necesita de direcciones de ningún tipo.
  - *Loopback*: Interfaz software que proporciona una manera estable y consistente de realizar pruebas.

6.3.1.3 VLANs

Una Virtual LAN es un método para crear redes lógicamente independientes dentro de la misma red física. Es un concepto básico para FEDERICA ya que una de sus características fundamentales es la de ofrecer a los diferentes usuarios finales un total aislamiento con el resto, a no ser que deseen lo contrario. Las VLANs que maneja el switch EX-3200 están implementadas bajo el protocolo IEEE 802.1Q. Este protocolo etiqueta los paquetes Ethernet usando un campo interno del mismo que se reserva para tal fin. Por lo tanto, 802.1Q añade un campo de 32 bits entre la dirección MAC de

origen y el indicador de longitud y *EtherType* del paquete, tal y como muestra el segundo *frame* de la figura:

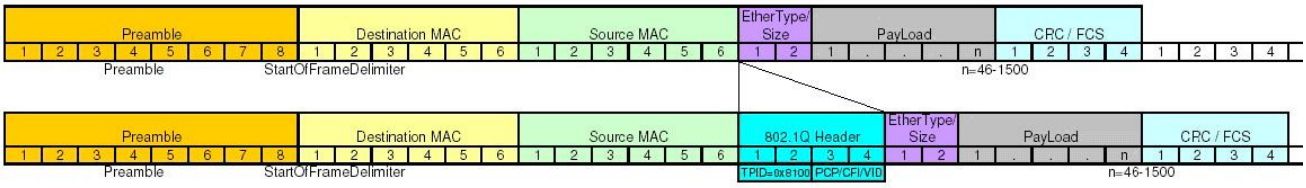


Figura 10: Inserción de una etiqueta 802.1Q en un paquete Ethernet

De esta manera se consigue que el tráfico que transporta una VLAN no sea visible por el resto de virtual LANs configuradas.

6.3.1.4 Command Line Interface (CLI)

La consola de comandos de Juniper funciona sobre un sistema operativo basado en el *kernel* de UNIX y proporciona también herramientas de este sistema. CLI tiene dos modos: El modo de operaciones se utiliza para monitorizar los diferentes componentes del dispositivo. El modo de configuración se ocupa de definir, precisamente, su configuración: VLANs, puertos, unidades lógicas, etc. Su principal ventaja es que las instrucciones son fáciles de enviar.

Algunas de las instrucciones que se pueden llevar a cabo y que están directamente relacionadas con las que permite realizar la aplicación son las siguientes:

Crear una VLAN:

```
user@switch# set vlans vlanTest vlan-id 111
user@switch# set vlans vlanTest description “Test vlan”
```

Asignar un puerto a una VLAN:

```
user@switch# set vlans vlanTest interfaces ge-0/0/0.0
```

Creación de un puerto lógico (unit) y configuración del mismo como *trunk*:

```
user@switch# set interfaces ge-0/0/0 unit 11
user@switch# set interfaces ge-0/0/0 unit 11 family ethernet-
  switching port-mode trunk
user@switch# set interfaces ge-0/0/0 unit 11 description
  "Testing trunk port mode"
```

Destrucción de una VLAN:

```
user@switch# delete vlans vlanTest
```

Estos son algunos de los comandos básicos que se pueden ejecutar sobre puertos y LANs virtuales. Hay muchos más con los que se ha probado un conjunto de configuraciones por tal de exprimir su potencial al máximo.

6.3.1.5 NetConf

Es un protocolo basado en XML cuya función es proporcionar mecanismos para instalar, manipular y eliminar la configuración de dispositivos de red. Un cliente puede comunicarse con el servidor NetConf que se encuentra en el propio dispositivo. Además, todas las operaciones disponibles vía CLI también lo están mediante este protocolo. Goza de tres ventajas fundamentales:

- La página Web de Juniper proporciona una amplia documentación sobre su funcionamiento y comandos disponibles.
- Facilidad para entender el contenido gracias a la estructura de etiquetas propia del lenguaje XML.
- Capacidad de obtener toda la configuración del dispositivo y parsearla de manera mucho más sencilla que mediante CLI.

Para poner algunos ejemplos de cómo son los mensajes bajo este protocolo se muestran los siguientes fragmentos de código, análogos a los mostrados anteriormente en CLI:

Crear una VLAN y asignarle un puerto determinado:

```
<configuration>
   <vlans>
      <vlan>
         <name>vlanTest</name>
         <tag>111</tag>
         <description>Test vlan</description>
         <interface>
            <name>ge-0/0/0.0</name>
         </interface>
      </vlan>
   </vlans>
</configuration>
```

Creación de un puerto lógico (unit) y configuración del mismo como *trunk*:

```
<configuration>
   <interfaces>
      <name>ge-0/0/0</name>
         <unit>
            <name>11</name>
            <desciption>Testing trunk port mode</desciption>
               <family>
                  <ethernet-switching>
                     <port-mode>trunk</port-mode>
                  </ethernet-switching>
               </family>
         </unit>
   </interfaces>
</configuration>
```

Destrucción de una VLAN:

```
<configuration>
   <vlans>
      <vlan operation = delete>
         <name>testVlan</name>
      </vlan>
   </vlans>
</configuration>
```

Después del análisis de ventajas y desventajas del uso de CLI o NetConf para manejar los switchs, éste último ha sido seleccionado para ser utilizado en la implementación de la aplicación. El factor decisivo para hacerlo es su capacidad de mapeo. Se puede decir que NetConf está más orientado a objetos que CLI. Una vez decidido esto, el transporte a aplicar es inmediato ya que es el único que soporta NetConf: SSH.

6.3.2 IaaS Framework

Como se ha ido mencionando en puntos anteriores, el desarrollo de este PFC está basado en este conjunto de herramientas y librerías. IaaS Framework es un entorno de desarrollo ágil para la creación de recursos Infrastructure as a Service (IaaS). Simplifica la elaboración de aplicaciones Web dinámicas ofreciendo capacidad de agregación en Java y Groovy. Está optimizado para crear representaciones de dispositivos hardware mediante servicios de Web Service o REST. El framework se divide en cuatro módulos independientes:

- Engine framework para el control de dispositivos.
- Framework de recursos. Mantienen el estado de ciertos objetos.
- Soluciones para ofrecer servicios remotos. Son capaces de acceder y gestionar los recursos.
- Frameworks de persistencia. Es capaz de proporcionar persistencia a los recursos.

De entre todos estos módulos, existen varios casos de uso. Por ejemplo, un servicio que realice operaciones remotas sin persistencia sobre cierto dispositivo. En este PFC el caso de uso sería el de una serie de servicios para operar sobre un conjunto de dispositivos (switchs) cuyos recursos tendrán una parte persistente y otra transitoria.

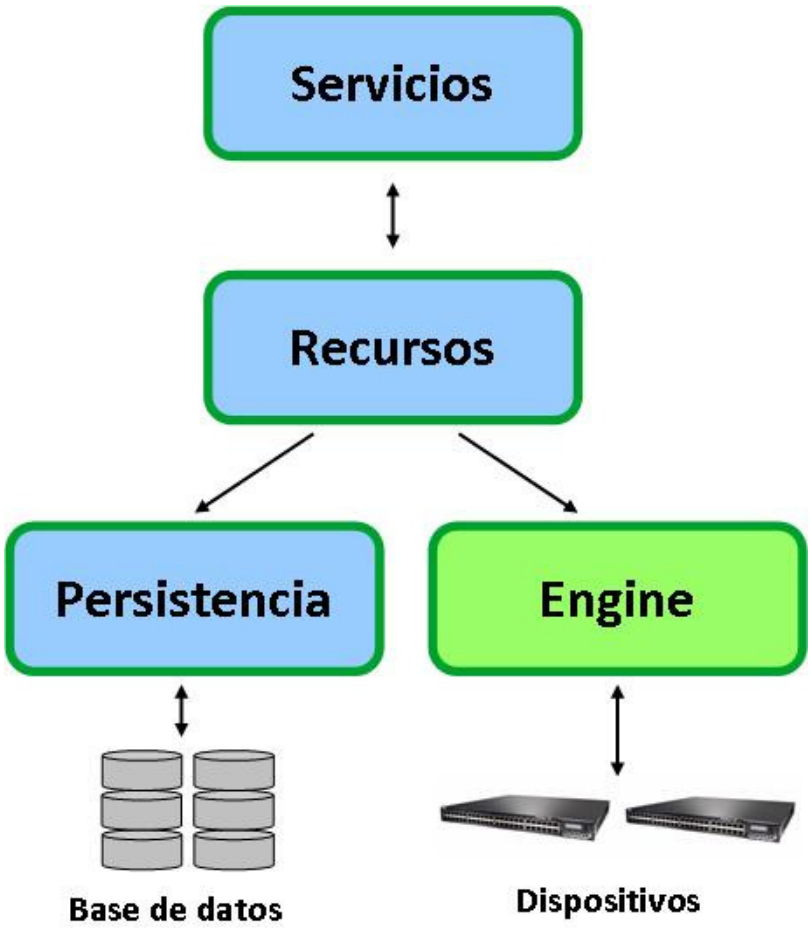


Figura 11: Caso de uso de IaaS Framework empleado para este PFC

En este proyecto se plasman conjuntamente los módulos de servicios, recursos y persistencia en uno solo ya que están fuertemente relacionados en su desarrollo. Con lo cual, tenemos dos módulos: Engine y Web Services. A continuación se ven cada uno de ellos en detalle.

6.3.3 Desarrollo del Engine

Es una de los módulos de IaaS Framework. Se trata de una serie de herramientas para crear un modelo que represente un dispositivo físico. Se puede decir que es una especie de *driver* sobre el que se accederá vía WS, por ejemplo. Los engines deben ser fáciles de crear y mantener y deben estar sincronizados con sus aparatos. El núcleo del engine tiene, entre otras cosas, mecanismos de transporte (SSH, TCP, Telnet…), implementación de protocolos (CLI, NetConf…), acciones y comandos; todo ello se

detallará a continuación. Sobre estos engines está basada la implementación específica para los switchs que aquí se presenta.

El engine tiene dos propósitos: El primero es proporcionar una descripción, ya sea en RDF o en XML, basada en el estado del dispositivo. El segundo es facilitar las acciones que serán utilizadas por los programas correspondientes para actuar sobre el aparato. Es importante también que la modelización sea lo mas genérica posible. Es decir, que la implementación sea transparente al modelo y fabricante del aparato en cuestión en sus niveles más altos. En el caso del switch, todos tienen más o menos las mismas funcionalidades independientemente de si se trata de un Cisco, Nortel o un Juniper.

6.3.3.1 Arquitectura

Se trata de una arquitectura modular. A continuación se muestran los componentes básicos del engine ya adaptados al propósito del proyecto:

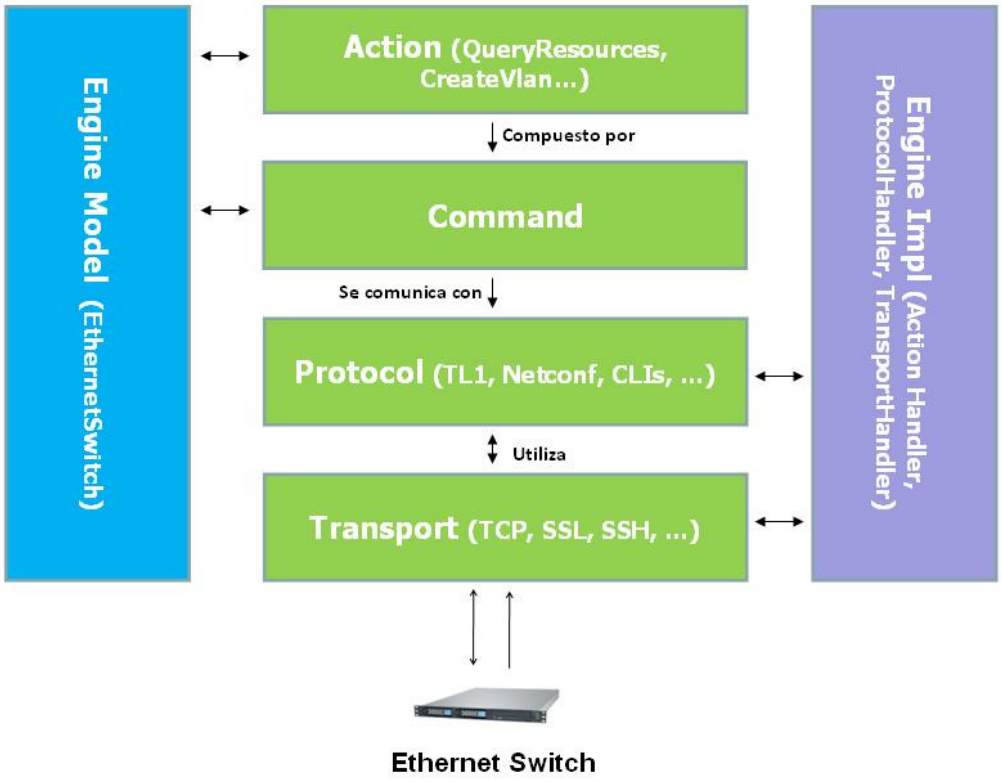


Figura 12: Arquitectura/Componentes básicos del engine en este proyecto

Transporte:

Es el componente sobre el cual obtenemos la conexión física o de red con el dispositivo en sí. Su función es transmitir la información pertinente entre el protocolo (definido a continuación) y el dispositivo. Además, debe tener el nivel de abstracción necesario para ser utilizado por los diferentes protocolos que puedan funcionar sobre él. En este caso, el transporte utilizado es SSH y su implementación debe ser capaz de soportar tanto CLI como NetConf y todos aquellos protocolos que funcionen sobre este transporte.

Protocolo:

Cada protocolo tiene su propio formato de mensajes. Es por esto que cada uno está ligado estrictamente con los grupos de comandos (explicados en el siguiente punto). Dado que cada protocolo funciona únicamente bajo ciertos transportes, también está fuertemente relacionado con ellos. Por ejemplo, NetConf funciona únicamente sobre SSH.

Tanto el transporte como el protocolo ya estaban implementados por otros proyectos, con lo cual en éste se hace uso de ellos sin modificar prácticamente su código.

Comando:

Este componente se ocupa de los comandos específicos que se hacen sobre el aparato. Se pueden separar en grupos de comandos, donde cada uno de ellos se encarga de una versión de un dispositivo. En este caso, todos los comandos son propios de NetConf para una versión del S.O. del switch (Junos), con lo cual solo tenemos un grupo de comandos. Se crearía otro diferente si además se permitiese, por ejemplo, conexión con el switch vía CLI.

Acción:

La acción es una funcionalidad de nivel más alto y, por lo tanto, más compleja. Está compuesta normalmente por un conjunto de comandos y es independiente del protocolo que se esté empleando. Pueden ser síncronas o asíncronas. En este PFC cada

acción está compuesta por un único comando. Si se decidiera cambiar de NetConf a CLI por algún motivo, este módulo no se vería alterado.

6.3.3.2 Descripción de las clases

A continuación se detallan todos aquellos módulos (clases Java) del engine que han sido necesarios para la modelización del switch. Por cada uno de ellos se describirá su función, sus componentes y una anotación que informa de si ha sido implementado o reutilizado (ya desarrollado por algún proyecto anterior).

EthernetSwitchModel (Implemetado):

Contiene la modelización del switch propiamente dicha.

- switchId: Identificador del dispositivo.
- manufacturer: Fabricante del dispositivo. En este proyecto todos los switchs son Juniper.
- model: Modelo del switch (EX-3200)
- protocol: Protocolo utilizado (NetConf).
- transport: Transporte sobre el que corre en protocolo (SSH).
- equipment: Se trata del inventario del switch. En él está la información de los puertos y las VLANs.
- security: contiene el usuario y la contraseña para conectarse con el dispositivo (ver Security).

EngineCore (Utilizado):

Está un nivel por encima del EthernetSwitchModel. Se encarga de inicializar el modelo, el transporte y el protocolo a partir de la información específica del dispositivo (transporte, protocolo, etc.) así como de recibir las acciones que se realizan sobre el aparato.

- actionHandler: gestiona las acciones (ver ActionHandler).
- protocolHandler: gestiona el protocolo utilizado para comunicarse con el dispositivo (ver NetConfProtocolHandler).

- transportHandler: gestiona el transporte utilizado para comunicarse con el dispositivo (ver SSHTransportHandler).
- engineModel: mantiene la instancia del modelo (ver EthernetSwitchModel).

NetConfProtocolHandler (Utilizado):

Contiene las acciones y funciones necesarias para la comunicación con el dispositivo vía NetConf. Envía y recibe mensajes a partir del transporte (SSH en este caso). Las peticiones se realizan encajando las cabeceras que sean oportunas en XML. Algunas de las funciones que implementa son:

- open: inicia el SSHTransportHandler. Es decir, establece la sesión para comunicarse con el dispositivo.
- sendReceive: utilizada por aquellas acciones que esperan una respuesta.
- send: utilizada por aquellas acciones que no esperan ninguna respuesta.
- close: acaba la sesión (una sesión por cada acción).
- init: define el transporte sobre el cual actuará NetConf (SSH).

SSHTransportHandler (Utilizado):

Establece la conexión SSH sobre la que correrá el protocolo que sea oportuno. Con el usuario y contraseña definidos en el modelo (ver Security), inicia, envía y recibe, y termina la comunicación con el dispositivo para cada una de las acciones.

ActionHandler (Implemetado):

Se ocupa de gestionar las acciones que se pueden lanzar contra el dispositivo y de controlar que éstas sean las correctas. El listado de acciones que soporta actualmente este engine son las siguientes:

- CreateLogicalInterfaceAction: Crea una o varias interfaces lógicas sobre una física.
- DeleteLogicalInterfaceAction: Elimina una o varias interfaces lógicas.
- UpdateLogicalInterfaceAction: Actualiza una o varias interfaces lógicas.
- CreateVlanAction: Crea una o varias VLANs.
- DeleteVlanAction: Elimina una o varias VLANs.

- UpdateVlanAction: Actualiza una o varias VLANs.
- QueryResourcesAction: Pide la configuración de las interfaces y VLANs del switch por tal de actualizar el modelo del engine.
- ConfigurePhysicalPortAction: Configura una o varias interfaces físicas.

Cada una de las acciones anteriores tiene su correspondiente comando (CreateLogicalInterfaceCommand, etc). Además, ActionHandler contiene el protocolo (NetConfProtocolHandler en este caso) para poder lanzar las acciones sobre él.

Security (Utilizado):

Tiene una serie de parámetros de seguridad para la comunicación con los dispositivos, como por ejemplo, la posibilidad de incluir certificados, intermediarios o *passwords* privilegiados. En este proyecto sólo se utiliza una de sus funcionalidades: la de establecer un nombre de usuario y una contraseña por tal de poder ser identificados por el dispositivo.

Equipment (Implementado):

Contiene tanto la configuración actual del switch como aquella que está por aplicar (información para las acciones).

- configuredPhysicalPorts: Lista con la configuración de cada uno de los puertos físicos del switch.
- requestedPhysicalPorts: Lista con la configuración de aquellos puertos que se quieren modificar en una acción determinada.
- configuredVlans: Lista de VLANs configuradas en el switch. Cada VLAN tiene un identificador, nombre y descripción, así como la lista de interfaces que forman parte de ella, entre otros parámetros.
- requestedVlans: Lista con las VLANs que se quieren configurar en una acción determinada.

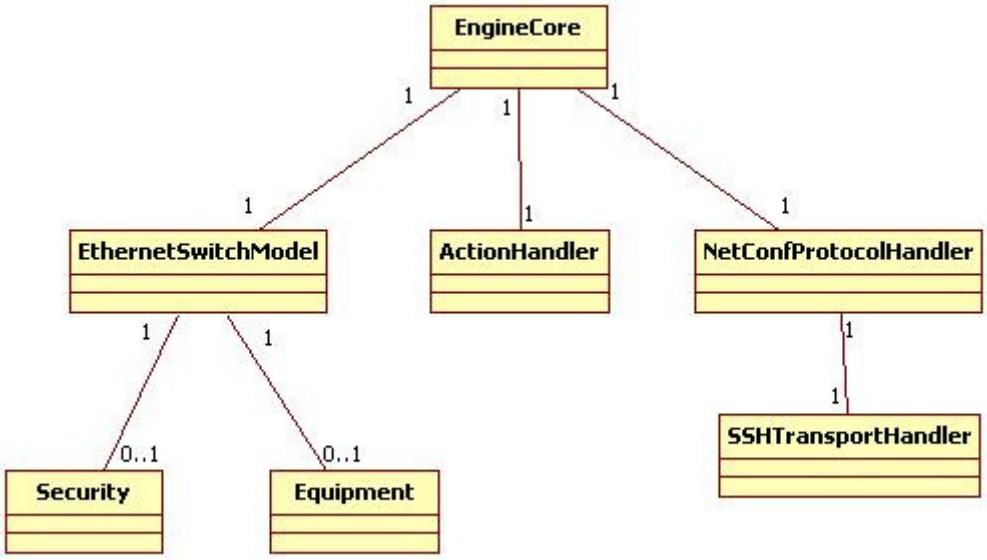


Diagrama UML 6: De clases – Modelo conceptual (parte 1)

PhysicalEthernetSwitchPort (Implementado):

Es la modelización de una interfaz física del switch.

- name: Nombre del puerto físico, por ejemplo ge-0/0/2.
- description: Descripción del puerto. Es útil para tener información de referencia sobre lo que hará la interfaz, dónde estará conectada, el tráfico que circulará por ella, etc.
- linkSpeed: Capacidad del puerto. En el switch Juniper EX-3200 puede ir a 10Mpbs, 100Mbps o 1Gbps.
- linkMode: Determina si el puerto actúa como full-duplex, half-duples o automático.
- configuredUnits: Contiene la lista de interfaces virtuales (también denominadas units) en que se subdivide cada puerto físico del switch (ver UnitType).

UnitType (Implementado):

Son las unidades virtuales en que se divide un puerto. Cada una de ellas actúa como si se tratase de una única interfaz física. UnitType es una clase abstracta de la cual heredan dos clases diferenciadas por el tipo de configuración de la interfaz lógica

(EthernetSwitchingUnitType e InetUnitType). Los parámetros que tienen en común son los siguientes:

- family: es el campo que determina si se trata de un puerto configurado para funcionar como ethernet-switching o como inet.
- description: Descripción del puerto lógico. Es útil para tener información de referencia sobre lo que hará la interfaz, dónde estará conectada, el tráfico que circulará por ella, etc.
- state: para determinar si la unidad está activada (enabled) o desactivada (disabled).
- interfaceName: Contiene el nombre de la interfaz física a la que pertenece la *unit*.
- unitName: Cada *unit* se identifica por un dígito cuyo rango va del 0 al 16384. El nombre completo de la interfaz lógica se compone por interfaceName.unitName; por ejemplo ge-0/0/3.0 o ge-0/1/0.11.

EthernetSwitchingUnitType (Implementado):

Funcionan bajo Ethernet estrictamente. Sus atributos específicos son:

- trunkPortMode: indica si el puerto está en modo *trunk* o en modo acceso.
- nativeVlanId: si el puerto está en modo *trunk*, determina la VLAN por defecto para aquellos paquetes que reciba sin etiquetar.
- configuredVlans: contiene la lista de VLANs a las que pertenece. Si no es *trunk* solo puede pertenecer a una.

InetUnitType (Implementado):

Tiene una dirección IP para conectarse a otras VLANs a nivel 3. No se utiliza en el proyecto pero está implementado por si fuera necesario en una etapa posterior.

- IPAddress: Dirección IP del puerto en cuestión.

VLANType (Implementado):

Modelización de una Virtual LAN.

- name: nombre identificador y único de la VLAN.

- tag: número identificador y único de la VLAN.
- description: Descripción de la VLAN. Es útil para tener información de referencia sobre el uso de la VLAN en cuestión.
- state: Indica si la VLAN está activa o inactiva.
- units: Listado de las interfaces que pertenecen a la VLAN.

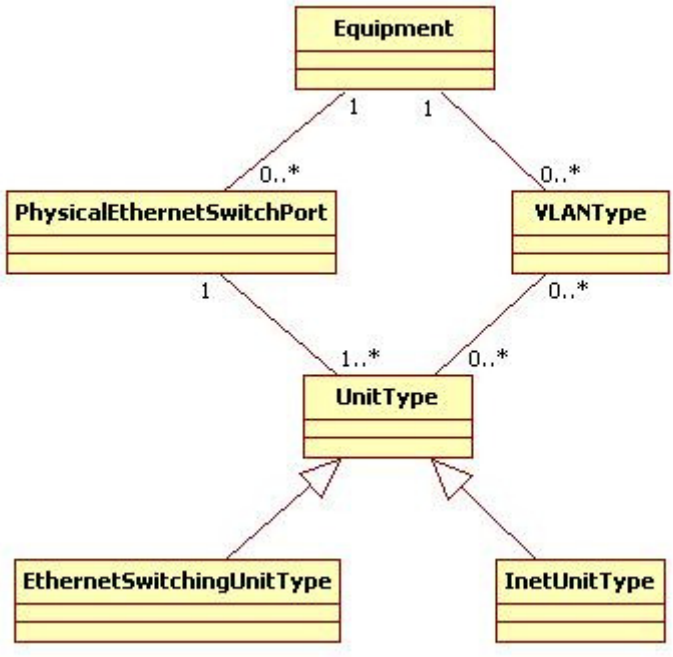


Diagrama UML 7: De clases – Modelo conceptual (parte 2)

ObjectMapper (Implementado):

Es un conjunto de acciones y funciones que parsean la respuesta del switch cuando se realiza la acción que actualiza el modelo (queryResources). El switch envía su configuración actual en formato XML; esta clase se encarga de transformar esta información en el modelo descrito en los módulos anteriores.

Otros Módulos:

- EngineConstants (Utilizado): algunas constantes necesarias para el engine.
- JuniperEX3200 (Implementado): contiene información por defecto para este modelo y fabricante (lista de acciones, protocolo, transporte, modelo, nombre etc…).

En la siguiente figura se muestra un modelo conceptual básico con todas las clases que componen el engine:

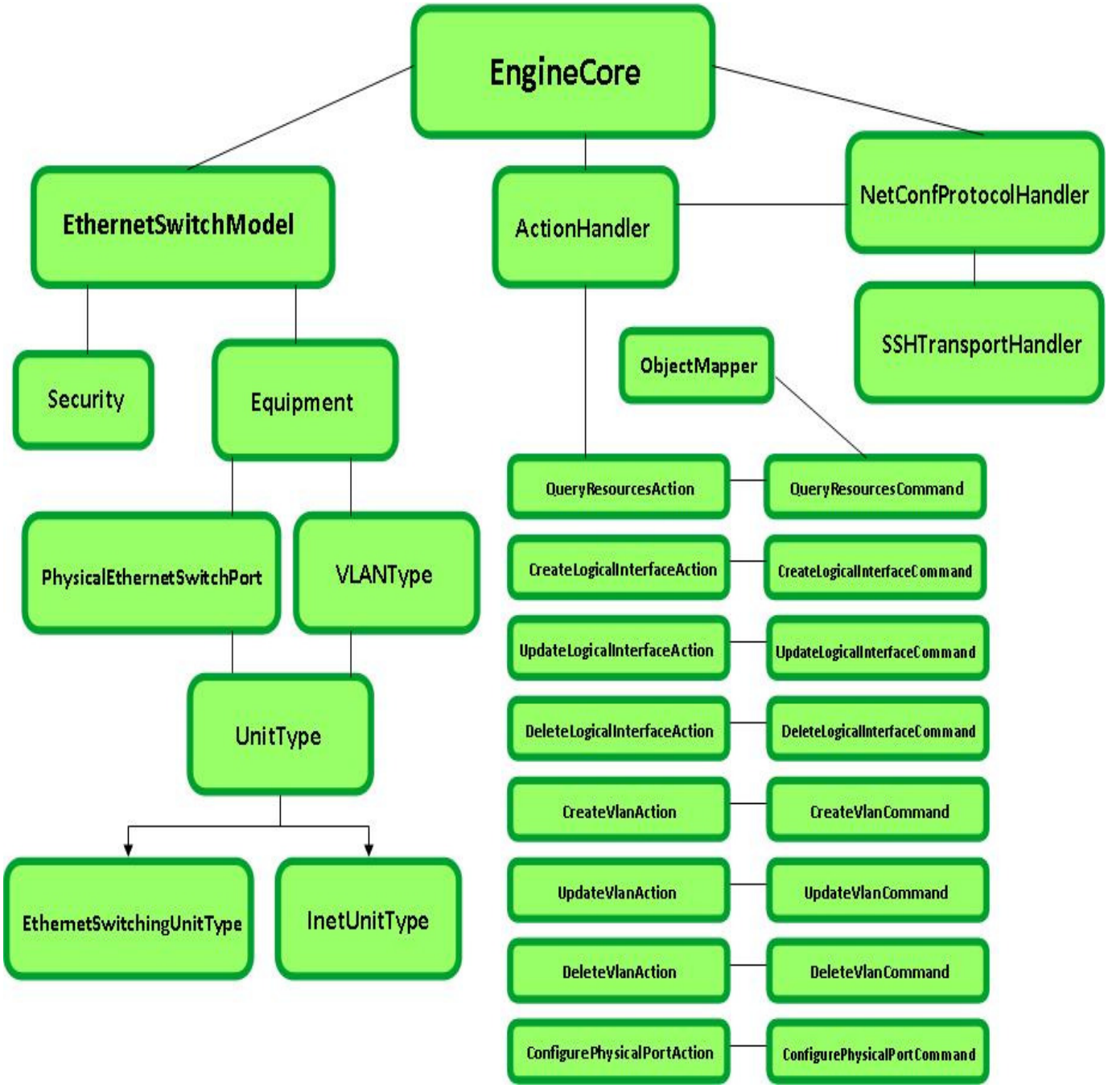


Figura 13: Modelo conceptual básico del engine

6.3.4 Desarrollo de los Web Services

Este módulo del PFC es el que permite la comunicación remota entre la aplicación o GUI local donde estará el administrador y los servicios ofrecidos. Está desarrollado con Globus Toolkit. Se trata de un software libre para construir *grids* (parrillas) computacionales. Tiene muchas funcionalidades de entre las cuales este proyecto utiliza la de crear una serie de servicios (Java Web Service Core) con persistencia (vía Hibernate) de cierta información.

A continuación se describe cómo se han desarrollado estos servicios.

6.3.4.1 Conocimientos previos

Antes de detallar la implementación, es necesario tener claros ciertos conceptos para entender el trabajo realizado. ¿Qué es un Web Service? Se trata de una tecnología para aplicaciones distribuidas (como COBRA, RMI, EJB, etc). Permite crear aplicaciones cliente/servidor. Como ejemplo sencillo, supongamos que se mantiene una base de datos con el la situación climatológica en tiempo real de un país y se pretende distribuir esta información a cualquiera que esté interesado. Para hacerlo, se puede "publicar" dicha información a través de un Web Service que, dado un código postal, proporcione las condiciones climatológicas. No hay que confundirlo con publicar algo en una página Web. En éste último caso, la información está dirigida a ser interpretada por humanos. La información proporcionada por un Web Service siempre está pensada para ser interpretada por software. Es decir, no tienen relación alguna con el los sitios Web o el HTML. Las ventajas de los Web Services sobre el resto de tecnologías de este tipo son:

- Son independientes de la plataforma y del lenguaje de programación ya que utilizan estándares de XML. Es decir, el cliente puede tener una aplicación implementada en C++ corriendo bajo Windows mientras que el Web Service está implementado en Java bajo Linux.
- Utilizan HTTP para transmitir los mensajes. Es una ventaja importante si, como en este caso, se pretende construir una aplicación que funcione a través de Internet (COBRA, por ejemplo, tiene problemas con *firewalls* debido a no tener esta cualidad).

Por otro lado, también tiene ciertas desventajas:

- *Overhead*: Transmitir todos los datos en XML no es tan eficiente como utilizar un código binario propio y adaptado. Lo que se gana en portabilidad se pierde en eficiencia. En este proyecto, como en la mayoría de este tipo, el envío ineficiente de información es aceptable.
- Falta de versatilidad: Ofrecen formas muy básicas de invocar los servicios. No obstante existen especificaciones (como WSRF, que se verá a continuación) que aportan más versatilidad.

La arquitectura de cualquier Web Service tiene la siguiente estructura:

Figura 14: Arquitectura de los Web Services

- Procesos: Tiene, entre otras cosas, servicios de descubrimiento de entre un conjunto de Web Services. En este PFC las direcciones de los WS (URIs) están definidas por lo que no se necesitan este tipo de procesos.
- Descripción: Una de las particularidades de los WS es que se describen a si mismos. De ello se ocupa el Web Service Description Language (WSDL).

- Invocación: Esto incluye el flujo de información cliente/servidor. SOAP especifica el formato de las peticiones al servidor y cómo deben ser las respuestas.
- Transporte: Como en las páginas Web convencionales de Internet, HyperText Transfer Protocol se encarga del transporte.

A continuación se muestra un diagrama con el procedimiento típico que se lleva a cabo en una invocación a un servicio de este tipo:

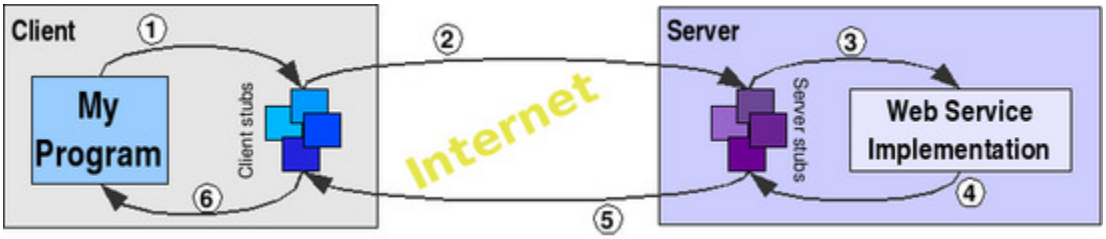


Figura 15: Invocación típica de un Web Service

1. Cuando un cliente realiza una petición sobre un WS, lo hace realmente sobre lo que se denomina *stub* del cliente, los cuales están generados automáticamente a partir del WSDL. Éstos se encargan de transformar la solicitud "local" en su correspondiente petición SOAP.
2. La petición SOAP se envía sobre la red usando el protocolo HTTP. El servidor recibe el mensaje SOAP y lo entrega a los *stubs* del servidor. Éstos lo transforman en algo que será interpretado por la implementación del servicio.
3. Hecho esto, los *stubs* del servidor invocan la implementación del servicio, la cual llevará a cabo el trabajo que se ha solicitado.
4. El resultado es manipulado por los *stubs* del servidor, transformándolo esta vez en una respuesta SOAP.
5. La respuesta SOAP se envía sobre la red usando el protocolo HTTP. Los *stubs* del cliente reciben el mensaje y lo transforman en algo que será interpretado por la implementación de la aplicación del cliente.

6. Finalmente la aplicación recibe el resultado de la invocación al WS y lo utiliza para aquello que sea necesario.

Otro concepto que hay que tener en cuenta es el WSRF (Web Service Resource Framework), mencionado anteriormente. Se trata de una especificación para WS que mejora varios aspectos por tal de hacer que los Web Services se ajusten más al *grid computing*. La más destacada y fundamental para este proyecto es que permite mantener el estado. Los Web Services planos son normalmente *stateless*, es decir, no pueden guardar información entre una invocación y otra. Para ello, y por tal de no contradecir la filosofía de este tipo de tecnología, se separa claramente lo que es el Web Service de la información del estado. Dicha información se mantiene en una entidad llamada *resource*. Cada *resource* tiene una clave única, con lo cual lo único que se tiene que hacer al llamar al servicio es indicarle la *resource* particular a la que se quiere acceder. Estas *resources* se pueden mantener en memoria o en una base de datos. En este proyecto se guardan ciertos parámetros en la BD mientras que otros son cargados dinámicamente.

Para acabar con la puesta en situación, es necesario familiarizarse con la terminología y las propiedades de WSRF que se utilizan en este PFC:

- El conjunto Web Service más *resource* se denomina WS-resource.
- La dirección de un WS-resource se denomina *endpoint reference* (EPR).
- WS-ResourceProperties: Un *resource* está compuesta de una o más *resource properties*. Especifica cómo son las propiedades del recurso y cómo se accede a ellas. Están definidas en la descripción WSDL.
- WS-BaseFaults: Proporciona una manera estándar de cobertura de errores cuando algo no funciona como es debido durante la invocación del WS.
- WS-Addressing: Mecanismo para localizar los WS más versátil que las URIs. Se puede utilizar WS-Addressing para encontrar el par WS más recurso (WS-resource).
- *Namespace*: nombre dado a uno o varios Web Services. Suelen ser de la forma "http://core.upc.cat/2008/12/ethernetswitch".

6.3.4.2 Ethernet Switch Web Services

A partir de este punto se detalla el procedimiento que se ha seguido para implementar los Web Services necesarios en este proyecto. Se describirán los WS que se encargan de comunicarse con el engine del punto anterior:

- EthernetSwitchService: Habrá una instancia de este Web Service. Dispone de una serie de recursos (resources); uno por cada switch (engine) del que disponga la aplicación. Permite que el usuario ejecute las siguientes operaciones:
    - Invoke: Invoca una acción sobre el engine (crear VLAN, configurar puerto, añadir puerto a VLAN, etc).
    - Refresh: Fuerza al WS a llamar al engine para actualizar la información del switch (query resources).
    - Virtualize: Crea un EthernetResource. Es decir, crea un recurso para que pueda ser utilizado por un usuario de FEDERICA.

- EthernetSwitchFactoryService: Es una única instancia sin persistencia que se encarga de gestionar los EthernetSwitchServices. Sus operaciones son:
    - Create: Crea un nuevo WS-resource. Es decir, el servicio EthernetSwitchService junto con su módulo encargado de la información con estado (EthernetSwitchResource).
    - Clone: Crea un nuevo EthernetSwitch a partir de otro ya existente.
    - Find: Devuelve los EPR de aquellos EthernetSwitch que coincidan con una plantilla dada.

Los pasos que se han llevado a cabo para implementar estos Web Services son 5:

1. Definir la interfaz de los servicios. Realizado con WSDL.

Especifica lo que ofrecerán nuestros servicios. En este punto no importa cómo estén implementados; ni qué algoritmos tendrán; ni cómo interactúan con otros sistemas. Tan solo define qué operaciones estarán disponibles para los usuarios. Se

emplea un lenguaje XML especial para definir Web Services (WSDL, WS Description Language). Cada WS tiene su propia definición. Por lo tanto, tenemos los siguientes archivos:

- EthernetSwitch.wsdl contiene la descripción en XML de los servicios que proporciona (también denominado *portType*), así como del recurso adjunto al WS y la declaración de los mensajes de petición/respuesta que éstos reciben/envían.
- EthernetSwitchFactory.wsdl está compuesto de la misma manera, pero define los parámetros de la factoría de EthernetSwitch.
- EthernetSwitch.xsd se utiliza para describir la estructura y las restricciones de los mensajes de las operaciones y los recursos (resources) de los dos ficheros anteriores. Lo hace de una forma muy precisa más allá de las normas sintácticas impuestas por el propio lenguaje XML (XML-Schema).
- namespace2package.mappings: Debido a que WSDL no menciona el lenguaje en que será implementado ni el servicio ni el cliente, se necesita un nexo, en este caso con Java. De esto se encargan los *stubs*, los cuales son generados a partir de los ficheros WSDL utilizando GT4. Para ello, necesitamos este archivo, que se encarga de decirle a la herramienta dónde (en qué paquete Java) debe colocarlos. Mapea los *namespaces* WSDL a paquetes Java.

```
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
```

La primera línea mapea el *namespace* de los dos ficheros WSDL. Los otros dos *namespaces* se generan automáticamente cuando Globos Toolkit interpreta sus descripciones.

2. Implementar los servicios. Realizado con Java.

En este punto se implementa lo descrito en el paso anterior. Es decir, se contesta a la pregunta ¿cómo hace el servicio lo que dice que hace? Para ello, se presentan las clases Java que han sido implementadas:

- EthernetSwitchQNames: Para referirse a algún parámetro de los Web Services hay que hacerlo a partir de los *qualified names* o QNames. Están formados por un *namespace* más el nombre de la propiedad referida. Su traducción en java presenta el siguiente aspecto:

```
[…]
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");
public static final QName RP_CONFIGURATION = new QName(NS, "Configuration");
[…]
```

  Debido a que se hace referencia frecuentemente a los QNames, es una buena práctica colocarlos separados en esta clase.

- EthernetSwitchService presenta la implementación de las operaciones remotas que se pueden efectuar sobre el servicio (invoke, refresh y virtualize). Los parámetros y los valores de retorno de éstas deben ser de una manera determinada, ya que son interpretados por los *stubs* (generados automáticamente). También implementa operaciones que se encargan de obtener los recursos relacionados.
- EthernetSwitchResource. Tiene las siguientes *resource properties*:
  - Name: Guarda el nombre del switch (con persistencia en la BD).
  - Model: Guarda el modelo del switch (con persistencia en la BD).
  - Manufacturer: Guarda el fabricante del switch (con persistencia en la BD).
  - Configuration: Guarda la información necesaria para la comunicación con el switch: dirección IP, puerto, usuario, contraseña, transporte y protocolo (con persistencia en la BD).

  Es importante que estos atributos tengan el mismo nombre que las *resource properties* definidas en el fichero WSDL. Además, esta clase mantiene una instancia del engine sobre la cual se transmiten las operaciones y que proporciona la información de la configuración del dispositivo. Es decir, esta

información no se mantiene en la BD si no que se carga cada vez que se inicia el Web Service. Finalmente, implementa funciones de inicialización (initEngine), de guardado (store) y carga (load) en la base de datos.

- EthernetSwitchResourceHome es utilizado por EthernetSwitchFactory WS para crear recursos para cierto EthernetSwitch WS.
- EthernetSwitchStore representa una única tabla de hash donde se guardan los engines activos durante la ejecución de los Web Services. EthernetSwitchResource accede a ella para gestionar (guardar, cargar, eliminar) su engine.
- EthernetSwitchFactoryService implementa las tres operaciones descritas en el fichero WSDL.

La siguiente figura representa la interacción de todas las clases implicadas en la implementación de los servicios:

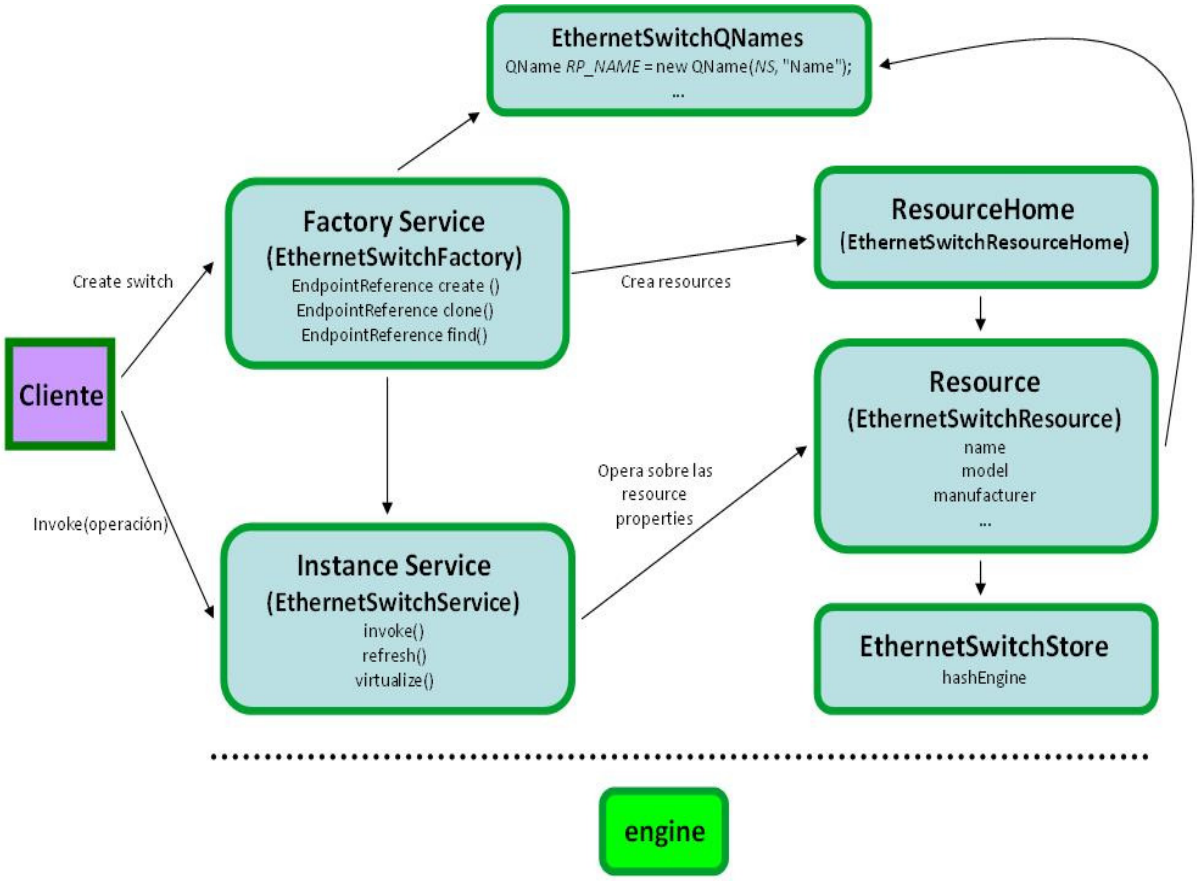


Figura 16: Interacción de las clases del Ethernet Switch Web Service

Otras clases de interés que forman parte de la implementación del Ethernet Switch Web Service:

- ObjectMapper: Mapea la información de la configuración del switch (engine) a la estructura de los mensajes que deben saber interpretar los *stubs*.
- EthernetSwitchConstants: Conjunto de constantes que se utilizan en la ejecución de los servicios.
- EthernetResourceWrapper: Se encarga de la comunicación entre EthernetSwitchService y EthernetResource (WS que se ocupa de las interfaces virtualizadas). Se utiliza en la operación virtualize.

3. Definir los parámetros de despliegue. Realizado con WSDD y JNDI.

Una vez definidos e implementados los WS, se procede a desplegarlos para hacerlos disponibles. Para ello son fundamentales dos ficheros:

- deploy-server.wsdd: WS Deployment Descriptor (XML). Describe cómo deben publicarse los Web Services. Entre otras cosas, define las direcciones (URIs) de los servicios, señala las clases Java que los implementan, etc.

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

Estas líneas especifican la localización de los Web Services. Combinados con la dirección base del servidor se obtiene la URI de cada uno. Por lo tanto, poniendo por ejemplo que la dirección base del servidor es "http://147.83.118.238:8080/wsrf/services", las URIs quedarían de la siguiente manera:

http://147.83.118.238:8080/wsrf/services/core/EthernetSwitchService y

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

- deploy-jndi-config.xml: Define qué *resource home* debe utilizarse para cada servicio y otros parámetros por tal de conseguir un despliegue correcto de los Web Services.

4. Compilar todo y generar los archivos .gar. Realizado con Ant.

En este punto del proceso se tienen (1) las definiciones de los servicios en WSDL, (2) la implementación en Java y (3) la descripción del despliegue (WSDD y JNDI) que dicen cómo se presentan (1) y (2) al mundo exterior. Sin embargo, son un conjunto de ficheros sueltos. ¿Cómo se colocan en el servidor? ¿Se copian en ciertos directorios? ¿Qué sucede con las clases Java? GT4 se encarga de generar un archivo *Grid* (GAR) que contiene toda la información que los Web Services necesitan para ser desplegados en el servidor. El proceso de creación del GAR puede ser muy complicado. Requiere los siguientes pasos:

- Procesar los ficheros WSDL.
- Crear los *stubs* a partir de los WSDL.
- Compilar las clases *stub*.
- Compilar la implementación de los servicios.
- Organizar todos los ficheros de un modo específico.

Esto está automatizado gracias a la herramienta de construcción Java llamada ANT (Apache). Conceptualmente es muy similar al *make* de Unix. Simplemente se necesita un fichero build.xml con ciertos parámetros (ubicación de ciertos ficheros, de los paquetes Java, etc) y se obtiene el archivo gar. El siguiente diagrama muestra todo lo necesario para llevarlo a cabo:

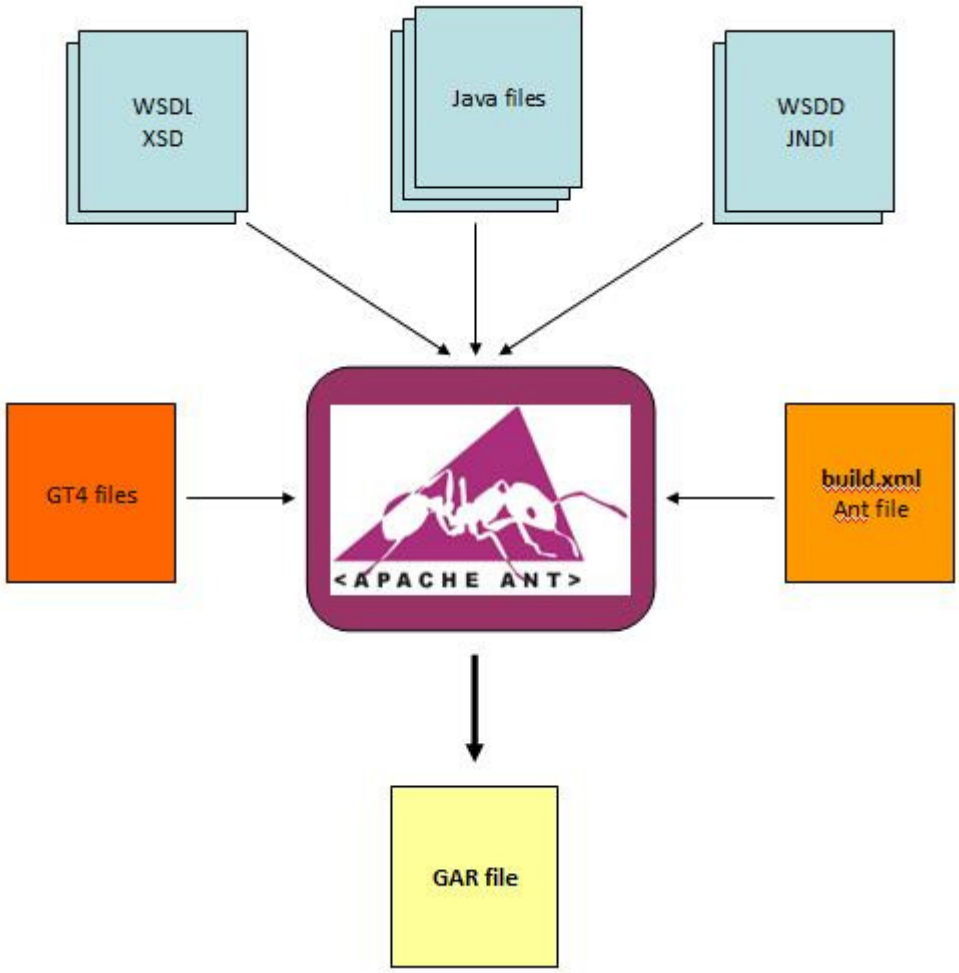


Figura 17: Generación del archivo GAR con Ant

Por lo tanto, simplemente se trata de configurar el build.xml y ejecutarlo desde Eclipse, obteniendo como resultado el fichero ethernetswitch.gar.

5. Desplegar los servicios en el servidor. Realizado con Globus Toolkit 4.

El despliegue se hace a través de GT4; utilizando Ant, desempaqueta el .gar y copia los ficheros que contiene (WSDL, *stubs* compilados, implementación compilada, WSDD, etc) en localizaciones clave dentro del árbol de directorios de Globos Toolkit. Una vez se tiene el GAR en el servidor, se ejecuta el siguiente comando para desplegarlo/agruparlo:

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

Para poner en marcha el servcio solo queda solo queda activar el contenedor:

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

Cuando el container está activado, se muestra una lista con las direcciones (URIs) de todos los servicios desplegados en el servidor. Entre ellas se muestran las de los dos que han sido implementados:

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

Una vez se han iniciado los Web Services, si existen switchs y/o recursos Ethernet en la Base de Datos, se carga toda la información de éstos en *resources* para que puedan ser accesibles remotamente desde la aplicación.

6.3.4.3 Ethernet Resource Web Services

El procedimiento de implementación es muy similar para EthernetResource Web Services. De los cinco pasos descritos, lo único que difiere son las operaciones que se realizan sobre los WS y los parámetros de los recursos. EthernetResourceService es el servicio que gestiona las instancias de las interfaces virtualizadas. Se obtiene como resultado dos URIs más en el servidor:

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

Los recursos de EthernetResource tienen mucha información ya que también son utilizados para representar interfaces a nivel tres: ipAddressV4, ipAddressV6, netmaskV4, netmaskV6, connectedTo, entre muchos otros. De todos ellos nos interesan los siguientes:

- networkElement: Contiene el nombre de la interfaz que ha sido virtualizada. Mantiene el formato *interfaceName.unitName* (ge-0/0/3.0).

- physicalBinding: Contiene el nombre del dispositivo físico al que pertenece la interfaz.
- owner: Contiene el nombre del usuario al que se asigna el recurso.
- vlans: Contiene la VLAN asignada a la interfaz.
- isTrunk: Indica si el puerto lógico es *trunk* o *access*.

Todos los atributos descritos tienen persistencia en la base de datos y han sido añadidos durante la elaboración del proyecto. El comportamiento de estos Web Services se verá en el punto de interacción de los módulos.

En esta sección, el término resource (o recurso) se refería al módulo del Web Service que se encarga de mantener el estado. De ahora en adelante el término recurso o *resource* se referirá, si no se expresa explícitamente, a los dispositivos y a aquellas interfaces del switch que son virtualizadas. Es decir, recursos de la infraestructura de FEDERICA.

6.3.4.4 Desarrollo de la base de datos

Cada una de las tablas de la base de datos se genera mediante Hibernate. Éste mapea las clases Java definidas en los WS (EthernetSwitchResource y EthernetResource) con un archivo XML que define los campos a partir de los atributos que se deseen de dichas clases.

El sistema de BD consta de las siguientes tablas:

Tabla ethernet_switch_resources		
Campo	**Tipo**	**Descripción**
resourcekey (PK)	character varying (255)	Clave primaria
version	integer	Versión del producto
name	character varying (255)	Nombre del switch
model	character varying (255)	Modelo del switch
manufacturer	character varying (255)	Fabricante del switch
destructiontime	big int	Tiempo de vida del switch

Tabla switch_configmodules		
Campo	**Tipo**	**Descripción**
resourcekey (FK)	character varying (255)	Clave foránea a un switch
configmodules	character varying (1000)	Transporte y protocolo

Tabla switch_configparameters		
Campo	**Tipo**	**Descripción**
resourcekey (FK)	character varying (255)	Clave foránea a un switch
configparameters	character varying (1000)	Contiene un parámetro

Las anteriores son las tablas para los switchs. En cuanto a los recursos tenemos las dos tablas que se muestran a continuación. Para la tabla principal, se omiten algunos de los campos debido a que no se utilizan en este proyecto.

Tabla ethernet_resources		
Campo	**Tipo**	**Descripción**
resourcekey (PK)	character varying (255)	Clave primaria
owner	character varying (255)	Propietario del recurso
isTrunk	boolean	Trunk/Acces
type	character varying (255)	Tipo del recurso
physicalbinding	character varying (255)	Dispositivo vinculado
destructiontime	big int	Tiempo de vida del switch

Tabla ethernetresource_vlans		
Campo	**Tipo**	**Descripción**
resourcekey (FK)	character varying (255)	Clave foránea a un recurso
vlan	character varying (255)	Contiene VLAN id

6.3.5 Desarrollo de la GUI (Graphical User Interface)

La interfaz gráfica de este PFC es temporal. Ha sido implementada en Eclipse mediante RCP (Rich Client Platform). Se trata de una GUI sencilla basada en *wizards* para comprobar y testear el funcionamiento de los WS y el engine. Con lo cual no implementa todas las funcionalidades de las que son capaces los otros módulos. Además, sufre ciertas limitaciones que sí deberán ser explotadas por la aplicación final dentro de FEDERICA. Por ejemplo, la interfaz tan solo permite la asignación de un puerto a cierta VLAN. Tanto los Web Services como el engine están programados para permitir la asignación de varias unidades a una VLAN en la misma instrucción. De esta manera se reduce considerablemente la velocidad de configuración, ya que se realizarían las operaciones con una única conexión al dispositivo (switch).

Esta sección se divide en dos partes: primero se hace una explicación de RCP para después describir las clases más importantes de la aplicación.

Rich Client Platform tiene los siguientes componentes:

- Núcleo (microkernel)
- Framework estándar
- Widget toolkit
- Buffer de ficheros, manejo de texto
- Workbench (vistas, editores, perspectivas, wizards)
- Mapeo de datos (XML)
- Gestor de actualizaciones

Con todo lo anterior, se puede programar una aplicación desde una base ofrecida por la plataforma en lugar de tener que crearla de cero. Una de sus principales cualidades es que se pueden construir interfaces gráficas independientes del sistema operativo en el que se deseen utilizar, manteniendo la riqueza/calidad necesaria para GUIs cliente-servidor. También permite ejecutar y desplegar sus aplicaciones sin que el usuario conozca nada del código Java o Eclipse que hay debajo. Con lo cual, se puede generar un ejecutable con proceso de instalación similar a InstallShield o NSIS. Algunos

ejemplos de programas desarrollados con RCP son el propio Eclipse, NetBeans, Spring Framework.

La interfaz gráfica de este PFC está desarrollada como un *plug-in* de Eclipse. Para ello, se ha creado un proyecto a través de PDE (Plug-in Development Environment) que sirve de guía en el proceso a partir de una serie de *wizards*. Se le proporcionan ciertos parámetros (nombre, versión de Eclipse bajo la que se programa…) y, a partir de alguna sus plantillas (en este caso se ha utilizado RCP Mail Template), te construye el esqueleto sobre la cual se erigirá la aplicación. Las clases más importantes de este esqueleto están descritas a continuación:

- MANIFEST.MF: Generado en el proceso de creación del proyecto. Contiene la información necesaria para cargar dinámicamente los plug-in's. También se encuentran en este fichero las librerías necesarias: las de GT, *stubs*, engine, etc…
- Plugin.xml: Define las extensiones (application, perspectives, views, wizards…) de la aplicación.
- Application: Rutina principal de la aplicación. Se trata del controlador del programa. Es ligero de código y suele tener siempre la misma estructura: Crea un workbench y adjunta la clase ApplicationWorkbenchAdvisor.
- Perspective: Es el contenedor de las vistas de la aplicación. Presenta el siguiente aspecto:

```
[…]

public class Perspective implements IPerspectiveFactory {
  public void createInitialLayout(IPageLayout layout) {
    String editorArea = layout.getEditorArea();
    layout.setEditorAreaVisible(false);

    layout.addStandaloneView(NavigationView.ID,false,
       IPageLayout.LEFT,0.25f, editorArea);
    layout.getViewLayout(NavigationView.ID).setCloseable(false);
  }
}
```

El área de edición se deja desactivada ya que no se utiliza en esta GUI. Lo único que contiene la perspectiva es la ventana de navegación donde se muestra la lista con los switchs de la infraestructura. La otra perspectiva, la que muestra

los recursos para los usuarios finales, tiene prácticamente el mismo aspecto y se llama UserResourcesPerspective.

- Advisors: sirven para configurar el workbench de la aplicación.
    - ApplicationWorkbenchAdvisor: Contiene la perspectiva por defecto de la aplicación.
    - ApplicationWorkbenchWindowAdvisor: Parámetros de configuración de las ventanas: tamaño, título, barra de herramientas entre otras cosas.
    - ApplicationActionBarAdvisor: Configura la barra de herramientas, la barra de estado y menús.

En definitiva, esta interfaz gráfica se compone simplemente de una *application* con dos perspectivas: una que contiene una vista donde figura la configuración de los switchs y sobre la cual se realizan las operaciones. La otra contiene la lista de recursos que se proporcionan a los usuarios de FEDERICA. Ahora se verán las clases propias del PFC:

- NavigationView: Es la vista donde aparecen, en forma de árbol, los switchs que se encuentran el la base de datos. Se carga al iniciar el programa obteniendo la lista conectando con los Web Services (mediante EthernetSwitchActions). El árbol tiene la estructura siguiente:

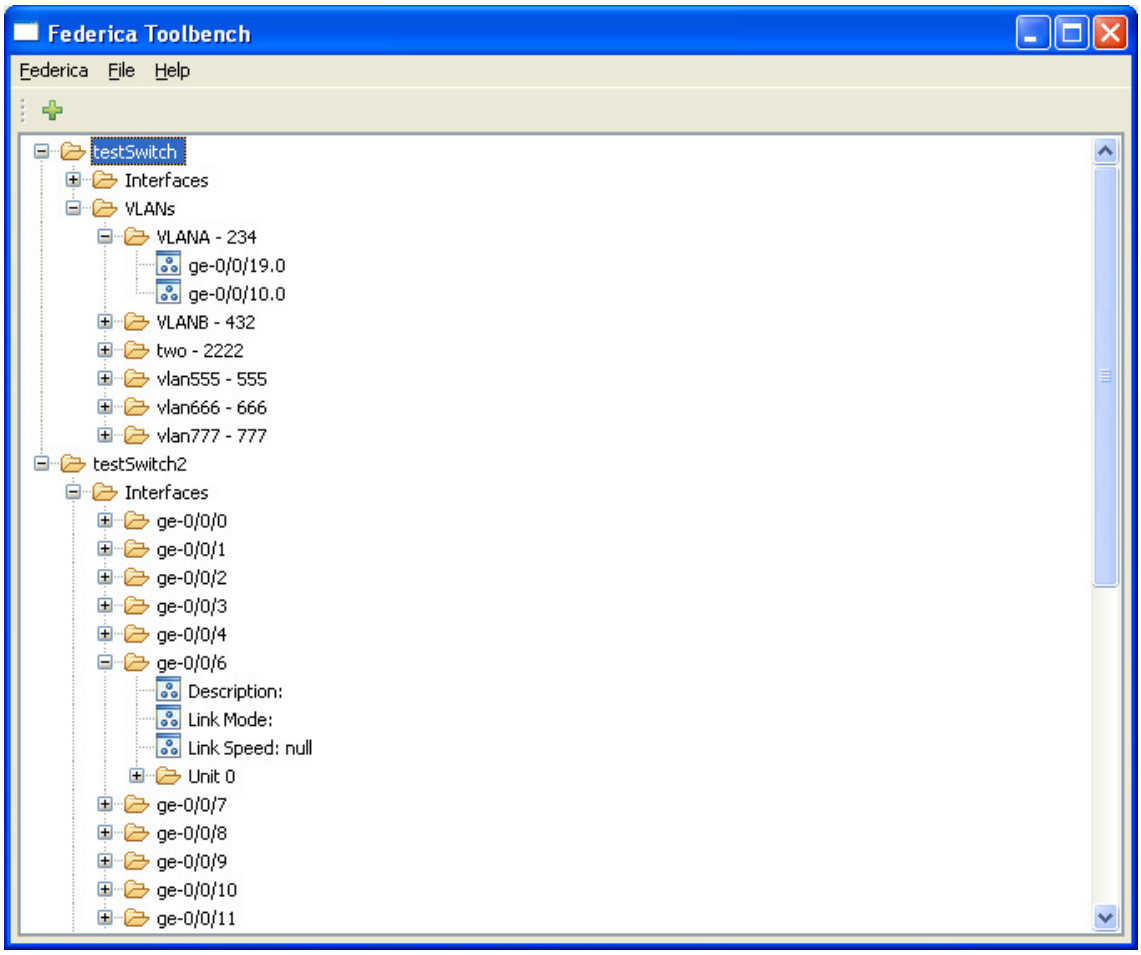


Figura 18: Lista de los switchs con sus respectivas configuraciones (puertos y vlans)

Cada switch tiene como hijos las interfaces y las VLANs configuradas. Para cada interfaz se muestran sus parámetros así como los puertos lógicas que la componen, que a su vez tienen su propia configuración (trunk/acces, descripción…). Cada VLAN contiene una lista con los miembros (puertos lógicos) que forman parte de ella.

Además, esta vista es la que recibe las operaciones que se realizan sobre los dispositivos. Al hacer clic con el botón derecho sobre alguno de los switchs, se abre un desplegable con las operaciones disponibles: configurar interfaz, actualizar switch, crear vlan, etc. Para añadir un switch a la infraestructura, no hay más que hacer clic derecho sobre un punto vacío de la vista.

- UserResourcesView: Es la vista contenida dentro de la otra perspectiva de la aplicación (UserResourcesPerspective) donde aparecen, también en forma de árbol, aquellos puertos que han sido virtualizados. Cada uno de estos recursos

contiene el nombre de la interfaz, el usuario al que es asignado y el dispositivo físico al que está asociado. Además, también contendrá la VLAN por la cual circulará su flujo de información y un campo que informa de si el puerto lógico es *trunk* o *access*. Esta vista es para que el administrador tenga presente todos aquellos recursos que ya han sido publicados. Presenta el siguiente aspecto:

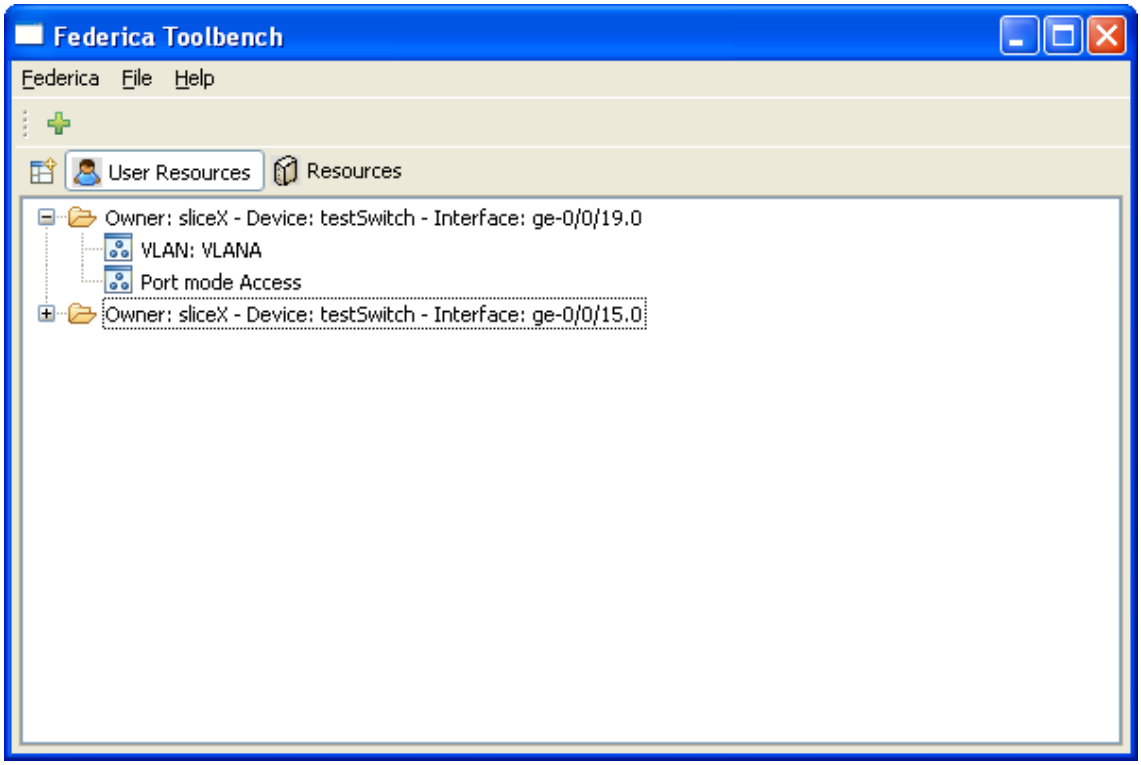


Figura 19: Lista de los recursos para los usuarios

- Wizards: Un *wizard* está formado por una serie de paneles; cada uno de ellos contiene parámetros configurables tales como listas, cuadros de texto, etc. La idea consiste en que el usuario, haciendo clic en los botones "siguiente"/"anterior", vaya de un panel a otro introduciendo la información requerida hasta completar el camino y conseguir cierta configuración (botón "finalizar"). Es posible anular la operación que se está ejecutando haciendo clic en el botón "cancelar". Por lo tanto, cada vez que se realiza una operación, se lanza un *wizard* que guía al usuario por tal de llevarla a cabo. Esta interfaz gráfica está basada en este sistema de procedimiento. Los *wizards* que implementa son los siguientes:

- AssignVlanToInterface: Asigna una interfaz a una VLAN. Está formado por tres paneles. En el primero se selecciona la VLAN a la cual se desea asignar el puerto. El segundo muestra una lista con las interfaces disponibles. El último muestra si la operación se ha realizado con éxito.

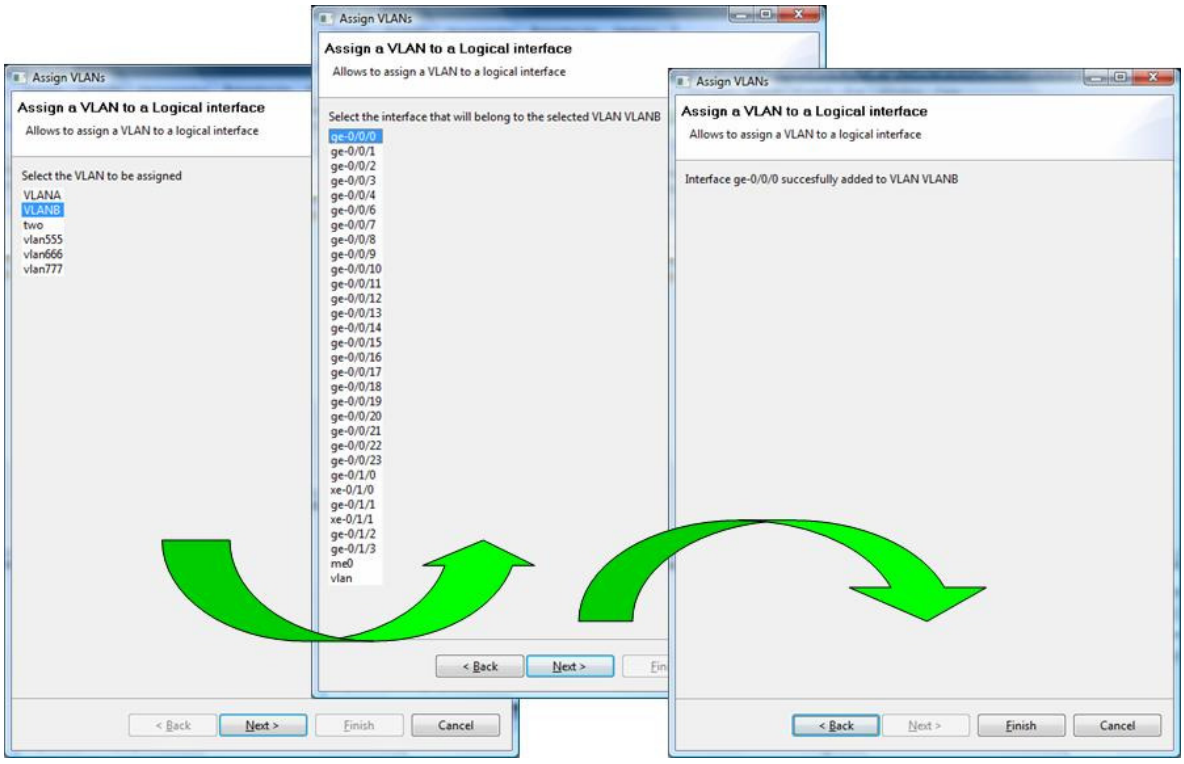


Figura 20: Wizard para asignar una interfaz a una VLAN

- ConfigureUnit: Configura una interfaz lógica. Está formado por tres paneles. En el primero se selecciona la interfaz que se desea modificar. En el siguiente se introduce la nueva configuración. En esta GUI de testeo solo es configurable el parámetro de ethernet-switching *trunk*/*access*. En la interfaz gráfica definitiva de FEDERICA se podrán configurar el resto de parámetros, los cuales son accesibles/modificables por los otros dos módulos (WS y engine).
- CreateSwitch: Introduce un switch en la infraestructura. Este *wizard* está compuesto por dos paneles. En el primero se introduce la información necesaria para la comunicación con el switch y el segundo confirma que se ha creado e iniciado el dispositivo correctamente.

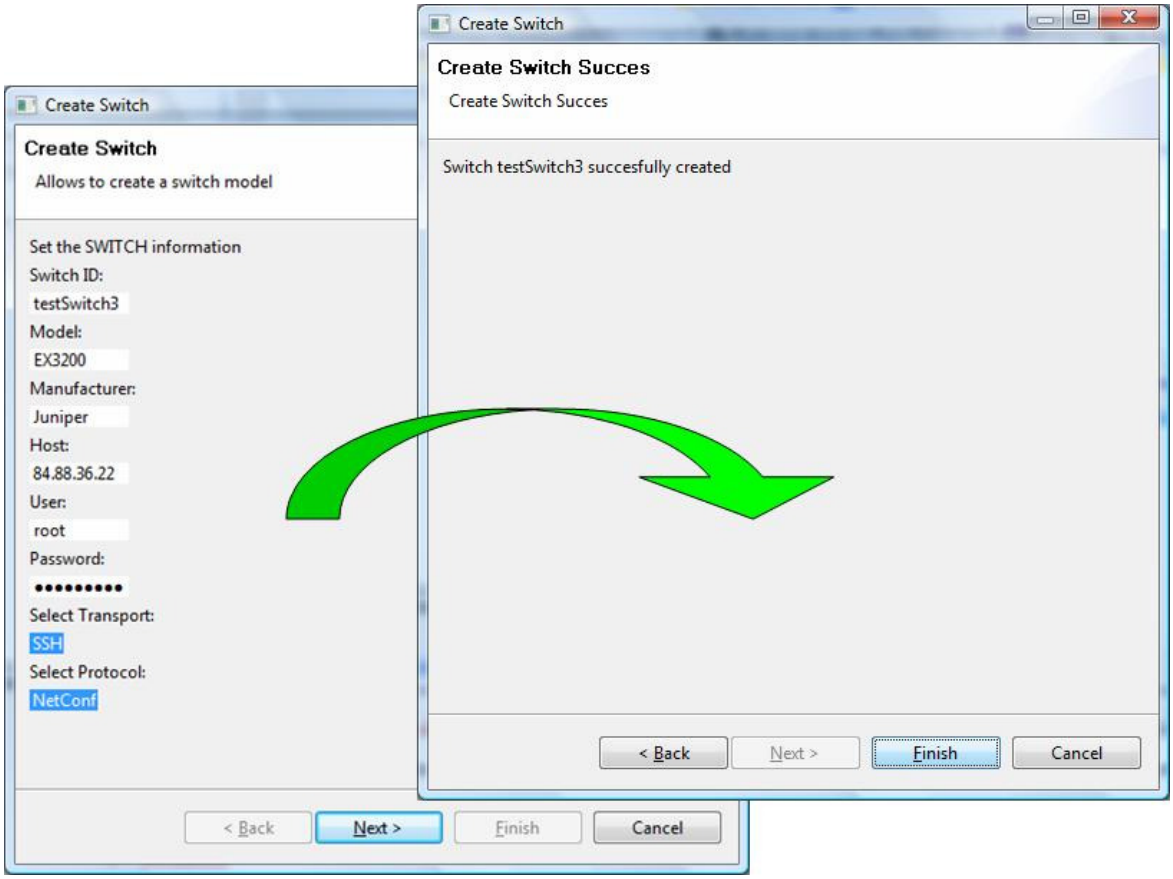


Figura 21: Wizard para introducir un switch a la infraestructura

- CreateVlan: Crea una VLAN en el dispositivo seleccionado en el NavigationView. Está compuesto de dos paneles. En el primero se introduce en nombre e identificador de la Virtual LAN y el segundo confirma si se ha creado con éxito.
- DeleteVlan: Elimina una VLAN de cierto dispositivo. Está compuesto de dos pasos. En el primero se selecciona la VLAN que se desea borrar y el segundo confirma si la operación ha resultado satisfactoria.
- RefreshSwitch: Lo compone un panel en el cual se confirma pulsando "siguiente" que se desea actualizar la configuración del switch. Un segundo panel muestra si la operación se ha llevado a cabo correctamente.
- Virtualize: Crea un recurso que será posible asignar a un usuario de FEDERICA. Para ello el NOC pasa por tres paneles. En el primero se selecciona la interfaz que se desea virtualizar. En el segundo se selecciona la VLAN que se le proporcionará al usuario así como su

identificador (owner). Un tercer panel muestra si la operación se ha realizado con éxito.

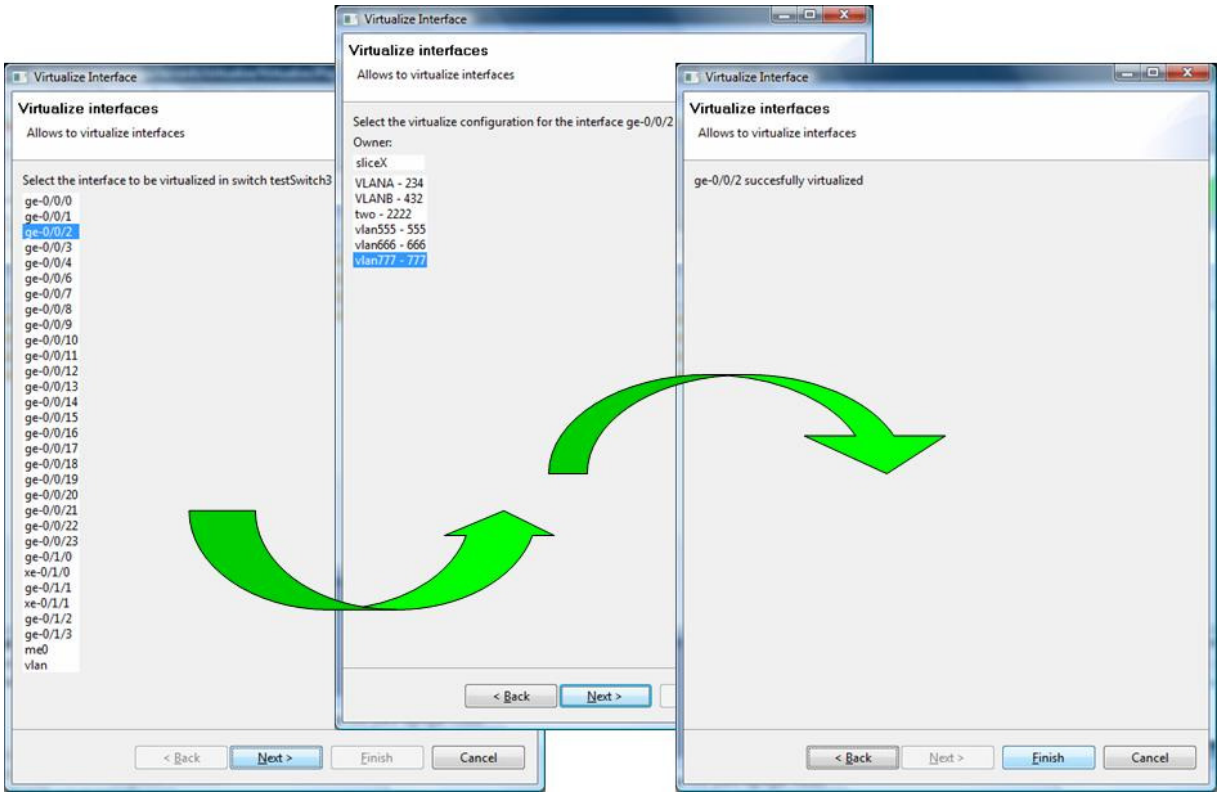


Figura 22: Wizard para virtualizar un puerto

- EthernetSwitchActions: Es la clase que se ocupa de la comunicación con los Ethernet Switch Web Services. Contiene un vector con la configuración de cada uno de los dispositivos así como las acciones necesarias para llevar a cabo las operaciones (createVlan, deleteVlan, updateInterface, etc). Interacciona directamente con la factoría y con las instancias de los web services.
- EthernetResourceActions: Es la clase que se ocupa de la comunicación con los Ethernet Resource Web Services. Contiene un vector con la configuración de cada uno de los recursos que se han virtualizado así como las acciones necesarias para llevar a cabo la visualización en forma de árbol de dichos recursos. Esta información queda representada en la vista UserResourcesView. Básicamente consta de una operación de inicialización que obtiene de los Web Services la lista de recursos existentes en la base de datos. Del mismo modo que lo hace la clase encargada de los switchs, utiliza funciones auxiliares para

recoger e interpretar la información de los *stubs* tal y como se verá en los diagramas de secuencia del siguiente apartado.

La siguiente figura muestra la arquitectura de la GUI a grandes rasgos. En ella aparecen algunos de los módulos anteriormente descritos y se plasma visualmente la interacción y colocación de cada uno de ellos.

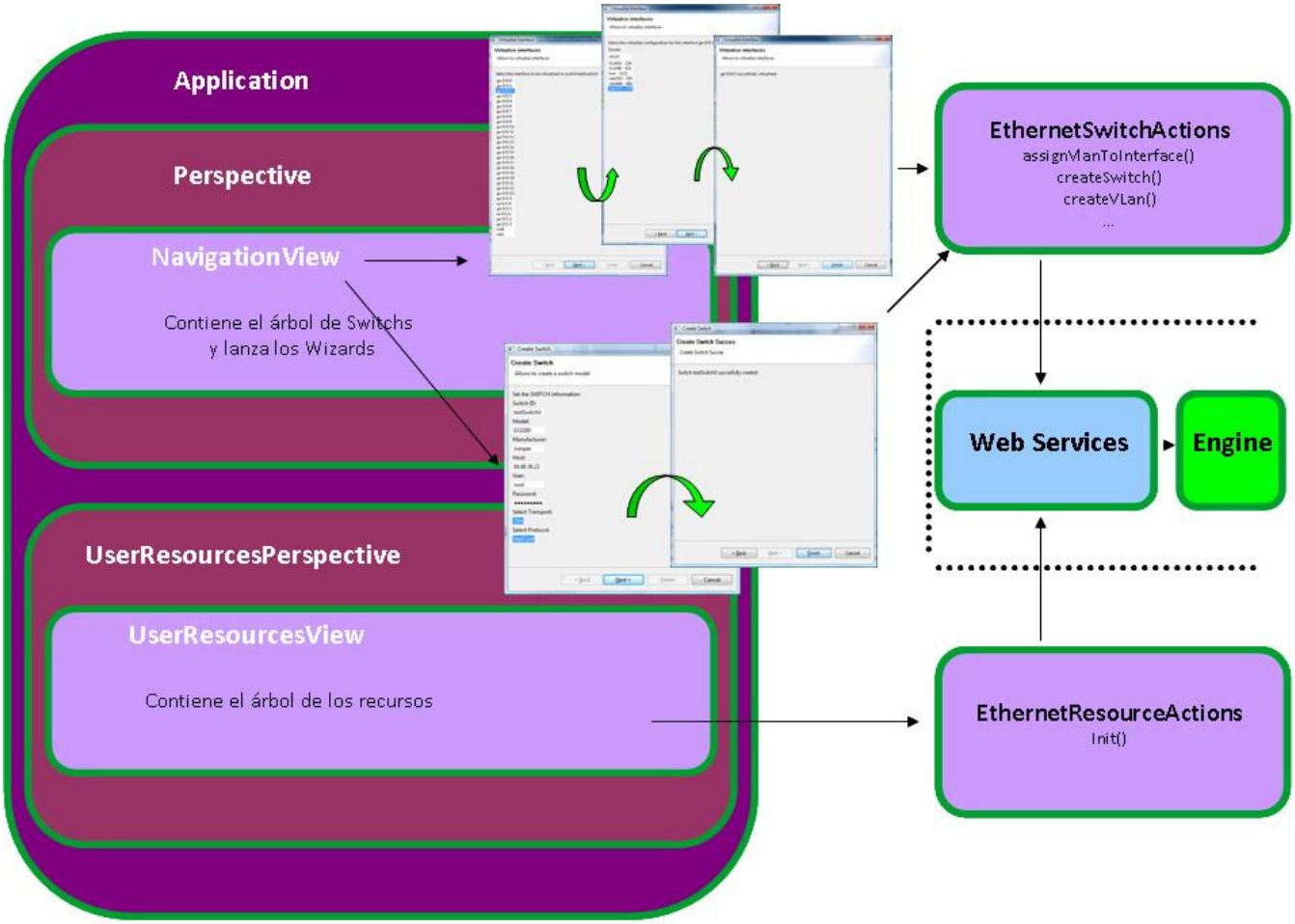


Figura 23: Arquitectura/Componentes básicos de la interfaz gráfica

6.3.6 Interacción de los módulos

Ahora que se han presentado todos los componentes del sistema, se procede a detallar cómo interactúan los módulos entre ellos. Para empezar, se explica cómo es la creación de un nuevo switch y después se detalla cómo se opera sobre él a partir de algunos de los casos de uso que se han explicado a nivel básico. El funcionamiento

interno de la GUI queda en un segundo plano ya que, como se ha dicho anteriormente, se trata de un módulo temporal.

Añadir un switch a la infraestructura:

El administrador de la infraestructura de FEDERICA ha recibido un nuevo dispositivo de nivel dos (switch) y procede a añadirlo al programa de gestión. En el momento en que ejecuta la interfaz gráfica, obtiene la lista con todos los switchs y sus respectivas configuraciones (ver NavigationView – figura 17).

Para ello, el programa ha llamado a la operación init sobre la clase encargada de interactuar con los Web Services (EthernetSwitchActions). El procedimiento queda plasmado en el siguiente diagrama de secuencia:

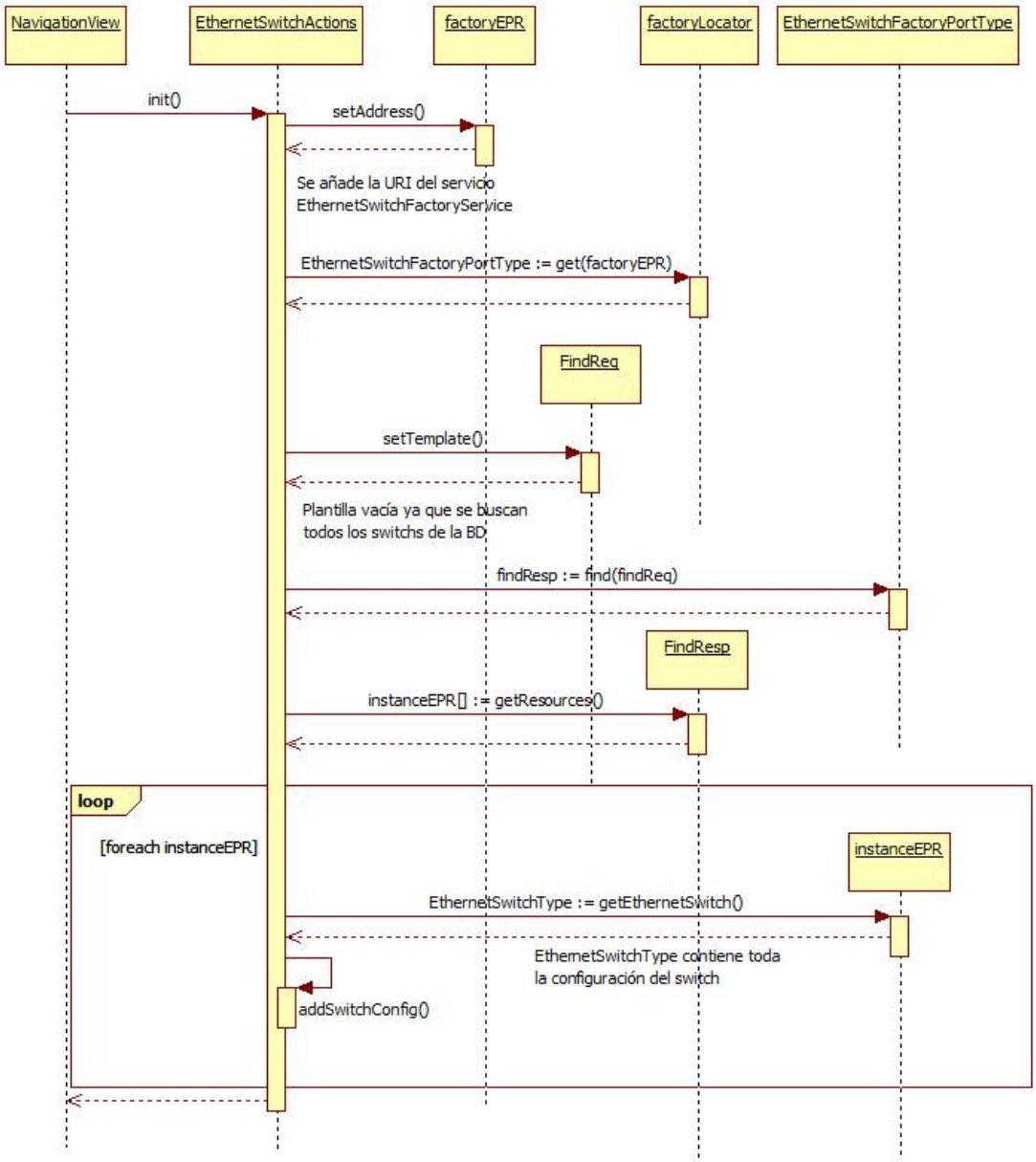


Diagrama UML 8: De secuencia - Inicialización de la GUI

Se puede ver el proceso de comunicación entre la interfaz gráfica con el Ethernet Switch WS. Entre los *stubs* del cliente se encuentra un localizador (factoryLocator) que, mediante la URI del WS devuelve el *portType* (clase que contiene los servicios) de la factoría del WS. Sobre él se realiza la operación de búsqueda (find), cuyo objetivo, en este caso, es encontrar todos los dispositivos de la base de datos. Después de viajar la petición por la red en formato XML tal y como lo describen los ficheros WSDL, los *stubs* interpretan la información y la transforman en un vector de *Endpoint references* dentro de findResp. A modo de recordatorio, un Endpoint reference es la dirección de

un WS-resource. Por cada uno de los elementos del vector, es decir, de cada uno de los switchs de la infraestructura, se obtiene su información y configuración. Esto se lleva a cabo mediante la función getEthernetSwitch, la cual accede al recurso (resource) para obtenerla. Finalmente se añade a la lista de la clase EthernetSwitchsActions. La vista, NavigationView, acude a esta lista para formar el árbol.

Una vez cargada la interfaz gráfica, el administrador hace clic con el botón derecho sobre algún punto de la vista de la GUI para añadir un switch. Al hacerlo, se lanza el *wizard* que le guiará a través de la configuración (ver figura 20). Debe introducir el nombre del switch (identificador único del dispositivo), modelo y fabricante (FEDERICA trabaja con switchs Juniper), host (dirección IP del dispositivo), usuario y contraseña para acceder al aparato así como el tipo de transporte (SSH) y protocolo (NetConf). Al pulsar el botón *next*, EthernetSwitchsActions se comunica de nuevo con la factoría de Ethernet Switch WS. El procedimiento para encontrar su *portType* es el mismo que el de la operación anterior; pero la acción que se realiza sobre éste es la de *create*, plasmada en el siguiente diagrama de secuencia:

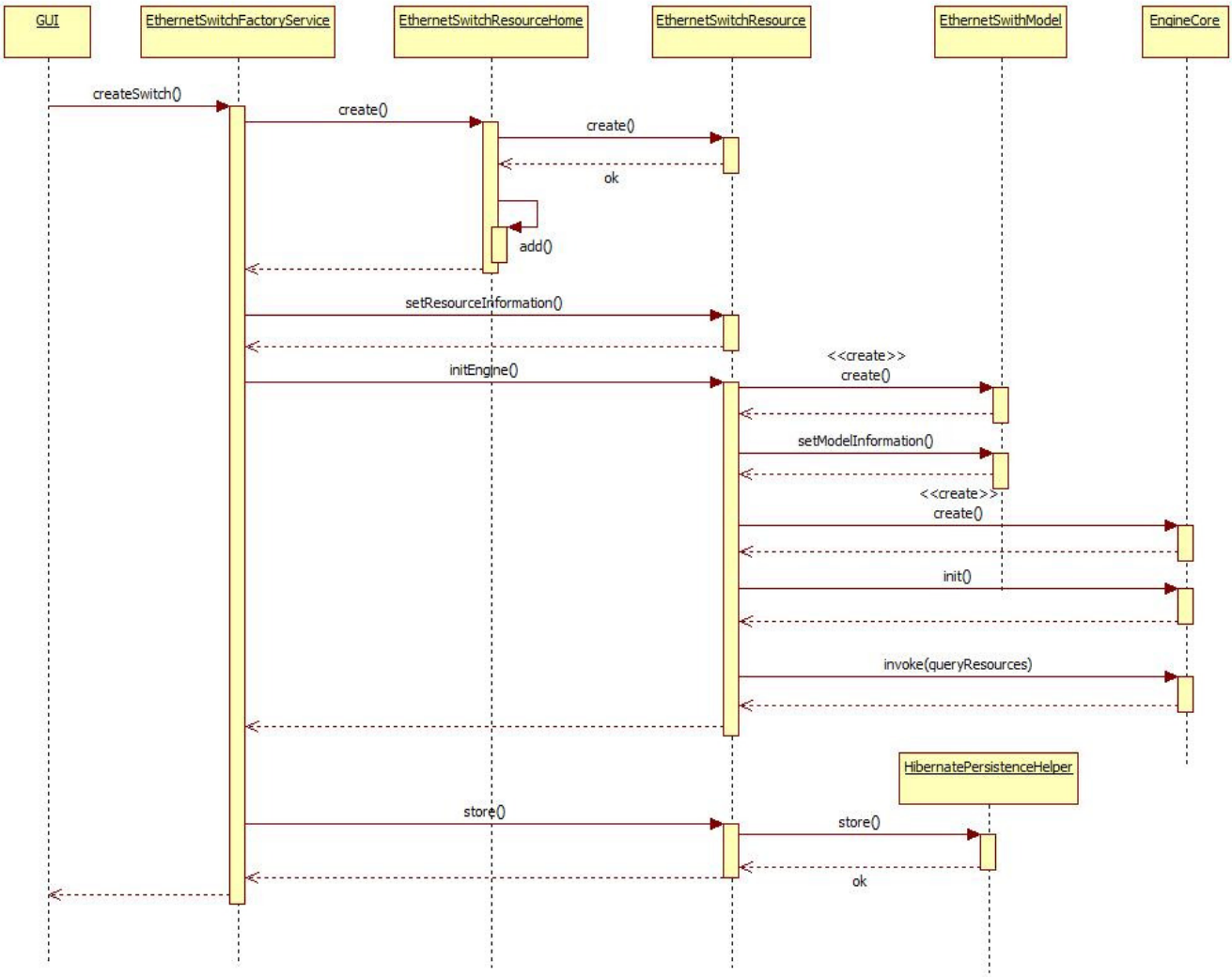


Diagrama UML 9: De secuencia – Crear Switch

La factoría envía el *create* sobre la clase EthernetSwitchResourceHome, la cual hace lo propio sobre el recurso del switch (EthernetSwitchResource). Se añade a la lista de switchs y se procede a rellenar su información (con los datos introducidos en el *wizard*) mediante setResourceInformation de la siguiente manera:

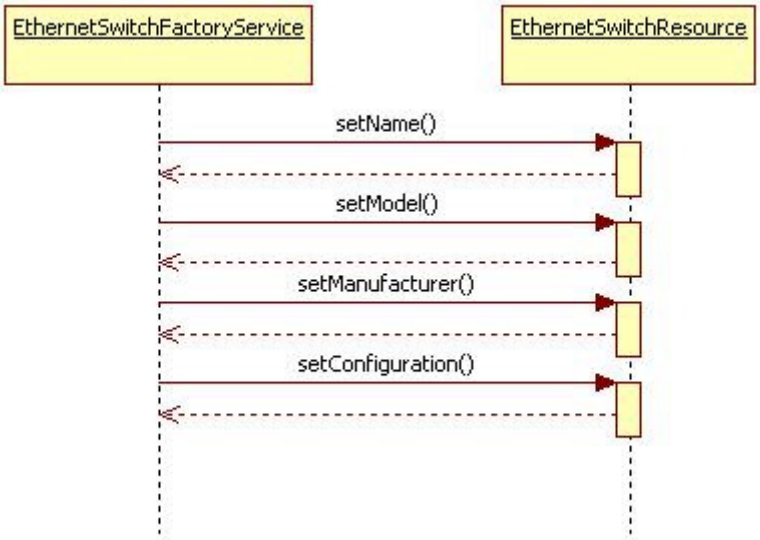


Diagrama UML 10: De secuencia - setResourceInformation

Dentro de la configuración se pasan los datos de transporte, protocolo, usuario, contraseña y dirección IP.

Una vez hecho esto, comienza el proceso de interacción entre los WS y el Engine. Éste último es iniciado (initEngine) mediante el resource del switch (EthernetSwitchResource). El engine actuará como *driver* del dispositivo. Cuando el modelo está creado, se añade la información necesaria para la comunicación:

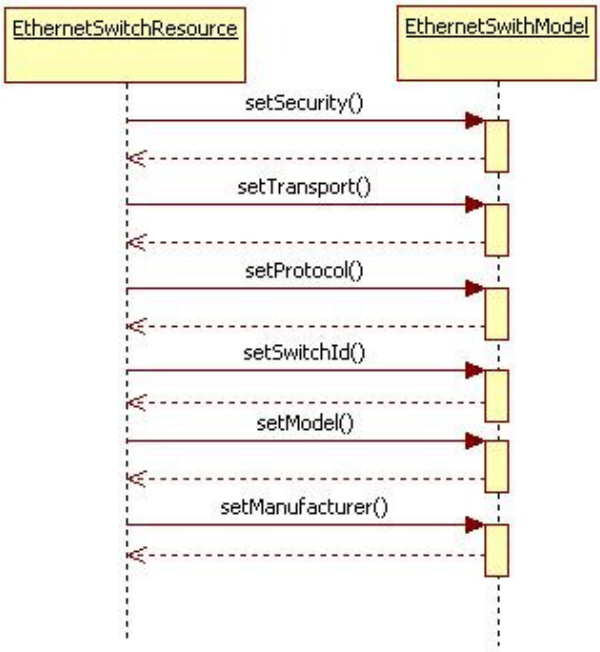


Diagrama UML 11: De secuencia - setModelInformation

En este punto, el EngineCore recibe la operación de init que, a partir del modelo recientemente creado, inicializa las clases encargadas de gestionar el transporte (SSHTransportHandler), el protocolo (NetConfProtocolHandler) y las acciones (ActionHandler). Todo ello lo hace consultando la clase JuniperEX3200, la cual verifica que los datos son compatibles con el dispositivo del fabricante y modelo en cuestión:

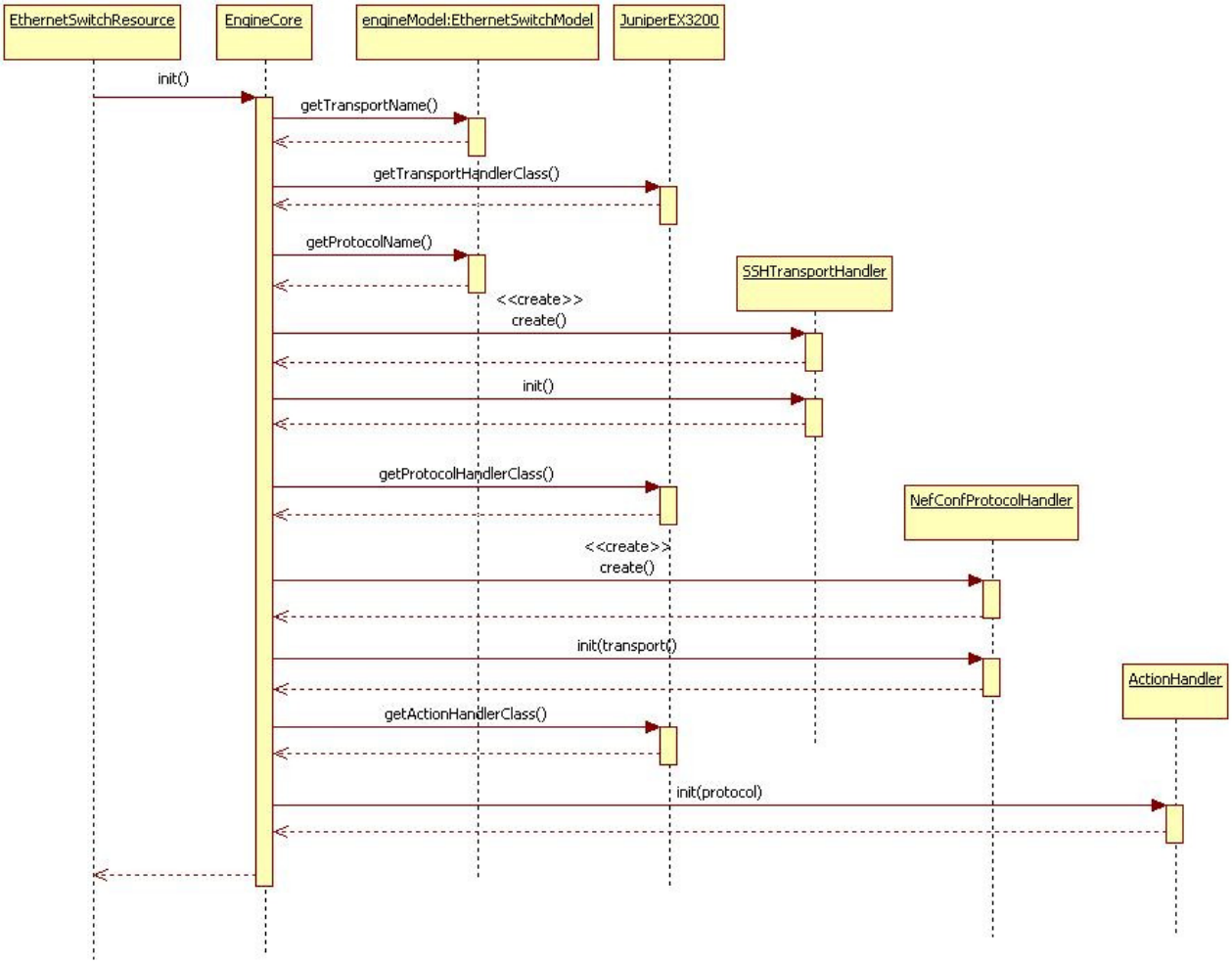


Diagrama UML 12: De secuencia – Inicialización del EngineCore

Ahora el modelo está listo para establecer la primera comunicación con el switch. El objetivo es obtener la configuración de las interfaces y VLANs del dispositivo físico por tal de añadirlas al modelo. A modo de recordatorio, esta información no persiste en la base de datos. El recurso (resource) del Web Service (EthernetSwitchResource) mantiene una instancia del modelo, el cual realiza esta operación cada vez que se activan los WS. Se procede por lo tanto a realizar una acción sobre el dispositivo desde el engine. ActionHandler recibe el *invoke* con el nombre de la acción (queryResources). El siguiente diagrama de secuencia muestra el comportamiento genérico de esta operación. Sea cual sea, el comportamiento es prácticamente el mismo. Lo único que cambia son los comandos y la forma en que se parsea la respuesta del dispositivo.

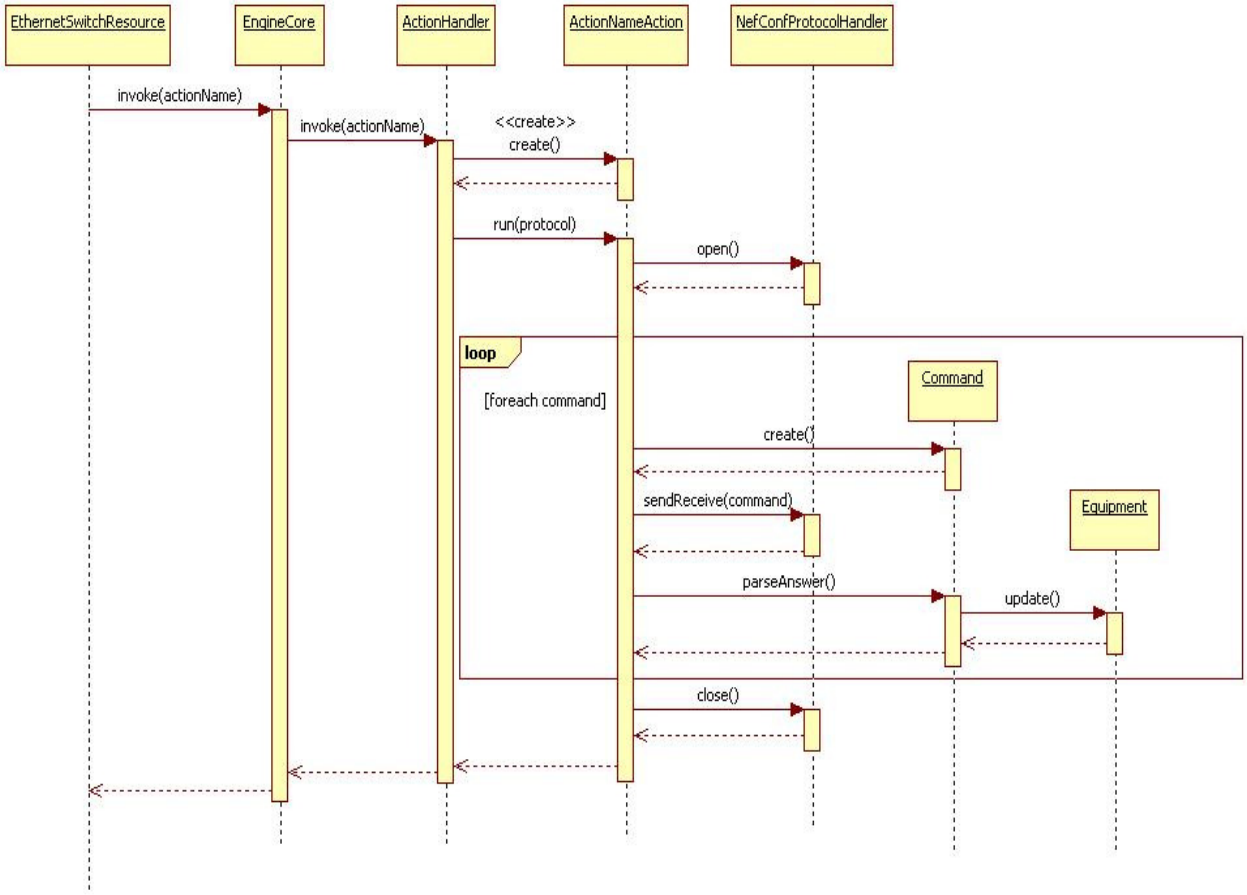


Diagrama UML 13: De secuencia - invoke genérico

EngineCore delega la invocación al ActionHandler, el cual crea la acción específica (en este caso QueryResourcesAction). Esta clase pone en marcha el gestor del protocolo que abre el canal de comunicación mediante el transporte dado. El funcionamiento de las operaciones sobre el transporte y el protocolo (open, close, sendReceive) se han omitido del diagrama de secuencia ya que es algo que ya estaba implementado y se desvía del propósito de la explicación. Una vez hecho esto, se procede a crear el comando de la acción. Se trata de un mensaje NetConf, es decir, un mensaje XML. Para obtener la configuración de puertos y VLANs; el código en el cual se crea el mensaje presenta el siguiente aspecto:

```
[…]

public String getCommand() {
   String CONSULT_HEADER = "<rpc>\n<get-config>\n
    <source>\n<candidate/>\n</source>\n<filter>\n";
   String CONSULT_BOTTOM = "</filter>\n</get-config>\n</rpc>\n";
   String command = CONSULT_HEADER+"<configuration>\n
    <interfaces>\n</interfaces>\n<vlans>\n</vlans>\n
    </configuration>\n"+CONSULT_BOTTOM;
   return command;
}

[…]
```

Como se ha dicho anteriormente, una acción puede estar compuesta de varios comandos; por esa razón en el diagrama de secuencia genérico aparece la condición de bucle (loop). En esta aplicación, sin embargo, a cada acción le corresponde un único comando. Por lo tanto, el comando NetConf se envía al switch a través de SSH y se obtiene una respuesta con la configuración del switch. El mensaje presenta el siguiente aspecto:

```
[…]
<interfaces>
   <interface>
      <name>ge-0/0/0</name>
      <unit>
         <name>0</name>
         <description>interface description</description>
         <family>
            <ethernet-switching>
               <port-mode>trunk</port-mode>
            </ethernet-switching>
         </family>
      </unit>
   </interface>
      <interface>
         <name>ge-0/0/1</name>
         <unit>
            <name>0</name>
            <family>
               <ethernet-switching\>
            </family>
         </unit>
      </interface>
[…]
</interfaces>
<vlans>
   <vlan>
      <name>VLANB</name>
      <description>Test VLAN</description>
      <vlan-id>432</vlan-id>
      <interface>
         <name>ge-0/0/0.0</name>
      </interface>
      <interface>
         <name>ge-0/0/11.0</name>
      </interface>
      <interface>
         <name>ge-0/0/20.0</name>
      </interface>
   </vlan>
   <vlan>
      <name>test23</name>
      <description>Vlan Test Demo Berlin</description>
      <vlan-id>23</vlan-id>
   </vlan>
[…]
</vlans>
[…]
```

Para parsear el mensaje (parseAnswer), el comando delega el trabajo a la clase ObjectMapper descrita anteriormente. Esta clase interpreta las etiquetas XML obteniendo las interfaces y VLANs junto con sus respectivas configuraciones y las coloca en el Equipment del modelo. Para ello, el *parser* desecha aquellas secciones del

mensaje que tiene que ver con el establecimiento de la conexión/desconexión y se centra únicamente en todo aquello rodeado por las etiquetas “interfaces” y “vlans mostradas en el cuadro anterior.

Una vez hecho esto, el *resource* del Web Service ya tiene el estado inicial del switch. Con lo cual, la factoría procede a guardar en la base de datos la información de dicho recurso que sea persistente. Para ello, se utiliza la clase HibernatePersistenceHelper.

Finalmente, el control vuelve a la GUI, que actualiza el árbol de dispositivos añadiendo el recientemente creado.

Crear una VLAN:

Ahora que el nuevo dispositivo está integrado en el sistema, se puede operar sobre él. Por ejemplo, si se quiere añadir una VLAN, se clica con el botón derecho sobre el nombre del switch y se selecciona *create vlan*. Esto lanza el *wizard* homónimo que guía al administrador en la configuración. Esto es: el nombre de la VLAN y su identificador numérico. La GUI busca el EthernetSwitch WS a partir de su factoría. El procedimiento es el mismo que el del diagrama UML 6 con la diferencia de que el *find* realiza la búsqueda de un único dispositivo. Una vez se obtiene la instancia del servicio, se lanza sobre él la operación *invoke* con su correspondiente mensaje *invokeReq* que contiene los datos introducidos por el usuario. El siguiente diagrama de secuencia muestra el comportamiento del servicio a partir de este punto. Se obtiene el *resource* del Web Service y de éste su engineModel y se parsea el mensaje *invokeReq*. Ahora, dependiendo de la operación que se lleve a cabo, se procederá a rellenar de una manera u otra el equipamiento del modelo. En este caso, debido a que se trata de crear una VLAN, se procede a rellenar los campos de VLANType del engine y colocarlos en *requestedVlans*. En el diagrama se observa cómo se lleva a cabo la operación y cómo el WS y el engine están programados para, además de crear la virtual LAN, añadir a la vez las interfaces que se deseen, viendo de esta manera que la limitación es de la interfaz gráfica que, como se ha dicho anteriormente, es de carácter temporal. Una vez hecho esto, se envía el *invoke* al engine, el cual sigue el mismo procedimiento que el diagrama UML 11 (invoke genérico).

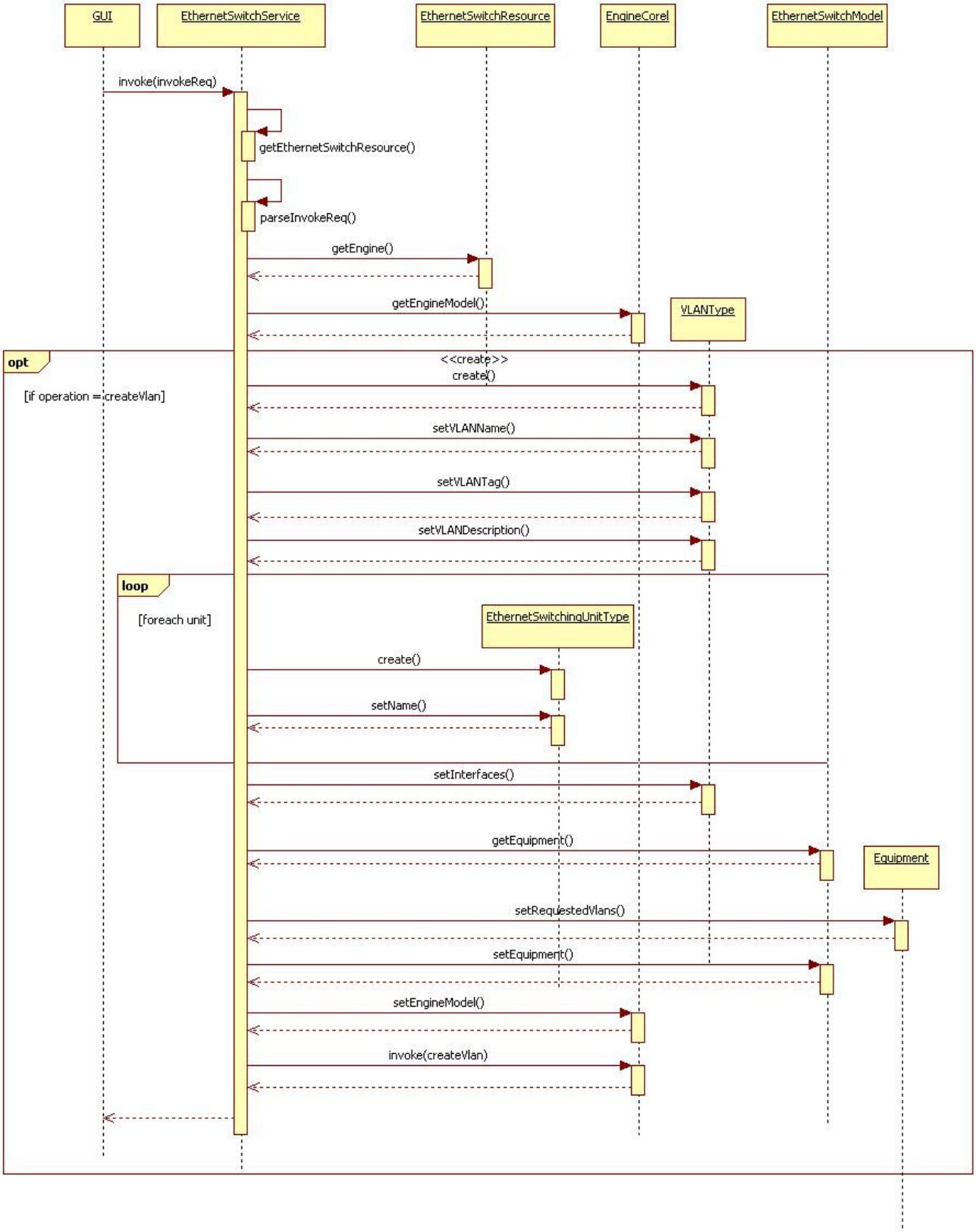


Diagrama UML 14: De secuencia – invoke con la operación de crear una VLAN

El resto de operaciones que actúan sobre el switch siguen un procedimiento muy similar, ya que todas ellas funcionan bajo la operación de *invoke*.

Virtualización de un puerto:

Por lo tanto, y por último, se detalla la operación de virtualización de un puerto, la cual crea un EthernetResource que estará a disposición de los usuarios de FEDERICA. Es una comunicación entre ambos Web Services y la interfaz gráfica. El administrador debe seguir el camino que le indica el *wizard* seleccionando la interfaz que desea publicar e introduciendo el nombre del usuario final al que se le asigna y su VLAN. Al pulsar en "siguiente" comienza el proceso de virtualización, representado en el siguiente diagrama de secuencia. Se puede ver cómo la factoría busca el switch sobre el que se realiza la petición y, una vez obtenida la instancia, se llama a la operación de *virtualize* sobre la instancia del servicio.

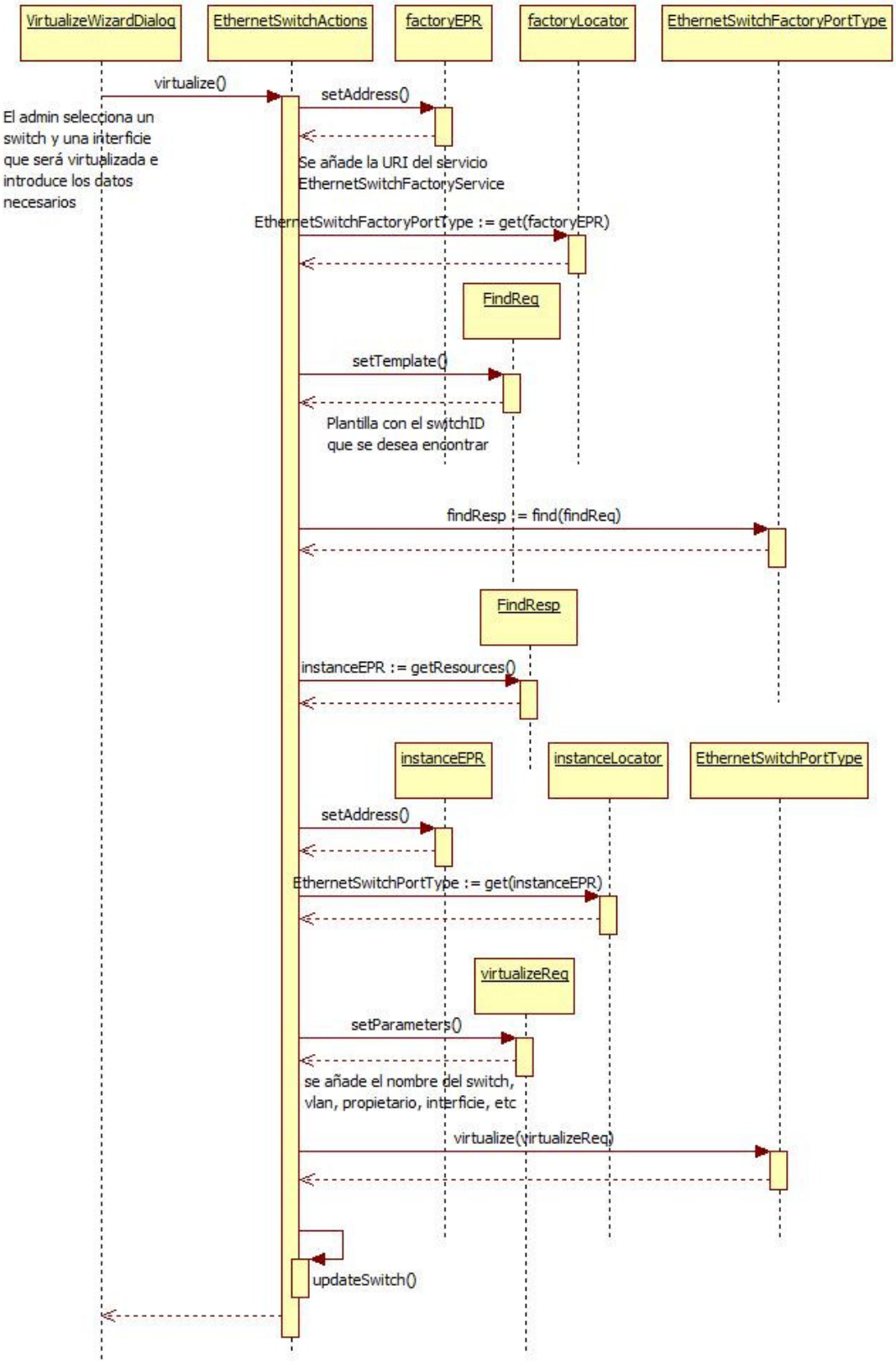


Diagrama UML 15: De secuencia – virtualize sobre EthernetSwitchActions

Se cogen los parámetros del mensaje de virtualización y se traspasan al mensaje para crear el nuevo recurso (createReq). El procedimiento de comunicación entre los Web Services es muy similar al descrito cuando se introducía un switch en la infraestructura. El siguiente diagrama de secuencia muestra el procedimiento.

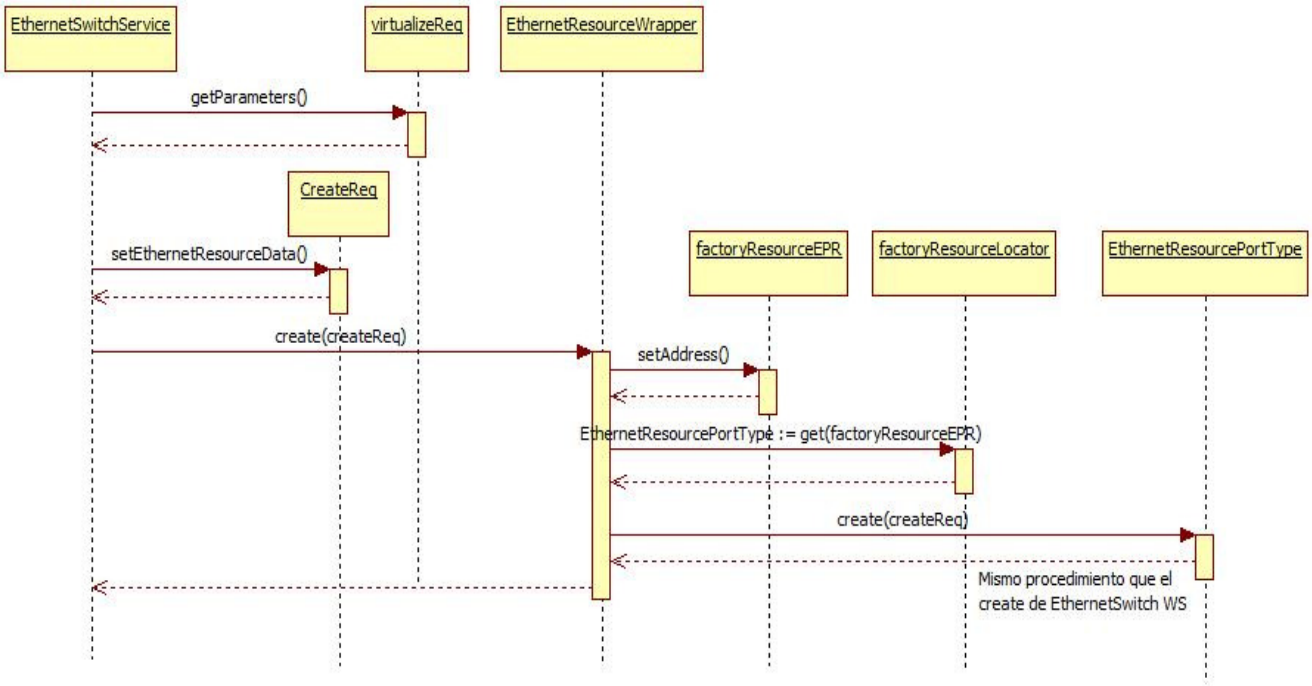


Diagrama UML 16: De secuencia – virtualize sobre EthernetSwitchService

Una vez vista la manera en que interactúan todos los módulos del sistema, se cierra la descripción de las tareas. En el siguiente punto se muestran las pruebas que se han realizado para comprobar el buen funcionamiento de la aplicación.

6.4 Tests

El presente punto está estructurado, tal y como se viene haciendo en las secciones anteriores, por módulos. Primero se muestran los tests realizados sobre el engine. Una vez comprobado que todo funciona de la manera deseada, se procede a la implementación de los Web Services y su correspondiente juego de pruebas. Para la elaboración de la GUI, primero se integró directamente con el engine para poder llevar a cabo las demostraciones necesarias para exponer el trabajo realizado al resto de los

partners de FEDERICA. Posteriormente se integraron los WS a dicha interfaz gráfica obteniendo así la aplicación que se presenta en este documento, la cual no es definitiva dentro del cómputo global de FEDERICA. También se incluyen los tests realizados para comprobar la conectividad del switch con las máquinas virtuales que funcionarán en FEDERICA. Finalmente se muestran los preparativos que se llevaron a cabo para la realización de una demo que tuvo lugar por tal de mostrar el trabajo realizado al resto de *partners* del proyecto.

6.4.1 Tests del Engine

Una vez implementado el modelo del dispositivo en el engine, se realizaron varias pruebas mediante JUnit sobre el switch del que se dispone con este fin. No está integrado dentro de la infraestructura de FEDERICA, si no que está ubicado en el Campus Nord. Se trata de un Juniper EX-3200, de esta manera queda asegurado que el comportamiento será el mismo que dentro de FEDERICA ya que el modelo del aparato es el mismo y se accede remotamente a través de una dirección IP pública.

El banco de pruebas consiste en una serie de funciones, utilizando JUnit, que retornan una respuesta positiva si el resultado es el esperado. Cada una de las funciones se encarga de una funcionalidad específica. Además, para comprobar que los resultados sobre el switch se han realizado satisfactoriamente, se implementó una función que mostraba la configuración actual presente en el modelo del engine que, posteriormente, se contrastaba con la configuración real del dispositivo. En la siguiente tabla se muestran las funciones realizadas:

Tabla 1: Acciones de testeo sobre el engine

OPERACIÓN	*OBJETIVO*	*ELEMENTOS*	*RESULTADO*
connection	Realizar una conexión con el dispositivo	Engine-Switch	OK
createLogicalInterface	Crear una interfaz lógica	Engine-Switch	OK
deleteLogicalInterface	Borrar una interfaz lógica	Engine-Switch	OK
addInterfaceToVlan	Añadir una interfaz a una VLAN	Engine-Switch	OK

addManyInterfaces	Añadir varias interfaces a una VLAN	Engine-Switch	OK
createVlans	Crear una o varias VLANs	Engine-Switch	OK
updateVlans	Modificar una o varias VLANs	Engine-Switch	OK
deleteVlans	Destruir una o varias VLANs	Engine-Switch	OK
queryResources	Obtener la configuración del dispositivo	Engine-Switch	OK
printModel	Imprimir la configuración del dispositivo	Engine	OK

6.4.2 Tests de los Web Services

Una vez comprobado el buen funcionamiento del engine e implementados los Web Services, se creó la clase de testeo de los WS también mediante JUnit de la misma manera. La tabla siguiente muestra los resultados obtenidos de las pruebas realizadas sobre EthernetSwitch Web Services. EWS hace referencia a estos últimos WS mientras que RWS hace referencia a los EthernetResource Web Services.

Tabla 2: Acciones de testeo sobre los Web Services

OPERACIÓN	*OBJETIVO*	*ELEMENTOS*	*RESULTADO*
createSwitch	Añadir un switch a la infraestructura	EWS-Engine-Switch-BD	OK
findSwitch	Buscar un switch de la base de datos	EWS-Engine-BD	OK
refresh	Actualizar la información del modelo forzando conexión	EWS-Engine-Switch	OK
invoke	Realiza mediante WS los comandos sobre el switch	EWS-Engine-Switch	OK
virtualize	Crea un recurso	EWS-RWS-BD	OK
unvirtualize	Destruye un recurso	EWS-RWS-BD	OK

6.4.3 Tests de la GUI

Se llevó a cabo en dos etapas. En la primera se comunica directamente con el engine. En la segunda se integran los Web Services. Por lo tanto, el testeo de este módulo corre en paralelo con los de los dos anteriores.

En la siguiente tabla se muestran las operaciones que se han testeado en la primera etapa desde la interfaz gráfica que, como se ha comentado en puntos anteriores, son menos que las soportadas por el engine. Son las mínimas por tal de ofrecer conectividad a un dispositivo de una red a nivel 2. En esta primera etapa todavía no se considera el concepto de persistencia ni gestión de varios switchs, con lo cual sólo se puede configurar un dispositivo proporcionando directamente su dirección IP.

Tabla 3: Acciones de testeo sobre la interfaz gráfica

OPERACIÓN	*OBJETIVO*	*ELEMENTOS*	*RESULTADO*
createVlan	Crea una VLAN en el dispositivo	GUI-Engine-Switch	OK
configurePort	Configura un puerto lógico	GUI-Engine-Switch	OK
assignVlanToIface	Asigna una VLAN a un puerto lógico	GUI-Engine-Switch	OK
refresh	Actualiza la configuración del modelo del switch	GUI-Engine-Switch	OK

En la segunda etapa de la construcción de la interfaz gráfica se han añadido los Web Services. Las pruebas son básicamente las mismas que las realizadas directamente sobre los servicios pero, en lugar de comprobarlo mediante JUnit, se han llevado a cabo utilizando la GUI. Con lo cual no se repite la tabla de pruebas, pues es prácticamente la misma. Tan sólo destacar que las operaciones de *invoke* no aprovechan la eficiencia de la que son capaces tanto los Web Services como el engine.

6.4.4 Tests de las VLANs sobre las máquinas virtuales

FEDERICA proporcionará máquinas virtuales (VMs), alojadas en servidores (VMSs), a los usuarios finales para operar con sus redes. Cada VMS posee una serie de tarjetas de red a las que se pueden conectar varias VMs a través de un switch virtual.

Un switch virtual (vSwitch) es un dispositivo de red virtual cuyo compartimiento es análogo al de uno físico. Sin embargo, estos vSwitchs tienen algunas limitaciones respecto a los físicos.

Para entender las pruebas realizadas en este apartado, es necesario introducir el concepto de *port group*, cuyo objetivo es conectar servicios de red a las máquinas virtuales y definir cómo se realiza esta conexión a través del vSwitch. Los *port groups* agrupan varios puertos bajo una misma configuración, proporcionando de esta manera la conectividad de una VM en una red etiquetada. Además, especifican otras opciones como el ancho de banda y otras políticas para cada miembro del grupo.

Con estos conceptos claros, se presentan las pruebas relazadas con el objetivo de establecer un comportamiento de red próximo al que se dará dentro de la infraestructura de FEDERICA.

Test 1:

En esta prueba se dispone de dos máquinas virtuales, cada uno de ellas conectada a una tarjeta de red física a través de un vSwitch. Cada VM pertenece a un *port group* ya que hay una por vSwitch. Éstos últimos están configurados para aceptar paquetes con cualquier identificador de VLAN. Las interfaces ge-0/0/14.0 y ge-0/0/15.0 del switch físico están configuradas como ethernet-switching y ambas son miembros de la VLAN “testVlan” cuyo identificador (etiqueta o *tag*) es 1234. Cada VM tiene una IP de la misma red configurada.

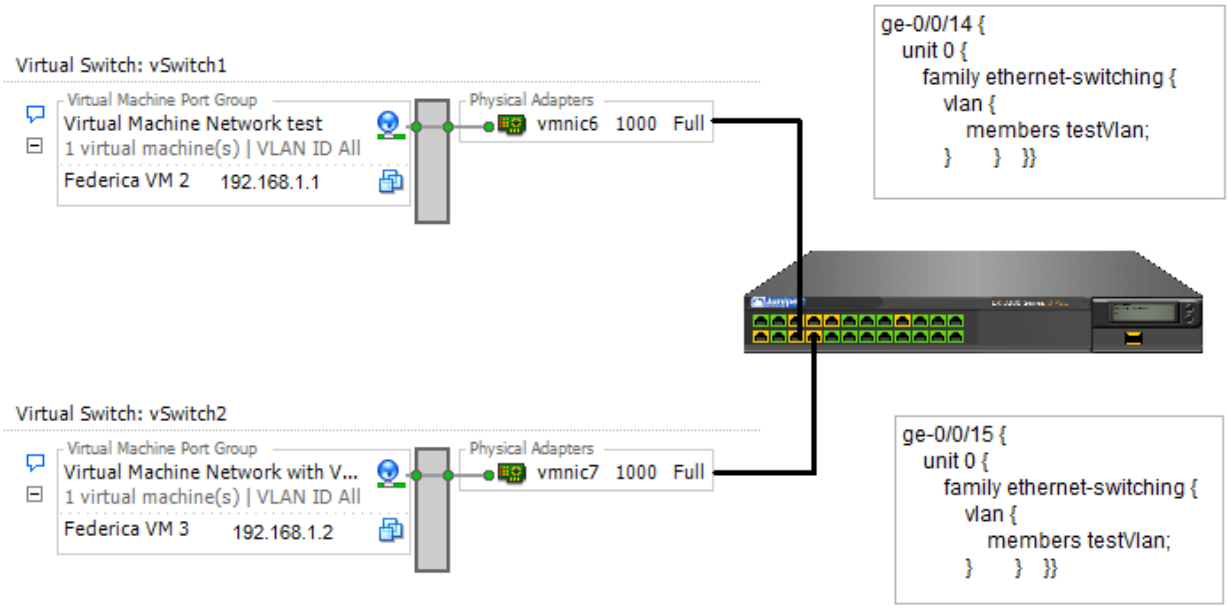


Figura 24: Topología y configuración del test 1 de las VLANs sobre máquinas virtuales

El resultado obtenido, a través de una serie de *pings* es que ambas VMs se comunican correctamente. Esto se debe a que no hay etiquetado alguno ya que los vSwitchs permiten cualquier paquete y las interfaces del dispositivo físico no etiquetan ya que están en modo acceso.

Test 2:

Con la misma topología, en esta prueba se configuran los vSwitchs de tal manera que solo aceptan paquetes etiquetados con la VLAN 1234. Las interfaces ge-0/0/14.0 y ge-0/0/15.0 del switch Juniper siguen perteneciendo a “vlanTest” (con identificador 1234) pero ahora están configuradas como *trunk*.

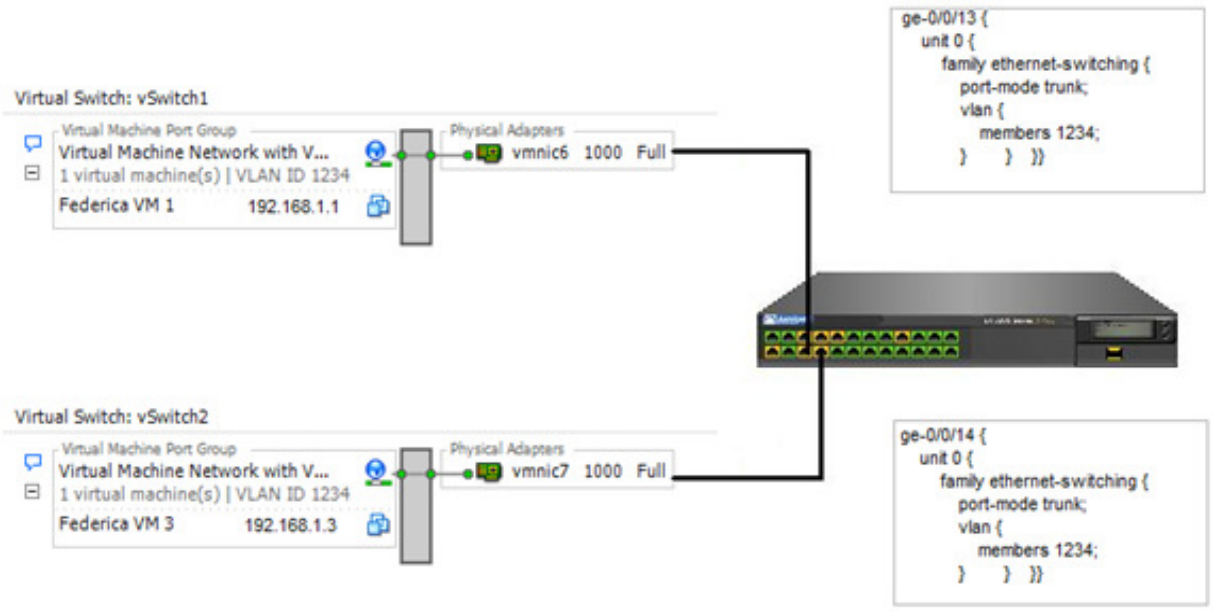


Figura 25: Topología y configuración del test 2 de las VLANs sobre máquinas virtuales

El resultado obtenido, a través de una serie de *pings* es que ambas VMs se comunican correctamente. En este caso sí que existe etiquetado, tanto en los dispositivos virtuales como en el físico. Si los puertos no estuvieran configurados como *trunk* no se interpretarían las etiquetas de los paquetes en el switch y, por consiguiente, se descartarían y el *ping* no llegaría a su destino.

Test 3:

El objetivo de este test es asegurarse de que dos VMs conectadas a la misma interfaz física, cada una de ellas en una VLAN diferente, no sean capaces de comunicarse. En el vSwitch, cada *port group*, es decir, cada VM, pertenece a una VLAN; una con etiqueta 1234 y la otra con etiqueta 1122. En el dispositivo físico, la interfaz ge-0/0/14.0 se configura como *trunk* y pertenece a ambas VLANs. El escenario muestra el siguiente aspecto:

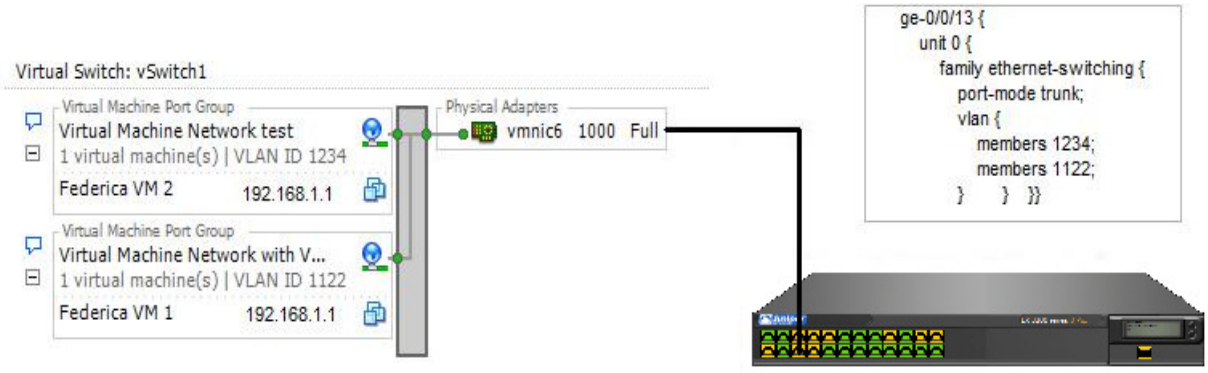


Figura 26: Topología y configuración del test 3 de las VLANs sobre máquinas virtuales

Tal y como se esperaba, no hay comunicación entre las máquinas virtuales implicadas ya que pertenecen a LANs virtuales diferentes.

Test 4:

Este test es una combinación de los dos anteriores. Se tiene, por un lado, dos VMs conectadas a la misma tarjeta de red física y, por ende, al mismo vSwitch. Ambas con la misma IP pero pertenecientes a VLANs diferentes (1234 y 1122). Por otro lado, tenemos una máquina virtual conectada a otra tarjeta que forma parte de la VLAN 1234. Su IP está en el mismo rango que las dos anteriores.

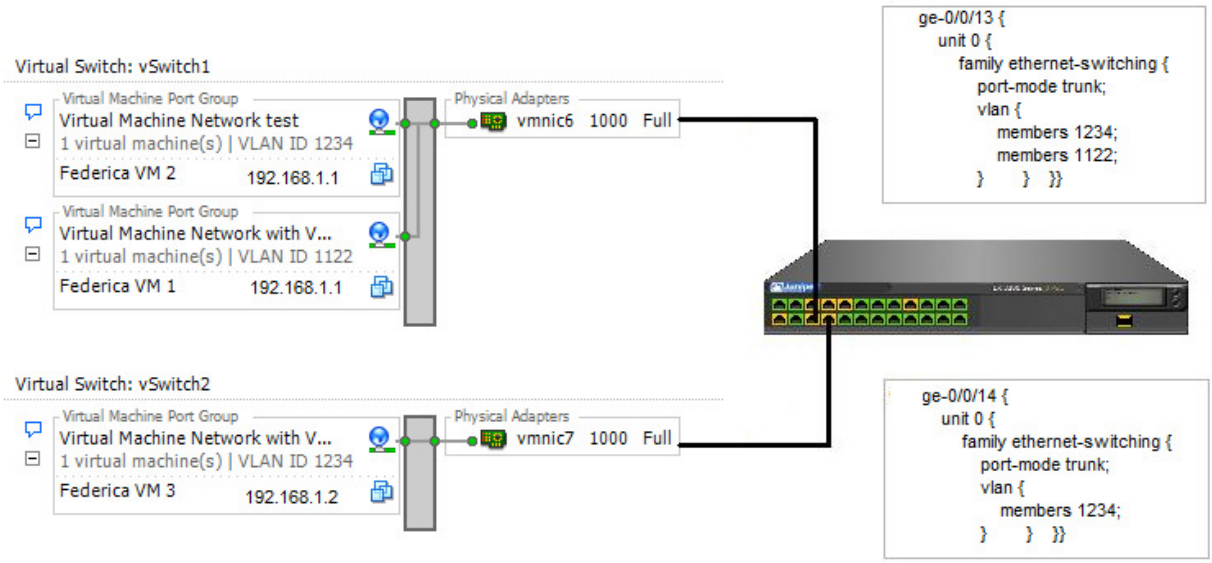


Figura 27: Topología y configuración del test 4 de las VLANs sobre máquinas virtuales

En este caso es posible hacer *ping* (con lo cual existe comunicación) entre las máquinas virtuales que pertenecen a la misma VLAN; es decir, entre “Federica VM 2” y “Federica VM 3”. Dado que “Federica VM 1” pertenece a otra virtual LAN diferente, no se da cuenta del tráfico que fluye entre las otras dos máquinas.

Test 5:

En esta última prueba, se introduce un cambio importante por tal de intentar encontrar una solución al hecho de que los vSwitchs no soportan Q-in-Q (ver anexo B) introduciendo etiquetado VLAN vía software. Con esta meta, se configuran los vSwitchs para que no etiqueten VLANs (del mismo modo que se ha hecho en el test número 1). Serán las propias VMs las que, mediante el sistema operativo, hagan el etiquetado/desetiquetado de los paquetes. La distribución Ubuntu tiene una herramienta de configuración de sus interfaces capaz de manejar VLANs.

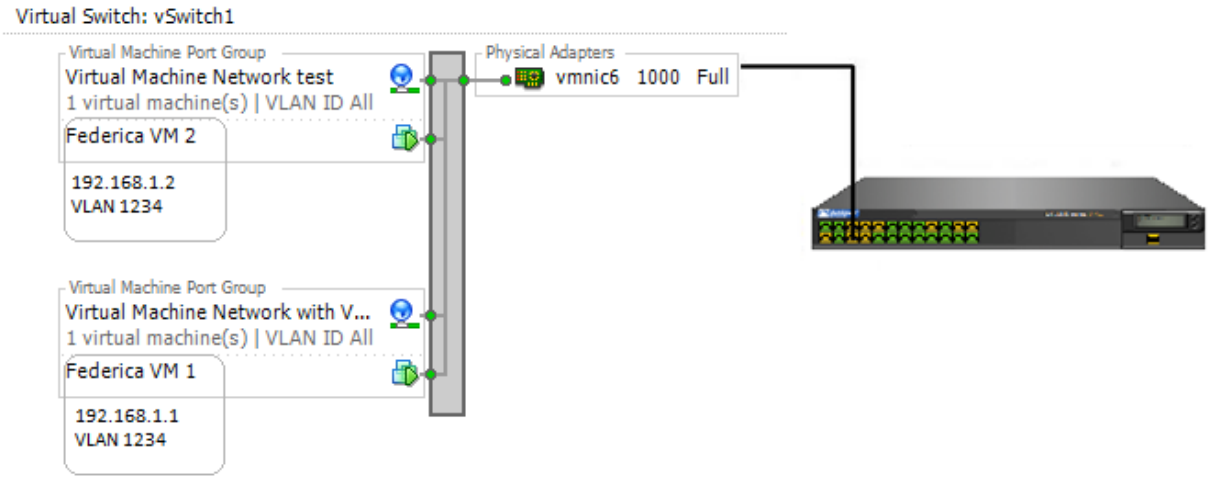


Figura 28: Topología y configuración del test 5 de las VLANs sobre máquinas virtuales

La configuración de las interfaces vía sofware presenta el siguiente aspecto:

```
#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
```

Las pruebas de este tipo no han dado resultados satisfactorios ya que hay problemas con los vSwitchs; no tiene actualmente las capacidades necesarias para implantar Q-in-Q. El switch físico no recibe ningún tráfico en esta situación. Esto obliga a tener que estudiar otras soluciones diferentes.

6.4.5 Demo

Durante el transcurso del proyecto FEDERICA se realizó una demostración en la que se creó un *slice* por tal de que el resto de *partners* vieran el trabajo realizado hasta el momento. El proyecto que aquí se presenta participó en la configuración del switch que aparece en la pequeña infraestructura que se construyó para tal evento, la cual presenta el siguiente aspecto:

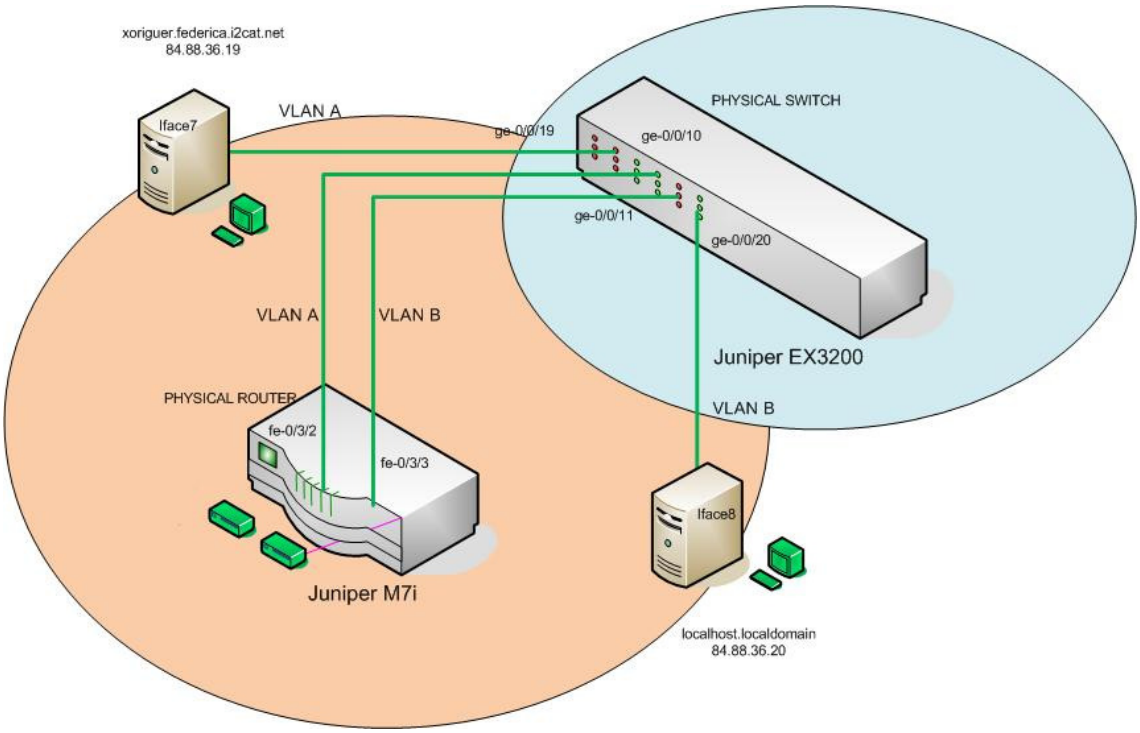


Figura 29: Configuración física de la demo de creación de un *slice*

Consta de dos servidores, en los cuales se crean sendas máquinas virtuales y de un router que se conecta, a través de un switch, con ambos servidores.

El objetivo es crear una red virtual en la que cada máquina se conecte con un router y éstos queden comunicados entre sí mediante un túnel.

Para ello se tuvo que configurar el switch, el cual es totalmente transparente en el *slice*, por tal de dirigir el tráfico a nivel dos de la manera deseada. Para ello se integró la interfaz gráfica en su primera etapa (todavía sin los WS y con las funciones básicas mostradas en la tabla correspondiente) con otra que configura las máquinas virtuales (VMs). Esta aplicación, junto con MANTICORE (encargada de configurar y virtualizar routers), permite habilitar una red virtual configurable a nivel tres para el usuario final. El procedimiento de configuración del NOC consta de los siguientes pasos (remarcados en verde en la figura anterior):

- Crear una máquina virtual en cada servidor y configurar VNC para acceder a ellas.
- Crear dos routers lógicos en el físico con dos interfaces lógicas cada uno.
- Crear dos *trunk* VLANs.

El resultado es el siguiente:

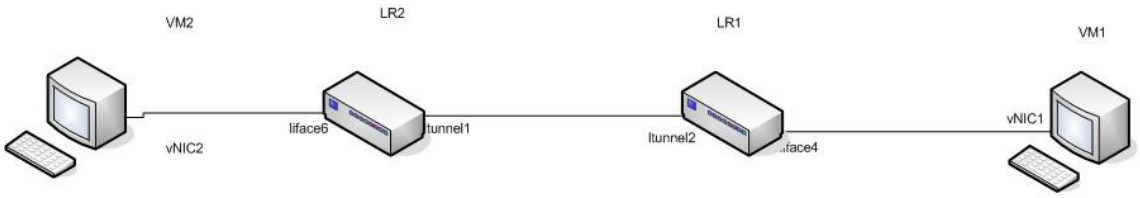


Figura 30: Configuración lógica de la demo de creación de un *slice*

Sobre esta red lógica, el usuario final puede configurar a su voluntad cada interfaz, tanto de las máquinas como de los routers como si se tratara de una red normal, totalmente ajeno a cómo es realmente la infraestructura.

Por tal de preparar la demo, se realizaron una serie de tests tanto con JUnit como con la propia GUI para asegurar el correcto funcionamiento y configuración del switch. Las siguientes líneas de código realizan secuencialmente las operaciones que se tuvieron que realizar sobre el switch por tal de configurar las dos VLANs.

```
@Test
public void testDemo() {
  initEthernetSwitch(); //Set connection values
  queryResources(); //Get switch configuration
  createVlan("VLANA", "234", "Demo Vlan A Description");
  createVlan("VLANB", "432", "Demo Vlan B Description");
  assignVlanToIface("ge-0/0/10", "0", "VLANA");
  assignVlanToIface("ge-0/0/19", "0", "VLANA");
  assignVlanToIface("ge-0/0/20", "0", "VLANB");
  assignVlanToIface("ge-0/0/11", "0", "VLANB");
  configurePort("ge-0/0/10", "0", "Iface 10 description", true);
  configurePort("ge-0/0/19", "0", "Iface 19 description", false);
  configurePort("ge-0/0/20", "0", "Iface 20 description", false);
  configurePort("ge-0/0/11", "0", "Iface 11 description", true);
}
```

Tras repetir reiteradamente los tests, contrastando a cada paso la configuración real del dispositivo mediante SSH, la demostración finalmente se llevó a cabo satisfactoriamente.

7 Trabajo por realizar

FEDERICA se acerca al ecuador de su vida y la parte que aquí se ha presentado debe evolucionar durante la segunda mitad del proyecto. La aplicación presentada se encuentra en una fase estable; no obstante queda bastante trabajo por realizar. A continuación se exponen algunas funcionalidades y capacidades que se deben incluir en los meses futuros:

- Concepto de red: Con todo lo implementado actualmente, la aplicación no goza de consciencia de red. Es decir, es posible la configuración de los dispositivos individualmente; uno a uno. Además es capaz de ofrecer una serie de recursos (interfaces) a los usuarios y mantenerlos almacenados para poder ser utilizados en un momento dado. Por lo tanto, queda por implementar un servicio que obtenga un conjunto de estos recursos y forme una red con ellos. Una red Ethernet. No está implementado todavía ya que, a nivel global del proyecto FEDERICA, está por ver si definitivamente se proporcionará al posible investigador capacidades de configuración a nivel dos.

- Q-in-Q: Se han llevado estudios al respecto durante todo este tiempo de elaboración del proyecto, pero todavía no se ha implementado nada. No obstante debe considerarse seriamente, pues es un requisito para poder proporcionar una red configurable a nivel dos a un posible usuario final. Existen problemas de aplicación debido a incompatibilidad del protocolo con las máquinas virtuales (VMs) que se utilizarán como *endpoints* en las redes virtuales de FEDERICA.

- CoS/QoS: También se han llevado estudios paralelos a la implementación sobre este asunto. A nivel dos se puede ofrecer, más que *Quality of* Service (Calidad de Servicio), Class *of Service* (Clase de Servicio). Es decir, clasificación de los paquetes de una determinada manera según su tipo, tamaño y otras variables.

Las pruebas realizadas sobre este punto y el anterior están detalladas en el anexo B que se encuentra al final del documento. Se incorpora a la memoria de esta manera debido a que finalmente no han sido implementados.

- Seguridad: Actualmente el único nivel de seguridad implantado en la aplicación es el de la conexión con el switch, el cual está protegido mediante usuario y contraseña. Globus Toolkit tiene una serie de módulos de seguridad por tal de salvaguardar el acceso a los Web Services. La aplicación de las medidas es sencilla y se establecerán en un momento más avanzado de la elaboración del sistema. Alguna de las medidas de seguridad sería, por ejemplo, la posibilidad de incluir certificados, intermediarios o *passwords* privilegiados.

8 Planificación

Dado que el PFC que aquí presento forma parte de un proyecto mucho más grande, sólo tengo en cuenta para el cálculo de la planificación el trabajo desarrollado que consta en el documento. Este factor hace que sea complicado definir con exactitud el coste completo del proyecto (temporal y económico). Además, paralelamente a este cometido, he colaborado en otro de los grupos de FEDERICA, con lo cual el tiempo invertido durante estos meses no lo he dedicado exclusivamente al desarrollo de la aplicación.

En el siguiente diagrama de GANTT se muestra la planificación del PFC detallando cada una de sus fases y el tiempo en el que han sido realizadas. Se ve cómo en los cuatro primeros meses los módulos de trabajo son más dilatados: la fase de estudio, implementación del engine y de los Web Services… En los siguientes cuatro meses las tareas son menos extensas y se superponen por tal de cumplir los objetivos en las fechas establecidas. Por ejemplo, la demostración de la semana ocho del presente año. También se debe a que el ritmo de trabajo era más continuo por la experiencia obtenida con las herramientas con las que se desarrollan las tareas. Esta memoria se ha ido escribiendo poco a poco a partir de inicios de marzo hasta principios de julio. A parte de esto, y también en conjunción con el proyecto, se ha ido redactando durante los ocho meses una serie de documentación para FEDERICA que no consta en el GANTT debido a que no influye en el trabajo presentado; no obstante, se ha echado mano de ella para redactar alguna de las partes del estudio.

Figura 31: Diagrama GANTT del PFC

9 Conclusiones

El resultado que se obtiene del trabajo que se describe en este documento es, principalmente, haber conseguido desarrollar una aplicación cuya funcionalidad es fundamental para el proyecto FEDERICA que lo contiene: La administración de los switchs. En este punto, se pueden configurar los dispositivos de nivel 2 de la infraestructura. Gracias a esta aplicación es posible controlar, desde cualquier ubicación y de manera centralizada, la configuración de los puertos (físicos y lógicos) y las LAN virtuales. Esto permite conducir el flujo de información que pasa por los switchs por tal de crear los diferentes *slices* de los que dispondrán los usuarios.

Esta funcionalidad estará integrada en una aplicación mucho más grande en la que se podrán configurar el resto de elementos implicados en la red, como pueden ser las máquinas virtuales, routers, datos de los usuarios finales etc.

No es trivial el modo en el que se ha conseguido una aplicación que puede ser usada "desde cualquier ubicación y de manera centralizada". Se ha seguido un modelo de desarrollo que actualmente está en boga (IaaS) y que incluye gran cantidad de módulos. Esto se ha conseguida echando mano de un conjunto de sistemas y protocolos (NetConf, Web Services…), de manera que queda todo perfectamente cohesionado consiguiendo lograr de esta manera los objetivos planteados al inicio de la memoria.

Otra conclusión interesante que se obtiene del resultado de este PFC es que el hecho de que la aplicación está enfocada a los investigadores del nuevo Internet, actúa como incentivo para conseguir las metas propuestas y fomenta el estudio de los elementos necesarios por tal de que dicho usuario final goce de las comodidades apropiadas para realizar sus pesquisas en un entorno estable con el fin de facilitar, en la medida de lo posible, el cumplimiento satisfactorio de sus propios objetivos.

Estas valoraciones y conclusiones globales son fruto de la conjunción de los resultados obtenidos de la elaboración de cada uno de los módulos descritos en el desarrollo del documento. Dichos módulos aportan su granito de arena mediante soluciones más concretas. Por lo tanto, del desglose del marco general del PFC se

obtienen a su vez valoraciones y conclusiones específicas para cada uno de sus elementos:

- Engine: Esta pieza de la aplicación es esencial para la comunicación con los dispositivos. Tiene un gran potencial ya que ofrece un perfil bajo el cual se pueden representar todo tipo de dispositivos de red. En este PFC se ha implementado lo necesario para gobernar switchs Juniper EX-3200. Una de sus principales cualidades (y el resultado obtenido lo certifica) es que, realizando pocos cambios (básicamente en los comandos), es sencillo adaptarlo a dispositivos de nivel dos de otros fabricantes y modelos.

  Por otro lado, el hecho de que ya lleva implementados algunos protocolos y transportes para la transferencia de datos ha facilitado en gran medida el trabajo.

  En definitiva, este módulo aporta al conjunto de la aplicación una manera eficaz de operar, configurar y describir los recursos de los dispositivos físicos.

- WS: Los servicios Web de la aplicación permiten acceder a ella remotamente. Están situados en un servidor que los aloja junto con el engine. El resultado es que, mediante las operaciones publicadas y descritas en los ficheros WSDL, se pueden llevar a cabo todos los casos de uso descritos en la memoria.

  Además, estos WS son capaces de mantener información persistente. Es decir, se puede guardar el estado de los switchs y de sus recursos en una base de datos para recuperar las configuraciones cuando sea necesario; por ejemplo, cada vez que se inician los servicios. Sin esta funcionalidad añadida, los Web Services no cumplirían los requisitos necesarios para el proyecto.

- GUI: Este módulo de la aplicación unifica bajo la misma interfaz gráfica un conjunto básico de las operaciones que son capaces de realizar los dos módulos anteriores. Tiene la virtud de centralizar la administración de los dispositivos de la infraestructura. Virtud que será aun mayor cuando la aplicación sea integrada dentro de la interfaz general de FEDERICA (*tool-bench*).

  La conclusión obtenida es que, mediante una interfaz simple, se han podido realizar demos, tests y pruebas muy útiles para mostrar y sopesar la calidad de los resultados.

Hay que decir que estos resultados y valoraciones que se extraen del trabajo son, desde el punto de vista de FEDERICA, parciales. Queda mucho por estudiar y por implementar a lo largo del año y medio que resta de proyecto. No obstante, los objetivos del PFC están cumplidos y forman parte de un *checkpoint* dentro del cómputo global de FEDERICA. De esta base partirán nuevas funcionalidades y por ende nuevas metas.

Para acabar con la evaluación de los resultados del PFC y dar paso a la valoración personal, puedo concluir, ensamblando de nuevo los módulos en un concepto integral, que el proyecto aquí descrito cumple con los requisitos necesarios para ejecutar satisfactoriamente su propósito.

El proyecto FEDERICA me está aportando gran cantidad de conocimientos. No solo informáticos; si no de telecomunicaciones, comunicativos, lingüísticos…

Durante los estudios de los switchs he aprendido conceptos que se han visto muy por encima a lo largo la carrera. Es el caso, por decir alguno, del funcionamiento de las LANs virtuales; donde he debido profundizar por tal de implementar el engine. Q-in-Q, CoS y otros aspectos a nivel dos han ampliado mis conocimientos de la red. Es positivo ver cómo éstos abarcan cada vez más superficie de la pila ISO/OSI.

Por otro lado, el hecho de formar parte de un proyecto europeo tan grande ha enriquecido mis capacidades de visión general. Aunque la materia que aquí se presenta es un módulo muy específico, es necesario ir leyendo los documentos sobre el resto de actividades que se están realizando. Esto implica que haya comunicación entre los grupos de trabajo con lo que mis aptitudes de trabajo en equipo han mejorado sustancialmente.

Todos los procesos de comunicación descritos en el párrafo anterior, tanto orales como escritos, se han realizado, por su condición de proyecto europeo, en inglés. No me ha resultado difícil desenvolverme con cierta soltura ya que durante la carrera y por cuenta propia he trabajado mucho en este idioma. No obstante, sí que se han visto mejoradas mis habilidades comprensivas tanto a nivel oral (video conferencias y *general meetings*) como escrito (documentos y mails).

En cuanto a mis conocimientos informáticos, se han visto continuamente comprometidos, pues la elaboración de cada una de las partes representaba un reto a superar. De todos ellos ya partía con cierta base, pero la elaboración de cada uno

implicaba dar un paso más. La implementación del engine tal vez sea la parte que me ha resultado más sencilla debido a que la base que proporciona la carrera en Java (al menos en nuestra generación) es muy estable.

Para la confección de los Web Services, he tenido que aprender prácticamente desde cero. Esto es debido a que se trata de unos servicios con ciertas particularidades propias de Globus Toolkit (sobretodo para la gestión de la persistencia de datos) y a que durante los estudios solo los he visto en una práctica de una asignatura. La parte positiva es que, pese a las particularidades mencionadas, la implementación es muy similar sea cual sea el objetivo de los WS ya que están fuertemente estandarizados; cosa que implica que he cogido más soltura para el desarrollo de aplicaciones de este tipo.

Para la GUI, ha sido suficiente con echar un vistazo a tutoriales relativamente básicos de RCP (Rich Client Platform) para implementarla. La parte más entretenida y complicada ha sido integrar en ella el engine (en el primer paso) y los servicios Web (en el segundo). Es gratificante ver cómo la aplicación va tomando forma, cada vez más integrada y sólida y evoluciona de manera positiva hasta la obtención del resultado final que se ha descrito.

Durante la elaboración de todo ello he utilizado una gran cantidad de herramientas, muchas de las cuales desconocidas para mí hasta la fecha; es por ello que he creído oportuno nombrarlas en el anexo A de este documento.

Puedo decir que, echando la vista atrás y releyendo el trabajo que contiene estas conclusiones, tanto los objetivos del proyecto como los personales han sido satisfechos. Este hecho proporciona una serie de motivaciones que dan pie a que los que quedan por venir también sean alcanzados.

10 Glosario

CLI (Command Line Interface): Intérprete de órdenes. Es un programa que actúa como interfaz para comunicar al usuario con el Sistema Operativo mediante comandos.

CoS (Class of Service): Términos ingles para designar Clase de Servicio.

Driver: Controlador de dispositivos. Es un programa que permite interactuar con un periférico haciendo una abstracción del mismo y proporcionando una interfaz para gestionarlo.

Framework: Es una estructura de soporte definida sobre la cual se desarrolla y organiza otro proyecto.

GUI (Graphical User Interface): Interfaz gráfica.

IP (Internet Protocol): Es un protocolo no orientado a la conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

Middleware: Software que conecta dos aplicaciones diferentes por separado (por ejemplo una base de datos con un servidor de la Web).

MPLS (Multiprotocol Label Switch): Protocolo de encaminamiento. Opera entre la capa de red y de la de enlace del modelo OSI que permite conmutar caminos mediante etiquetas.

NetConf: Protocolo basado en XML cuya función es proporcionar mecanismos para instalar, manipular y eliminar la configuración de dispositivos de red.

NOC (Network Operation Center): Centro de control de la red. Uno o más sitios desde los cuales se efectúa el control de las redes.

PoP (Point of Presence): Localización física que puede hospedar routers, switchs, máquinas virtuales etc.

QoS (Quality of Service): Términos ingleses para designar calidad de servicio.

REST (REpresentational State Transfer): Técnica de arquitectura de software para sistemas distribuidos; como por ejemplo la World Wide Web.

Router: Dispositivo hardware para la conexión de redes de computadoras que opera en la capa de red.

Slice: Literalmente rodaja o porción. Se utiliza para referirse a la red virtualizada sobre la que los usuarios finales de FEDERICA realizarán sus experimentos.

SOAP (Simple Object Access Protocol): Protocolo estándar que define cómo se comunican dos procesos por medio de intercambio de datos en XML.

Software: Conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware).

SSH (Secure Shell): Intérprete de órdenes seguro que sirve para acceder a máquinas remotas a través de una red.

Switch: Dispositivo hardware para la conexión de redes de computadoras que opera en la capa de enlace.

Trunk: Designa una conexión de red (puerto) que transporta múltiples VLANs.

VLAN: LAN virtual. Método para crear redes lógicamente independientes dentro de la misma red física.

Web Service: es una colección de protocolos y estándares que sirve para intercambiar datos entre aplicaciones de manera transparente a las aplicaciones software que deseen comunicarse.

Wizard: Elemento de una interfaz formado por varios panales que guían la usuario a través de ellos por tal de llevar a cabo cierta configuración.

WSDL (Web Service Description Language): Formato XML para describir servicios Web.

XML (eXtended Markup Language): Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

XML-Schema: Documento XML que describe la estructura y contenido de otro documento XML proporcionando así un nivel alto de abstracción.

11 Bibliografía

FEDERICA project: http://www.fp7-federica.eu/

GanttProject: http://www.ganttproject.biz/

GForge: http://lluna.argia.i2cat.net/gf/

Globus Toolkit:

http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.2.1.html

http://www.globus.org/

IaaS Framework: http://www.iaasframework.com/

Juniper: http://www.juniper.net/us/en/

Rich Client Platform: http://wiki.eclipse.org/index.php/Rich_Client_Platform

StarUML: http://staruml.sourceforge.net/docs/user-guide(en)/toc.html

VMware: http://www.vmware.com/support/

XML: http://www.w3schools.com/xml/

Información general:

http://www.google.com

http://en.wikipedia.org/wiki/Main_Page

http://www.rae.es/rae.html

http://www.wordreference.com/

12 Anexo A – Herramientas utilizadas

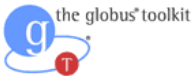
Apache Ant: Herramienta usada para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es similar a *make* pero independiente del sistema operativo sobre el que se utiliza.

Eclipse: Es un entorno de desarrollo integrado de código abierto multiplataforma para programar "Aplicaciones de Cliente Enriquecido". Se usa para crear entornos de desarrollo integrados (IDE); como el de Java, llamado Java Development Toolkit (JDK).

Fedora (Linux): Es una distribución de GNU/Linux para propósitos generales que se mantiene gracias a una comunidad internacional de ingenieros, diseñadores gráficos y usuarios que informan de fallos y prueban nuevas tecnologías. Cuenta con el respaldo y la promoción de Red Hat.

Firefox: Navegador de Internet libre y de código abierto; desarrollado por la Corporación Mozilla, la Fundación Mozilla y un gran número de voluntarios externos. Es el segundo navegador más popular en todo el mundo, después de Internet Explorer. Incluye navegación por pestañas, corrector ortográfico, búsqueda progresiva, entre otras muchas funcionalidades.

GanttProject: Herramienta para la planificación y la administración de proyectos. Funciona sobre Windows, Linux y MacOS. Es gratuito y de código abierto.



Globus toolkit: Software libre para construir *grids* (parrillas) computacionales desarrollado por la Globus Alliance.

Hibernate: Herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones. Es software libre; distribuido bajo los términos de la licencia GNU LGPL.

IaaS Framework: Es un ágil entorno de desarrollo para la creación de recursos Infrastructure as a Service. Simplifica la elaboración de aplicaciones Web dinámicas ofreciendo capacidad de agregación en Java y Groovy. Está optimizado para crear representaciones de dispositivos hardware mediante servicios de Web Service o REST.

Java: Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. A diferencia de C y C++, tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, las cuales suelen conducir a muchos errores, como la manipulación directa de punteros y memoria. Su máquina virtual de proceso hace que sea totalmente independiente de la plataforma; una de sus mayores cualidades (“write once, run everywhere”).

JUnit: Es un conjunto de bibliotecas que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

JUNOS (Juniper): Es el sistema operativo usado por los dispositivos Juniper. Facilita el manejo de éstos tanto para Juniper como para el usuario unificando tanto routers como switchs en el mismo sistema operativo.

Log4j: Utilidad Java para el control de los *logs*. Permite crear ficheros de éste tipo así como escribir los mensajes directamente por consola, ubicarlos en un servidor, entre otras cosas. Es una herramienta muy versátil.

pgAdmin: Es una plataforma de software libre cuyo objetivo es la administración y desarrollo para PostgreSQL.

PostgreSQL: Es un sistema de gestión de base de datos relacional orientada a objetos de software libre, publicado bajo la licencia BSD.

PuTTy: Cliente SSH, Telnet, rlogin, y TCP raw con licencia libre. Disponible originariamente sólo para Windows, ahora también está en varias plataformas Unix, y se está desarrollando la versión para Mac OS clásico y Mac OS X.

StarUML: Proyecto de código abierto y gratuito para desarrollar diagramas UML/MDA de manera flexible, extensible, y cómoda. Funciona sobre Windows.

Subversion: Es un software de sistema de control de versiones diseñado para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre bajo una

licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos. El *plug-in* de Eclipse que lo gestiona se llama Subclipse.

Ubuntu (Linux): Es una distribución de GNU/Linux tanto para propósitos generales como para proporcionar soporte para servidores. Es una de las distribuciones de este tipo más importantes a nivel mundial.

VMWare: Software de virtualización. Se incluyen VMware Workstation, y los gratuitos VMware Server y VMware Player. El software de VMware puede funcionar en Windows, Linux, y en la plataforma Mac OS X. El nombre corporativo de la compañía es un juego de palabras usando la interpretación tradicional de las siglas VM en los ambientes de computación, como máquinas virtuales (**V**irtual **M**achines).

Windows XP: Sistema operativo desarrollado por Microsoft que se hizo público en octubre de 2001. Las letras XP provienen de la palabra inglesa "eXPeriencie". Es el primer sistema operativo de Microsoft orientado al consumidor que se constituye con núcleo y arquitectura de Windows NT y que se encuentra disponible en versiones para PC de 32 y 64 bits.

WinSCP: Es una aplicación de Software libre. Es un cliente SFTP gráfico para Windows que emplea SSH. Su función principal es facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios SSH.

13 Anexo B – Estudios de Q-in-Q y QoS/CoS

Dado que los estudios sobre Q-in-Q y QoS/CoS no han sido implementados en la aplicación presentada en esta memoria, pero teniendo en cuenta que de todos modos es un trabajo que se ha estado realizando durante el proyecto y que muy probablemente deba ser integrado en un futuro próximo, se ha decidido plasmar los resultados obtenidos en este anexo, cuyo objetivo es resumir las pruebas realizadas al respecto.

1 Q-in-Q

Los túneles Q-in-Q permiten a los proveedores de servicios en las redes Ethernet extender una conexión de nivel dos clientes. Además, Q-in-Q también se utiliza para separar el tráfico de los clientes en diferentes VLANs. Este proceso se lleva a cabo aplicado a los paquetes Ethernet un segundo nivel de etiquetas 802.1Q. Este servicio es útil para que dos clientes puedan utilizar dos identificadores de VLAN iguales (pero mantenerse independientes unos de otros). Con lo cual, una VLAN cliente (C-VLAN) puede estar precedida por la etiqueta de una de servicio (S-VLAN).

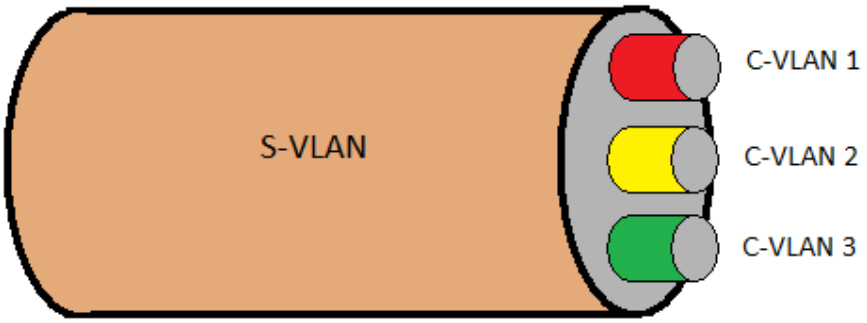


Ilustración 1: Representación de la relación entre S-VLANs y C-VLANs

Desde el punto de vista de FEDERICA, la S-VLAN correspondería a las virtual LANs configuradas en la infraestructura por tal de mantener su conectividad interna; mientras que las C-VLANs serían las que configura cada usuario final dentro de su *slice*.

1.1 Funcionamiento

Un usuario (o cliente) puede configurar una VLAN, la cual se etiqueta mediante el protocolo 801.1Q. Cuando el paquete viaja desde esta C-VLAN por la red del proveedor, se añade otra etiqueta perteneciente a su virtual LAN (S-VLAN). De esta manera, se conserva la etiqueta del cliente y se transmite de forma transparente a través de la red del proveedor.

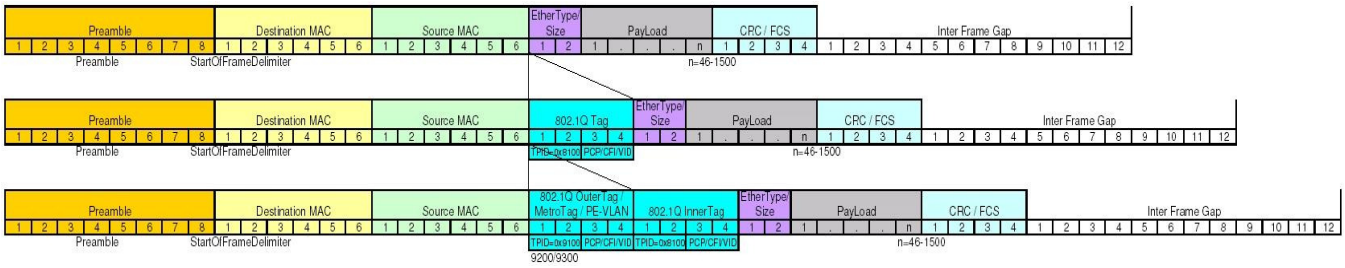


Ilustración 2: Inserción de doble etiquetado 802.1Q (Q-in-Q) en un paquete Ethernet

Cuando se activa Q-in-Q, las interfaces *trunk* forman parte de la red del proveedor y las interfaces de acceso (access) están destinadas a los usuarios (clientes). Cada S-VLAN puede contener varias C-VLANs (como se ve en la ilustración 1) de manera que, como están a diferentes niveles, pueden tener el mismo identificador.

En lo que respecta a los dispositivos EX-3200, las S-VLANs se añaden a las interfaces *access* para los paquetes entrantes. Este tipo de interfaces pueden recibir tanto paquetes etiquetados como sin etiquetar. Por lo tanto, cuando los paquetes abandonan la S-VLAN, se elimina también su etiqueta, con lo que la etiqueta original del cliente permanece inalterada en el paquete.

Por otro lado, existen algunas limitaciones en los switchs EX-3200. No se puede añadir una etiqueta de una C-VLAN en un puerto *access* si el paquete entrante no viene etiquetado. Además, no soporta ciertas condiciones de seguridad para los puertos.

1.2 Configuración

Para configurar Q-in-Q en el dispositivo es necesario que, previamente, se tengan configuradas las C-VLANs necesarias. Por lo tanto, una posible configuración sería la siguiente: dos puertos *trunk* para las S-VLANs (ge-0/0/15.0 y ge-0/0/18.0). Por otro lado, otras dos interfaces en modo acceso y sin etiquetar para los clientes (ge-0/0/16.0 y ge-0/0/17.0). Los comandos, vía CLI, que se deben introducir quedan de la siguiente manera:

Crear S-VLAN:

```
user@switch# set vlans qqVlan vlan-id 1234
```

Habilitar Q-in-Q y los rangos de las VLANs de usuario:

```
user@switch# set vlans qqVlan dot1q-tunneling customer-vlans 1-100
user@switch# set vlans qqVlan dot1q-tunneling customer-vlans 201-300
```

Configuración de los puertos, vlans y Q-inQ:

```
user@switch# set interfaces ge-0/0/15 unit 0 family ethernet-
   switching port-mode trunk
user@switch# set vlans qqVlan interfaces ge-0/0/15.0
user@switch# set interfaces ge-0/0/16 unit 0 family ethernet-
   switching port-mode access
user@switch# set vlans qqVlan interfaces ge-0/0/16.0
user@switch# set interfaces ge-0/0/17 unit 0 family ethernet-
   switching port-mode access
user@switch# set vlans qqVlan interfaces ge-0/0/17.0
user@switch# set interfaces ge-0/0/18 unit 0 family ethernet-
   switching port-mode trunk
user@switch# set vlans qqVlan interfaces ge-0/0/18.0
user@switch# set ethernet-switching-options dot1q-tunneling
   ether-type 0x9100
```

Para comprobar que la configuración se ha aplicado satisfactoriamente:

```
user@switch> show configuration vlans qinqvlan
vlan-id 1234;
      dot1q-tunneling {
      customer-vlans [ 1-100 201-300 ];
      interface {
            ge-0/0/15.0;
            ge-0/0/16.0;
            ge-0/0/17.0;
            ge-0/0/18.0;
      }
}
```

1.3 Posible implementación

FEDERICA proporcionará a los usuarios máquinas virtuales (VMs), alojadas en servidores (VMSs), para operar con sus redes. Esto es una limitación desde el punto de vista de Q-in-Q ya que éstas no lo implementan. Es por ello que se ha llevado a cabo un estudio para encontrar posibles soluciones. Se presenta la topología siguiente:

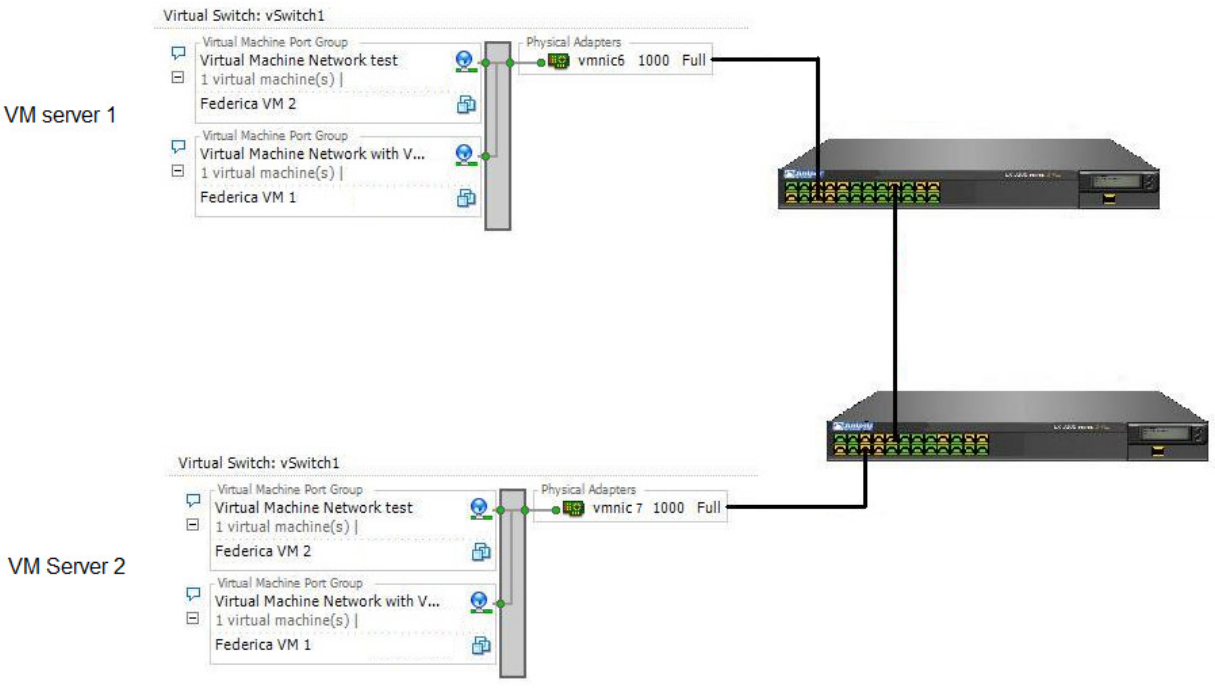


Ilustración 3: Topología del test de Q-in-Q sobre máquinas virtuales

Se pretende configurar un *slice* que consta de dos máquinas virtuales, cada una de ellas conectada a una tarjeta de red física y en *port groups* diferentes. Los switchs físicos las conectan. Con esta situación, el NOC debe configurar la VLAN de servicio (S-VLAN) para el usuario final 1. Debe seguir los siguientes pasos:

1. Crear el *port group* en el servidor VM 1, asignarle VLAN ID 1 y añadirle la VM1. Esto es equivalente a tener una máquina que acepta *frames* de la S-VLAN 1.
2. Hacer lo mismo en el servidor VM 2 asignándole la misma S-VLAN.
3. Crear la S-VLAN en los switchs con identificador 1.
4. Asignarle a la S-VLAN los puertos de los dispositivos físicos a los que se conectan las VMs.
5. Configurar a *trunk* las interfaces que conectan los switchs físicos entre sí y añadirlos a la S-VLAN 1.

Después de realizar los pasos anteriores ya está configurado el *slice*. A continuación se configura la C-VLAN (VLAN del cliente):

6. Crear en los switchs Juniper una VLAN destinada al cliente (C-VLAN) con identificador, por ejemplo, 1234.
7. Añadirle los puertos *trunk*.
8. Configurarlos, tal y como se ha visto en el apartado anterior, para permitir Q-in-Q añadiendo una etiqueta adicional (1234) a los paquetes entrantes desde el servidor de máquinas virtuales (VMS) 1; es decir, los que corresponden a la S-VLAN 1.

Si se configura otra C-VLAN, siguiente los tres últimos pasos, para otro usuario final o cliente (es decir, para otro *slice*) se obtiene la situación mostrada en la siguiente ilustración. Los paquetes verdes simbolizan los paquetes de la S-VLAN 1 y los amarillos representan las cabeceras de C-VLAN 1234.

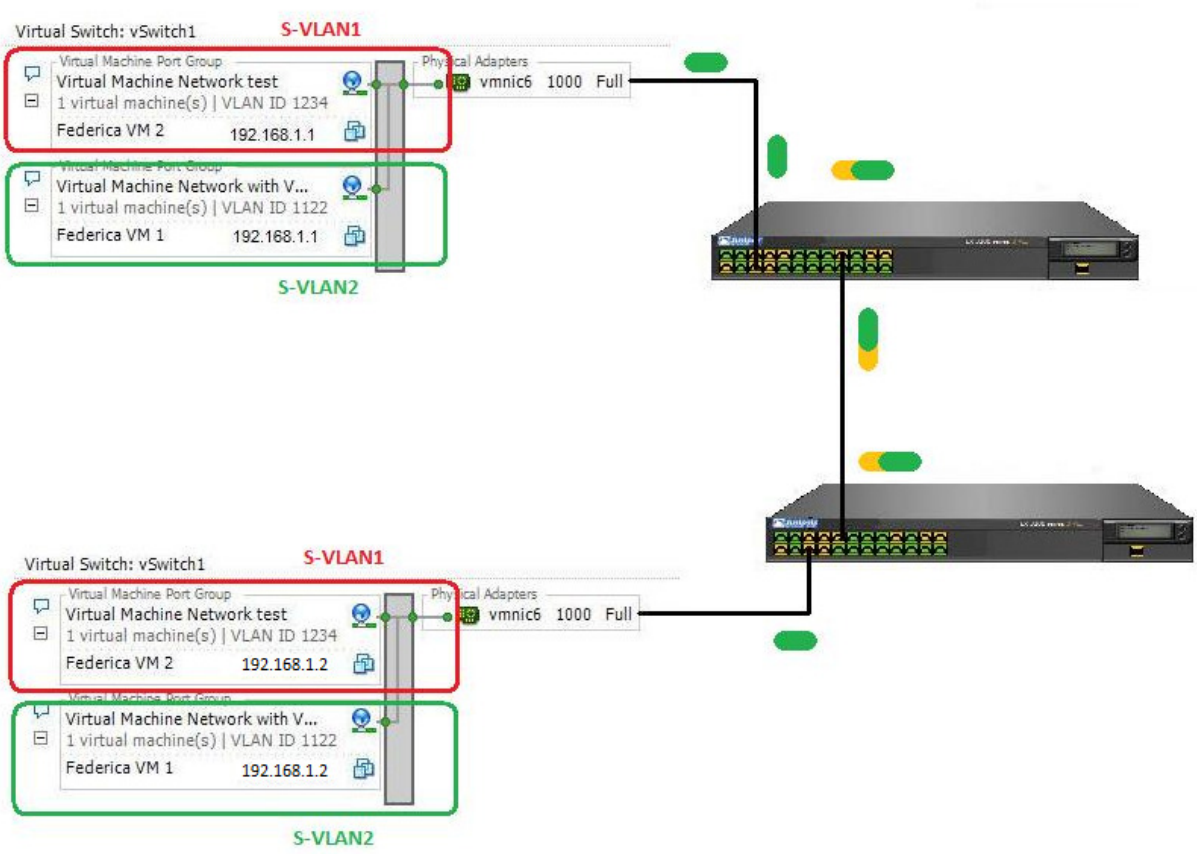


Ilustración 4: Topología y configuración del test de Q-in-Q sobre máquinas virtuales

Este sistema presenta un conjunto de puntos débiles importantes:

- Por tal de realizar correctamente el encapsulado de VLANs de servicio con las de cliente, es necesario que cada tarjeta de red física del VMS contenga una única S-VLAN. Esto es una limitación importante del número de *slices* que podría soportar FEDERICA (problema de escalabilidad).
- No se ha probado el funcionamiento de Q-in-Q entre los switchs Juniper.
- La aplicación del usuario debería configurar los switchs (VLANs, puertos, Q-in-Q) sin que éste pueda ver el resto de S-VLANs.

2 QoS/CoS

A nivel dos, es más apropiado hablar de CoS (Class of Service) que de QoS (Quality of Service) ya que solo se ofrecen procedimientos de este tipo (muy a menudo confundidos con los del segundo). Las medidas de CoS tienen en cuenta el tipo de paquete que se está gestionando. A continuación se describe la clasificación de servicios que se puede configurar sobre los switchs EX-3200 de Juniper.

2.1 Funcionamiento

Como se ha dicho, los switchs EX-3200 implementan algunos mecanismos de CoS. Son los siguientes:

- Clasificadores: La clasificación de paquetes asocia los paquetes entrantes con un nivel de CoS. Los clasificadores asocian paquetes a una *forwarding class* (clase de transferencia) y una prioridad de pérdida a la hora de asignarlos a las diferentes colas de salida. Existen dos tipos de clasificadores:
    - Clasificadores de CoS por valor: Basados en el valor de CoS que se encuentra en la cabecera del paquete.
    - Clasificadores de tráfico multicampo: Basados en diferentes campos del paquete como, por ejemplo, las direcciones MAC (fuente y/o destino).
- *Policers*: Limitan el tráfico de cierta clase basándose en propiedades como el ancho de banda o el *burst size* (número de paquetes recibidos en una ráfaga). Los *policers* se asocian a las interfaces entrantes. Los paquetes que superan el límite establecido pueden ser descartados.
- Clases de transferencia (forwarding classes): Agrupan los paquetes a transmitir. Éstos se añaden a diferentes colas de salida según ciertos parámetros. Para llevarlo a cabo, se aplican a los paquetes políticas de transmisión, programación y marcado. Existen cuatro clases de transferencia principales, si bien los dispositivos son capaces de gestionar dieciséis por tal de mejorar la granularidad de la clasificación:
    - *Best effort* (Mejores condiciones).

- *Assured forwarding* (Seguridad de la transferencia).
- *Expedited forwarding* (Aceleración de la transferencia).
- Control de red.

- *Tail drop* profiles (Perfiles para descartar paquetes de las colas): Es un mecanismo para evitar la congestión del dispositivo. Permite desechar los paquetes entrantes colocados al final de las colas cuando se llenan los *buffers* (o cierto porcentaje de ellos). Cuando existe un cien por cien de probabilidad de pérdida, los paquetes son descartados automáticamente del final de la cola.
- *Schedulers* (Planificadores): Se utilizan para definir las propiedades siguientes de las colas de salida:
  - Cantidad de ancho de banda por interfaz asignado a cierta cola.
  - El tamaño de la memoria del *buffer.*
  - La prioridad de la cola.
  - Los perfiles de descarte asociados a las colas.